

Interrupt 21H (33) Function 3FH (63)

2.0 and later

Read File or Device

Function 3FH reads from the file or device referenced by a handle.

To Call

AH = 3FH
BX = handle number
CX = number of bytes to read
DS:DX = segment:offset of data buffer

Returns

If function is successful:

Carry flag is clear.

AX = number of bytes read from file
DS:DX = segment:offset of data read from file

If function is not successful:

Carry flag is set.

AX = error code:
05H access denied
06H invalid handle

Programmer's Notes

- Data is read from the file beginning at the current location of the file pointer. After a successful read, the file pointer is updated to point to the byte following the last byte read.
- If Function 3FH returns 00H in the AX register, the function attempted to read when the file pointer was at the end of the file. If AX is less than CX, a partial record at the end of the file was read.
- Function 3FH can be used with all handles, including standard input (normally the keyboard). When reading from standard input, this function normally reads characters only to the first carriage-return character. Thus, the number of bytes read in AX will not necessarily match the length requested in CX.
- On networks running under MS-DOS version 3.1 or later, the user must have Read access to the directory and file containing the information to be read.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

40H (Write File or Device)
 42H (Move File Pointer)
 59H (Get Extended Error Information)

Example

```

;*****;
;
;      Function 3FH: Read File or Device
;
;      int read(handle,pbuffer,nbytes)
;          int handle,nbytes;
;          char *pbuffer;
;
;      Returns -1 if there was a read error,
;      otherwise returns number of bytes read.
;
;*****;

cProc  read,PUBLIC,ds
parmW  handle
parmDP pbuffer
parmW  nbytes
cBegin
mov    bx,handle      ; Get handle.
loadDP ds,dx,pbuffer ; Get pointer to buffer.
mov    cx,nbytes      ; Get number of bytes to read.
mov    ah,3fh         ; Set function code.
int    21h            ; Ask MS-DOS to read CX bytes.
jnb   rd_ok           ; Branch if read worked.
mov    ax,-1          ; Else return -1.

rd_ok:
cEnd

```

Interrupt 21H (33) Function 40H (64)

2.0 and later

Write File or Device

Function 40H writes the specified number of bytes to a file or device referenced by a handle.

To Call

AH = 40H
BX = handle
CX = number of bytes to write
DS:DX = segment:offset of data buffer

Returns

If function is successful:

Carry flag is clear.

AX = number of bytes written to file or device

If function is not successful:

Carry flag is set.

AX = error code:
05H access denied
06H invalid handle

Programmer's Notes

- Data is written to the file or device beginning at the current location of the file pointer. After writing the specified data, Function 40H updates the position of the file pointer and returns the actual number of bytes written in AX.
- Function 40H returns error code 05H (access denied) if the file was opened as read-only with Function 3CH (Create File with Handle), 3DH (Open File with Handle), 5AH (Create Temporary File), or 5BH (Create New File). On networks running under MS-DOS version 3.1 or later, access is also denied if the file or record has been locked by another process.
- The handle number in BX must be one of the predefined device handles (0 through 4) or a handle obtained through a previous call to open or create a file (such as Function 3CH, 3DH, 5AH, or 5BH).
- If CX = 0, the file is truncated or extended to the current file pointer location. Clusters are allocated or released in the file allocation table (FAT) as required to fulfill the request.

- If the handle parameter for Function 40H refers to a disk file and the number of bytes written (returned in AX) is less than the number requested in CX, the destination disk is full. The carry flag is *not* set in this situation.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

3FH (Read File or Device)

42H (Move File Pointer)

Example

```

;*****;
;
;           Function 40H: Write File or Device
;
;           int write(handle,pbuffer,nbytes)
;               int handle,nbytes;
;               char *pbuffer;
;
;           Returns -1 if there was a write error,
;           otherwise returns number of bytes written.
;
;*****;

cProc  write,PUBLIC,ds
parmW  handle
parmDP pbuffer
parmW  nbytes
cBegin
    mov     bx,handle      ; Get handle.
    loadDP ds,dx,pbuffer  ; Get pointer to buffer.
    mov     cx,nbytes     ; Get number of bytes to write.
    mov     ah,40h        ; Set function code.
    int     21h           ; Ask MS-DOS to write CX bytes.
    jnb    wr_ok          ; Branch if write successful.
    mov     ax,-1         ; Else return -1.

wr_ok:
cEnd

```

Interrupt 21H (33) Function 41H (65)

2.0 and later

Delete File

Function 41H deletes the directory entry of the specified file.

To Call

AH = 41H
DS:DX = segment:offset of ASCIIZ pathname

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
02H file not found
03H path not found
05H access denied

Programmer's Notes

- The pathname must be a null-terminated ASCII string (ASCIIZ). Unlike Function 13H (Delete File), Function 41H does not allow wildcard characters in the pathname.
- Because Function 41H supports the use of full pathnames, it is preferable to Function 13H.
- Function 41H returns error code 05H (access denied) and fails if the file has either a directory or volume attribute or if it is a read-only file.

A directory can be deleted (if it is empty) with Function 3AH (Remove Directory). A read-only file can be deleted if its attribute is changed to normal with Function 43H (Get/Set File Attributes) before Function 41H is called.

- On networks running under MS-DOS version 3.1 or later, the user must have Create access to the directory containing the file to be deleted.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

3AH (Remove Directory)
43H (Get/Set File Attributes)

Example

```
*****;  
;  
;           Function 41H: Delete File           ;  
;  
;           int delete(pfilepath)               ;  
;           char *pfilepath;                   ;  
;  
;           Returns 0 if file deleted,         ;  
;           otherwise returns error code.      ;  
;  
*****;  
  
cProc   delete,PUBLIC,ds  
parmDP  pfilepath  
cBegin  
        loadDP  ds,dx,pfilepath ; Get pointer to pathname.  
        mov     ah,41h          ; Set function code.  
        int     21h             ; Ask MS-DOS to delete file.  
        jb     dl_err          ; Branch if MS-DOS could not delete  
;                               ; file.  
        xor     ax,ax           ; Else return 0.  
  
dl_err:  
cEnd
```

Interrupt 21H (33) Function 42H (66)

2.0 and later

Move File Pointer

Function 42H sets the position of the file pointer (for the next read/write operation) for the file associated with the specified handle.

To Call

AH	=	42H
AL	=	method code:
	00H	byte offset from beginning of file
	01H	byte offset from current location of file pointer
	02H	byte offset from end of file
BX	=	handle number
CX:DX	=	offset value to move pointer:
	CX	most significant half of a doubleword value
	DX	least significant half of a doubleword value

Returns

If function is successful:

Carry flag is clear.

DX:AX = new file pointer position (absolute byte offset from beginning of file)

If function is not successful:

Carry flag is set.

AX	=	error code:
	01H	invalid function (AL not 00H, 01H, or 02H)
	06H	invalid handle

Programmer's Notes

- The value in CX:DX is an offset specifying how far the file pointer is to be moved. With method code 00H, the value in CX:DX is always interpreted as a positive 32-bit integer, meaning the file pointer is always set relative to the beginning of the file. With method codes 01H and 02H, the value in CX:DX can be either a positive or negative 32-bit integer. Thus, method 1 can move the file pointer either forward or backward from its current position; method 2 can move the file pointer either forward or backward from the end of the file.

- Specifying method code 00H with an offset of 0 positions the file pointer at the beginning of the file. Similarly, specifying method code 02H with an offset of 0 conveniently positions the file pointer at the end of the file. With method code 02H offset 0, the size of the file can also be determined by examining the pointer position returned by the function.
- Depending on the offset specified in CX:DX, methods 1 and 2 may move the file pointer to a position before the start of the file. Function 42H does not return an error code if this happens, but later attempts to read from or write to the file will produce unexpected errors.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

3FH (Read File or Device)

40H (Write File or Device)

Example

```

;*****;
;
;           Function 42H: Move File Pointer
;
;           long seek(handle,distance,mode)
;           int handle,mode;
;           long distance;
;
;           Modes:
;           0: from beginning of file
;           1: from the current position
;           2: from the end of the file
;
;           Returns -1 if there was a seek error,
;           otherwise returns long pointer position.
;
;*****;

```

```

cProc   seek,PUBLIC
parmW   handle
parmD   distance
parmB   mode
cBegin
mov     bx,handle       ; Get handle.
les    dx,distance     ; Get distance into ES:DX.
mov    cx,es           ; Put high word of distance into CX.
mov    al,mode         ; Get move method code.
mov    ah,42h          ; Set function code.

```

(more)


```
int      21h          ; Ask MS-DOS to move file pointer.
jnb     sk_ok        ; Branch if seek successful.
mov     ax,-1        ; Else return -1.
cwd

sk_ok:
cEnd
```

Interrupt 21H (33) Function 43H (67)

2.0 and later

Get/Set File Attributes

Function 43H gets or sets the attributes of the specified file.

To Call

AH = 43H

To get file attributes:

AL = 00H

DS:DX = segment:offset of ASCII pathname

To set file attributes:

AL = 01H

CX = attributes to set:

Bit	Attribute
0	Read-only file
1	Hidden file
2	System file
5	Archive

DS:DX = segment:offset of ASCII pathname

Returns

If function is successful:

Carry flag is clear.

CX = attribute

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function (AL not 00H or 01H)
 02H file not found
 03H path not found
 05H access denied

Programmer's Notes

- The pathname must be a null-terminated ASCII string (ASCIIZ).
- Function 43H cannot be used to set or change either a volume-label or directory attribute (bits 3 and 4 of the attribute byte). With MS-DOS versions 3.x, Function 43H can be used to make a directory hidden or read-only.
- On networks running under MS-DOS version 3.1 or later, the user must have Create access to the directory containing the file in order to change the read-only, hidden, or system attribute. The archive bit, however, can be changed regardless of access rights.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;           Function 43H: Get/Set File Attributes
;
;           int file_attr(pfilepath,func,attr)
;           char *pfilepath;
;           int func,attr;
;
;           Returns -1 for all errors,
;           otherwise returns file attribute.
;
;*****;

cProc   file_attr,PUBLIC,ds
parmDP  pfilepath
parmB   func
parmW   attr
cBegin
        loadDP  ds,dx,pfilepath ; Get pointer to pathname.
        mov     al,func          ; Get/set flag into AL.
        mov     cx,attr         ; Get new attr (if present).
        mov     ah,43h          ; Set code function.
        int     21h             ; Call MS-DOS.
        jnb    fa_ok           ; Branch if no error.
        mov     cx,-1          ; Else return -1.
fa_ok:
        mov     ax,cx          ; Return this value.
cEnd

```

Interrupt 21H (33) Function 44H (68)

2.0 and later

IOCTL

Function 44H is a collection of subfunctions that provide a process a direct path of communication with a device driver. As such, this function is the most flexible means of gaining access to the full capabilities of an installed device.

An IOCTL subfunction is called with 44H in AH and the value for the subfunction in AL. If a subfunction has minor functions, those values are specified in CL. Otherwise, the BX, CX, and DX registers are used for such information as handles, drive identifiers, buffer addresses, and so on.

The subfunctions and the versions of MS-DOS with which they are available are

Subfunction	Name	MS-DOS Versions
00H	Get Device Data	2.0 and later
01H	Set Device Data	2.0 and later
02H	Receive Control Data from Character Device	2.0 and later
03H	Send Control Data to Character Device	2.0 and later
04H	Receive Control Data from Block Device	2.0 and later
05H	Send Control Data to Block Device	2.0 and later
06H	Check Input Status	2.0 and later
07H	Check Output Status	2.0 and later
08H	Check If Block Device Is Removable	3.0 and later
09H	Check If Block Device Is Remote	3.1 and later
0AH	Check If Handle Is Remote	3.1 and later
0BH	Change Sharing Retry Count	3.1 and later
0CH	Generic I/O Control for Handles	3.2
	Minor Code 45H: Set Iteration Count	
	Minor Code 65H: Get Iteration Count	
0DH	Generic I/O Control for Block Devices	3.2
	Minor Code 40H: Set Device Parameters	
	Minor Code 60H: Get Device Parameters	
	Minor Code 41H: Write Track on Logical Drive	
	Minor Code 61H: Read Track on Logical Drive	
	Minor Code 42H: Format and Verify Track on Logical Drive	
	Minor Code 62H: Verify Track on Logical Drive	

(more)

Subfunction	Name	MS-DOS Versions
0EH	Get Logical Drive Map	3.2
0FH	Set Logical Drive Map	3.2

These subfunctions are documented, either individually or in related pairs, in the entries that follow.

Interrupt 21H (33) Function 44H (68) Subfunction 00H

2.0 and later

IOCTL: Get Device Data

Function 44H Subfunction 00H gets information about a character device or file referenced by a handle.

To Call

AH = 44H
AL = 00H
BX = handle number

Returns

If function is successful:

Carry flag is clear.

DX contains information on file or device:

Bit	Value	Meaning
For a file (bit 7 = 0):		
8-15	0	Reserved.
7	0	Handle refers to a file.
6	0	File has been written.
0-5		Drive number (0 = A, 1 = B, 2 = C, and so on).
For a device (bit 7 = 1):		
15	0	Reserved.
14	1	Processes control strings transferred by IOCTL Subfunctions 02H (Receive Control Data from Character Device) and 03H (Send Control Data to Character Device), set by MS-DOS.
8-13	0	Reserved.
7	1	Handle refers to a device.
6	0	End of file on input.
5	0	Checks for control characters (cooked mode).
	1	Does not check for control characters (raw mode).

(more)

Bit	Value	Meaning
4	0	Reserved.
3	1	Clock device.
2	1	Null device.
1	1	Standard output device.
0	1	Standard input device.

If function is not successful:

Carry flag is set.

AX = error code:

01H	invalid IOCTL subfunction
05H	access denied
06H	invalid handle

Programmer's Notes

- Bits 8–15 of DX correspond to the upper 8 bits of the device-driver attribute word.
- The handle in BX must reference an open device or file.
- Bit 5 of the device data word for character-device handles defines whether that handle is in raw mode or cooked mode. In cooked mode, MS-DOS checks for Control-C, Control-P, Control-S, and Control-Z characters and transfers control to the Control-C exception handler (whose address is saved in the vector for Interrupt 23H) when a Control-C is detected. In raw mode, MS-DOS does not check for such characters when I/O is performed to the handle; however, it will still check for a Control-C entered at the keyboard on other function calls unless such checking has been turned off with Function 33H, the BREAK=OFF directive in CONFIG.SYS, or a BREAK OFF command at the MS-DOS prompt.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

33H (Get/Set Control-C Check Flag)
 3CH (Create File with Handle)
 3DH (Open File with Handle)

Example

```

;*****;
;
;      Function 44H, Subfunctions 00H,01H:
;      Get/Set IOCTL Device Data
;
;      int ioctl_char_flags(setflag,handle,newflags)
;      int setflag;
;      int handle;
;      int newflags;
;
;      Set setflag = 0 to get flags, 1 to set flags.
;
;      Returns -1 for error, else returns flags.
;
;*****;

cProc  ioctl_char_flags,PUBLIC
parmB  setflag
parmW  handle
parmW  newflags
cBegin
mov     al,setflag      ; Get setflag.
and     al,1           ; Save only lsb.
mov     bx,handle      ; Get handle to character device.
mov     dx,newflags    ; Get new flags (they are used only
                        ; by "set" option).
mov     ah,44h         ; Set function code.
int     21h           ; Call MS-DOS.
mov     ax,dx          ; Assume success - prepare to return
                        ; flags.
jnc     iocfx          ; Branch if no error.
mov     ax,-1          ; Else return error flag.

iocfx:
cEnd

```


Interrupt 21H (33)

2.0 and later

Function 44H (68) Subfunction 01H

IOCTL: Set Device Data

Function 44H Subfunction 01H, the complement of IOCTL Subfunction 00H, sets information about a character device — but not a file — referenced by a handle.

To Call

AH = 44H
 AL = 01H
 BX = handle number
 DX = device data word:

Bit	Value	Meaning
8–15	0	Reserved.
7	1	Handle refers to a device.
6	0	End of file on input.
5	0	Check for control characters (cooked mode).
	1	Do not check for control characters (raw mode).
4	0	Reserved.
3	1	Clock device.
2	1	Null device.
1	1	Standard output device.
0	1	Standard input device.

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid IOCTL subfunction
 05H access denied
 06H invalid handle

Programmer's Notes

- The handle in BX must reference an open device.
- DH must be 00H. If it is not, the carry flag is set and error code 01H (invalid function) is returned.
- Bit 5 of the device data word for character-device handles selects raw mode or cooked mode for the handle. In cooked mode, MS-DOS checks for Control-C, Control-P, Control-S, and Control-Z characters and transfers control to the Control-C exception handler (whose address is saved in the vector for Interrupt 23H) when a Control-C is detected. In raw mode, MS-DOS does not check for such characters when I/O is performed to the handle; however, it will still check for a Control-C entered at the keyboard on other function calls unless such checking has been turned off with Function 33H, the BREAK=OFF directive in CONFIG.SYS, or a BREAK OFF command at the MS-DOS prompt.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

33H (Get/Set Control-C Check Flag)
3CH (Create File with Handle)
3DH (Open File with Handle)

Example

See SYSTEM CALLS: INTERRUPT 21H: Function 44H Subfunction 00H.

Interrupt 21H (33)

2.0 and later

Function 44H (68) Subfunctions 02H and 03H

IOCTL: Receive Control Data from Character Device; Send Control Data to Character Device

Function 44H Subfunctions 02H and 03H respectively receive and send control strings from and to a character-oriented device driver.

To Call

AH = 44H
AL = 02H receive control strings
 03H send control strings
BX = handle number
CX = number of bytes to transfer
DS:DX = segment:offset of data buffer

Returns

If function is successful:

Carry flag is clear.

AX = number of bytes transferred

If AL was 02H on call:

Buffer at DS:DX contains data read from device driver.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 05H access denied
 06H invalid handle
 0DH invalid data (bad control string)

Programmer's Notes

- Subfunctions 02H and 03H provide a means of transferring control information of any type or length between an application program and a character-device driver. They do not necessarily result in any input to or output from the physical device itself.
- Subfunction 02H can be used to read control information about such features as device status, availability, and current output location. Subfunction 03H is often used to configure the driver or device for subsequent I/O; for example, it may be used to set the baud rate, word length, and parity for a serial communications adapter or to initialize a printer for a specific font, page length, and so on. The format of the control data passed by these subfunctions is driver specific and does not follow any standard.

- Character-device drivers are not required to support IOCTL Subfunctions 02H and 03H. Therefore, Subfunction 00H (Get Device Data) should be called before either Subfunction 02H or 03H to determine whether a device can process control strings. If bit 14 of the device data word returned by Subfunction 00H is set, the device driver supports IOCTL Subfunctions 02H and 03H.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

44H Subfunction 00H (Get Device Data)

44H Subfunction 04H (Receive Control Data from Block Device)

44H Subfunction 05H (Send Control Data to Block Device)

Example

```

;*****;
;
;   Function 44H, Subfunctions 02H,03H:
;           IOCTL Character Device Control
;
;   int ioctl_char_ctrl(recvflag,handle,pbuffer,nbytes)
;       int  recvflag;
;       int  handle;
;       char *pbuffer;
;       int  nbytes;
;
;   Set recvflag = 0 to receive info, 1 to send.
;
;   Returns -1 for error, otherwise returns number of
;   bytes sent or received.
;*****;

cProc  ioctl_char_ctrl,PUBLIC,<ds>
parmB  recvflag
parmW  handle
parmDP pbuffer
parmW  nbytes
cBegin
    mov     al,recvflag      ; Get recvflag.
    and     al,1            ; Keep only lsb.
    add     al,2            ; AL = 02H for receive, 03H for send.
    mov     bx,handle       ; Get character-device handle.
    mov     cx,nbytes       ; Get number of bytes to receive/send.
    loadDP ds,dx,pbuffer   ; Get pointer to buffer.
    mov     ah,44h          ; Set function code.
    int     21h             ; Call MS-DOS.
    jnc     iccx            ; Branch if no error.
    mov     ax,-1           ; Return -1 for all errors.

iccx:
cEnd

```

Interrupt 21H (33)

2.0 and later

Function 44H (68) Subfunctions 04H and 05H

IOCTL: Receive Control Data from Block Device; Send Control Data to Block Device

Function 44H Subfunctions 04H and 05H respectively receive and send control strings from and to a block-oriented device driver.

To Call

AH = 44H
AL = 04H receive block-device data
 05H send block-device data
BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
CX = number of bytes to transfer
DS:DX = segment:offset of data buffer

Returns

If function is successful:

Carry flag is clear.

AX = number of bytes transferred

If AL was 04H on call:

Buffer at DS:DX contains control data read from device driver.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 05H access denied
 06H invalid handle
 0DH invalid data (bad control string)

Programmer's Notes

- Subfunctions 04H and 05H provide a means of transferring control information of any type or length between an application program and a block-device driver. They do not necessarily result in any input to or output from the physical device itself.
- Control strings can be used to request driver operations that are not file oriented, such as tape rewind or disk eject (if hardware supported). The contents of such control strings are specific to individual device drivers and do not follow any standard format.

- Subfunction 04H can be used to obtain a code from the driver indicating device availability or status. Block devices that might use this subfunction include magnetic tape or tape cassette, CD ROM, and Small Computer Standard Interface (SCSI) devices.
- Block-device drivers are not required to support IOCTL Subfunctions 04H and 05H. If the driver does not support these subfunctions, error code 01H (Invalid Function) is returned.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

44H Subfunction 00H (Get Device Data)

44H Subfunction 02H (Receive Control Data from Character Device)

44H Subfunction 03H (Send Control Data to Character Device)

Example

```

;*****;
;
;   Function 44H, Subfunctions 04H,05H:
;           IOCTL Block Device Control
;
;   int ioctl_block_ctrl(recvflag,drive_ltr,pbuffer,nbytes)
;       int  recvflag;
;       int  drive_ltr;
;       char *pbuffer;
;       int  nbytes;
;
;   Set recvflag = 0 to receive info, 1 to send.
;
;   Returns -1 for error, otherwise returns number of
;   bytes sent or received.
;
;*****;

cProc   ioctl_block_ctrl,PUBLIC,<ds>
parmB   recvflag
parmB   drive_ltr
parmDP  pbuffer
parmW   nbytes
cBegin
mov     al,recvflag    ; Get recvflag.
and     al,1           ; Keep only lsb.
add     al,4           ; AL = 04H for receive, 05H for send.
mov     bl,drive_ltr   ; Get drive letter.
or      bl,bl          ; Leave 0 alone.
jz      ibc
and     bl,not 20h     ; Convert letter to uppercase.
sub     bl,'A'-1       ; Convert to drive number: 'A' = 1,
;                       ; 'B' = 2, etc.

```

(more)

```
ibc:      mov     cx,nbytes      ; Get number of bytes to receive/send.
          loadDP ds,dx,pbuffer ; Get pointer to buffer.
          mov     ah,44h       ; Set function code.
          int     21h         ; Call MS-DOS.
          jnc     ibcx        ; Branch if no error.
          mov     ax,-1       ; Return -1 for all errors.
ibcx:
cEnd
```

Interrupt 21H (33)

2.0 and later

Function 44H (68) Subfunctions 06H and 07H

IOCTL: Check Input Status; Check Output Status

Function 44H Subfunctions 06H and 07H respectively determine whether a device or file associated with a handle is ready for input or output.

To Call

AH = 44H
 AL = 06H get input status
 07H get output status
 BX = handle number

Returns

If function is successful:

Carry flag is clear.

AL = input or output status:

00H not ready
 FFH ready

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function
 05H access denied
 06H invalid handle
 0DH invalid data (bad control string)

Programmer's Notes

- The status returned in AL has the following meanings:

Status	Device	Input File	Output File
00H	Not ready	Pointer at EOF	Ready
0FFH	Ready	Ready	Ready

- Output files always return a ready condition, even if the disk is full or no disk is in the drive.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunctions 06H,07H:
;           IOCTL Input/Output Status
;
;   int ioctl_char_status(outputflag,handle)
;       int outputflag;
;       int handle;
;
;   Set outputflag = 0 for input status, 1 for output status.
;
;   Returns -1 for all errors, 0 for not ready,
;   and 1 for ready.
;
;*****;

cProc   ioctl_char_status,PUBLIC
parmB   outputflag
parmW   handle
cBegin
    mov     al,outputflag    ; Get outputflag.
    and     al,1             ; Keep only lsb.
    add     al,6             ; AL = 06H for input status, 07H for output
                                ; status.
    mov     bx,handle        ; Get handle.
    mov     ah,44h           ; Set function code.
    int     21h              ; Call MS-DOS.
    jnc     isnoerr          ; Branch if no error.
    mov     ax,-1            ; Return error code.
    jmp     short isx

isnoerr:
    and     ax,1             ; Keep only lsb for return value.

isx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 08H

3.0 and later

IOCTL: Check If Block Device Is Removable

Function 44H Subfunction 08H checks whether the specified block device contains a removable storage medium, such as a floppy disk.

To Call

AH = 44H

AL = 08H

BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)

Returns

If function is successful:

Carry flag is clear.

AX = 00H storage medium removable

01H storage medium not removable

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function

0FH invalid drive

Programmer's Notes

- This subfunction exists to allow an application to check for a removable disk so that the user can be prompted to change disks if a required file is not found.
- When the carry flag is set, error code 01H normally means that MS-DOS did not recognize the function call. However, this error can also mean that the device driver does not support Subfunction 08H. In this case, MS-DOS assumes that the storage medium is not removable.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;      Function 44H, Subfunction 08H:
;              IOCTL Removable Block Device Query
;
;      int ioctl_block_fixed(drive_ltr)
;              int drive_ltr;
;
;      Returns -1 for all errors, 1 if disk is fixed (not
;      removable), 0 if disk is not fixed.
;
;*****;

cProc   ioctl_block_fixed,PUBLIC
parmB   drive_ltr
cBegin
        mov     bl,drive_ltr    ; Get drive letter.
        or      bl,bl          ; Leave 0 alone.
        jz      ibch
        and     bl,not 20h      ; Convert letter to uppercase.
        sub     bl,'A'-1        ; Convert to drive number: 'A' = 1,
                                ; 'B' = 2, etc.

ibch:
        mov     ax,4408h        ; Set function code, Subfunction 08H.
        int     21h            ; Call MS-DOS.
        jnc     ibchx          ; Branch if no error, AX = 0 or 1.
        cmp     ax,1           ; Treat error code of 1 as "disk is
                                ; fixed."

        je      ibchx
        mov     ax,-1          ; Return -1 for other errors.

ibchx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 09H

3.1 and later

IOCTL: Check If Block Device Is Remote

Function 44H Subfunction 09H checks whether the specified block device is local (attached to the computer running the program) or remote (redirected to a network server).

To Call

AH = 44H

AL = 09H

BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)

Returns

If function is successful:

Carry flag is clear.

DX = device attribute word:

bit 12 = 1 drive is remote

bit 12 = 0 drive is local

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function

0FH invalid drive

Programmer's Notes

- This subfunction should be avoided. Application programs should not distinguish between files on local and remote devices.
- When the carry flag is set, error code 01H can mean either that the function number is invalid or that the network has not been started.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;      Function 44H, Subfunction 09H:
;      IOCTL Remote Block Device Query
;
;      int ioctl_block_redir(drive_ltr)
;      int drive_ltr;
;
;      Returns -1 for all errors, 1 if disk is remote
;      (redirected), 0 if disk is local.
;
;*****;

cProc  ioctl_block_redir,PUBLIC
parmB  drive_ltr
cBegin
    mov     bl,drive_ltr    ; Get drive letter.
    or      bl,bl          ; Leave 0 alone.
    jz      ibr
    and     bl,not 20h     ; Convert letter to uppercase.
    sub     bl,'A'-1      ; Convert to drive number: 'A' = 1,
                        ; 'B' = 2, etc.

ibr:
    mov     ax,4409h      ; Set function code, Subfunction 09H.
    int     21h          ; Call MS-DOS.
    mov     ax,-1         ; Assume error.
    jc      ibrx         ; Branch if error, returning -1.
    inc     ax            ; Set AX = 0.
    test    dh,10h       ; Is bit 12 set?
    jz      ibrx         ; If not, disk is local: Return 0.
    inc     ax            ; Return 1 for remote disk.

ibrx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 0AH

3.1 and later

IOCTL: Check If Handle Is Remote

Function 44H Subfunction 0AH checks whether the handle in BX refers to a file or device that is local (on the computer running the program) or remote (redirected to a network server).

To Call

AH = 44H
AL = 0AH
BX = handle

Returns

If function is successful:

Carry flag is clear.

DX = attribute word for file or device:

bit 15 = 1 remote

bit 15 = 0 local

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function

06H invalid handle

Programmer's Notes

- Application programs should not distinguish between files on local and remote devices.
- When the carry flag is set, error code 01H can mean either that the function number is invalid or that the network has not been started.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunction 0AH:
;           IOCTL Remote Handle Query
;
;   int ioctl_char_redir(handle)
;       int handle;
;
;   Returns -1 for all errors, 1 if device/file is remote
;   (redirected), 0 if it is local.
;
;*****;

cProc   ioctl_char_redir,PUBLIC
parmW   handle
cBegin
    mov    bx,handle        ; Get handle.
    mov    ax,440ah        ; Set function code, Subfunction 0AH.
    int    21h             ; Call MS-DOS.
    mov    ax,-1           ; Assume error.
    jc     icrx            ; Branch on error, returning -1.
    inc    ax              ; Set AX = 0.
    test   dh,80h         ; Is bit 15 set?
    jz     icrx            ; If not, device/file is local:
                                ; Return 0.
    inc    ax              ; Return 1 for remote.

icrx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 0BH

3.1 and later

IOCTL: Change Sharing Retry Count

Function 44H Subfunction 0BH sets the number of times MS-DOS retries a disk operation after a failure caused by a file-sharing violation before it returns an error to the requesting process.

To Call

AH = 44H
AL = 0BH
CX = pause between retries
DX = number of retries

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
01H invalid function

Programmer's Notes

- The pause between retries is a machine-dependent value determined by the CPU and CPU clock speed. MS-DOS performs a delay loop that consists of 65,536 machine instructions for each iteration specified by the value in CX. The actual code is as follows:

```
xor    cx, cx
loop   $
```

The default number of retries is 3, with a pause of one loop between retries — equivalent to calling this subfunction with DX = 3 and CX = 1.

- When the carry flag is set, error code 01H indicates either that the function code is invalid or that file sharing (SHARE.EXE) is not loaded.
- Subfunction 0BH can be used to tune the system if file-contention problems are likely to arise with shared files but are expected to last only a short while.
- If file contention is expected and if some applications will lock regions of the file for an appreciable period of time, the user may need to be informed. The best procedure is to set an initial small number of retries with a short pause period. After notifying the user, the application can wait a reasonable amount of time for file access by adjusting the retry or pause-period values.

- If a process uses this subfunction, it should restore the original default values for the pause and number of retries before terminating, to avoid unwanted effects on the behavior of subsequent processes.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunction 0BH:
;           IOCTL Change Sharing Retry Count
;
;   int ioctl_set_retry(num_retries,wait_time)
;       int num_retries;
;       int wait_time;
;
;   Returns 0 for success, otherwise returns error code.
;
;*****;

cProc   ioctl_set_retry,PUBLIC,<ds,si>
parmW   num_retries
parmW   wait_time
cBegin
        mov     dx,num_retries ; Get parameters.
        mov     cx,wait_time
        mov     ax,440bh       ; Set function code, Subfunction 0BH.
        int     21h           ; Call MS-DOS.
        jc      isrx          ; Branch on error.
        xor     ax,ax

isrx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 0CH

3.2

IOCTL: Generic I/O Control for Handles

Function 44H Subfunction 0CH sets or gets the output iteration count for character-oriented devices. *See also* APPENDIX A: MS-DOS Version 3.3.

To Call

AH = 44H
 AL = 0CH
 BX = handle
 CH = category code:
 05H printer
 CL = function (minor) code:
 45H set iteration count
 65H get iteration count
 DS:DX = segment:offset of 2-byte buffer receiving or containing iteration-count word

Returns

If function is successful:

Carry flag is clear.

If CL was 65H on call:

DS:DX = segment:offset of iteration-count word

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 06H invalid handle

Programmer's Notes

- The iteration count controls the number of times the device driver tries to send output to the printer before assuming that the device is busy.
- With MS-DOS version 3.2, only category code 05H (printer) is supported by this subfunction.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunction 0CH:
;           Generic IOCTL for Handles
;
;   int ioctl_char_generic(handle,category,function,pbuffer)
;       int  handle;
;       int  category;
;       int  function;
;       int  *pbuffer;
;
;   Returns 0 for success, otherwise returns error code.
;
;*****;

cProc   ioctl_char_generic,PUBLIC,<ds>
parmW   handle
parmB   category
parmB   function
parmDP  pbuffer
cBegin
    mov     bx,handle       ; Get device handle.
    mov     ch,category     ; Get category
    mov     cl,function     ; and function.
    loadDP  ds,dx,pbuffer   ; Get pointer to data buffer.
    mov     ax,440ch        ; Set function code, Subfunction 0CH.
    int     21h             ; Call MS-DOS.
    jc     icgx             ; Branch on error.
    xor     ax,ax
icgx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 0DH

3.2

IOCTL: Generic I/O Control for Block Devices

Function 44H Subfunction 0DH includes six input/output tasks, or minor functions, related to block-oriented devices. The tasks perform the following operations: set or get device parameters; write, read, format and verify, or verify tracks on a logical drive.

This entry covers general information on Subfunction 0DH. Details on each minor code are presented in subsequent entries.

To Call

AH = 44H
 AL = 0DH
 BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
 CH = category code:
 08H disk drive
 CL = function (minor) code:
 40H set parameters for block device
 41H write track on logical drive
 42H format and verify track on logical drive
 60H get parameters for block device
 61H read track on logical drive
 62H verify track on logical drive
 DS:DX = segment:offset of parameter block

Returns

If function is successful:

Carry flag is clear.

If CL was 60H or 61H on call:

DS:DX = segment:offset of parameter block

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 02H invalid drive

Programmer's Notes

- Set Device Parameters (minor code 40H) must be used before an attempt to write, read, format, or verify a track on a logical drive. In general, the following sequence applies to any of these operations:

1. Get the current parameters (minor code 60H). Examine and save them.
 2. Set the new parameters (minor code 40H).
 3. Perform the task.
 4. Retrieve the original parameters and restore them (minor code 40H).
- With version 3.2 of MS-DOS, only category code 08H is supported by this subfunction.
 - Parameter blocks in the data buffer vary with the task being performed.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunction 0DH:
;           Generic IOCTL for Block Devices
;
;   int ioctl_block_generic(drv_ltr,category,func,pbuffer)
;       int  drv_ltr;
;       int  category;
;       int  func;
;       char *pbuffer;
;
;   Returns 0 for success, otherwise returns error code.
;
;*****;

cProc  ioctl_block_generic,PUBLIC,<ds>
parmB  drv_ltr
parmB  category
parmB  func
parmDP pbuffer
cBegin
mov    bl,drv_ltr    ; Get drive letter.
or     bl,bl        ; Leave 0 alone.
jz     ibg
and    bl,not 20h   ; Convert letter to uppercase.
sub    bl,'A'-1     ; Convert to drive number: 'A' = 1,
                    ; 'B' = 2, etc.

ibg:
mov    ch,category  ; Get category
mov    cl,func      ; and function.
loadDP ds,dx,pbuffer ; Get pointer to data buffer.
mov    ax,440dh    ; Set function code, Subfunction 0DH.
int    21h         ; Call MS-DOS.
jc     ibgx        ; Branch on error.
xor    ax,ax

ibgx:
cEnd

```

Interrupt 21H (33) Function 44H (68) Subfunction 0DH Minor Code 40H

IOCTL: Generic I/O Control for Block Devices: Set Device Parameters

Function 44H Subfunction 0DH minor code 40H sets device parameters in the parameter block pointed to by DS:DX.

To Call

AH = 44H
 AL = 0DH
 BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
 CH = category code:
 08H disk drive
 CL = 40H
 DS:DX = segment:offset of parameter block

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 02H invalid drive

Programmer's Notes

- The parameter block is formatted as follows:

Special-functions field: offset 00H, length 1 byte

Bit	Value	Meaning
0	0	Device BIOS parameter block (BPB) field contains a new default BPB.
	1	Use current BPB.
1	0	Use all fields in parameter block.
	1	Use track layout field only.

(more)

Special-functions field: offset 00H, length 1 byte *(continued)*

Bit	Value	Meaning
2	0	Sectors in track may be different sizes. (This setting should not be used.)
	1	Sectors in track are all same size; sector numbers range from 1 to the total number of sectors in the track. (This setting should always be used.)
3-7	0	Reserved.

Device type field: offset 01H, length 1 byte

Value	Meaning
00H	320/360 KB 5.25-inch disk
01H	1.2 MB 5.25-inch disk
02H	720 KB 3.5-inch disk
03H	Single-density 8-inch disk
04H	Double-density 8-inch disk
05H	Fixed disk
06H	Tape drive
07H	Other type of block device

Device attributes field: offset 02H, length 1 word

Bit	Value	Meaning
0	0	Removable storage medium
	1	Nonremovable storage medium
1	0	Door lock not supported
	1	Door lock supported
2-15	0	Reserved

Number of cylinders field: offset 04H, length 1 word

Meaning: Maximum number of cylinders supported; set by device driver

Media type field: offset 06H, length 1 byte

Value	Meaning
00H (default)	1.2 MB 5.25-inch disk
01H	320/360 KB 5.25-inch disk

Device BPB field: offset 07H, length 31 bytes.

Meaning: See Programmer's Note below.

If bit 0 = 0 in special-functions field, this field contains the new default BPB for the device.

If bit 0 = 1 in special-functions field, BPB in this field is returned by the device driver in response to subsequent Build BPB requests.

Track layout field: offset 26H, variable-length table

Length	Meaning
Word	Number of sectors in track
Word	Number of first sector in track*
Word	Size of first sector in track*
	.
	.
	.
Word	Number of last sector in track
Word	Size of last sector in track

*Sector number and sector size fields are repeated for each sector on the track. If bit 2 of the special-functions field is set, all sector sizes in the track layout field must be the same.

- The device BPB field is a 31-byte data structure. Information contained in the device BPB field describes the current disk and disk control areas. The device BPB field is formatted as follows:

Byte	Meaning
00-01H	Number of bytes per sector
02H	Number of sectors per allocation unit
03-04H	Number of sectors reserved, beginning at sector 0
05H	Number of file allocation tables (FATs)
06-07H	Maximum number of root-directory entries
08-09H	Total number of sectors
0AH	Media descriptor
0B-0CH	Number of sectors per FAT
0D-0EH	Number of sectors per track
0F-10H	Number of heads
11-14H	Number of hidden sectors
15-1FH	Reserved

- When Set Device Parameters (minor code 40H) is used, the number of cylinders should not be reset — some or all of the volume may become inaccessible.
- Subfunction 0DH minor code 60H performs the complementary action, Get Device Parameters.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

None

Interrupt 21H (33) Function 44H (68) Subfunction 0DH Minor Code 60H

IOCTL: Generic I/O Control for Block Devices: Get Device Parameters

Function 44H Subfunction 0DH minor code 60H gets device parameters in the parameter block pointed to by DS:DX.

To Call

AH = 44H
 AL = 0DH
 BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
 CH = category code:
 08H disk drive
 CL = 60H
 DS:DX = segment:offset of parameter block

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 02H invalid drive

Programmer's Notes

- The parameter block is formatted as follows:

Special-functions field: offset 00H, length 1 byte

Bit	Value	Meaning
0	0	Returns default BIOS parameter block (BPB) for the device.
	1	Returns BPB that the Build BPB device driver call would return.
1-7	0	Reserved (must be zero).

Device type field: offset 01H, length 1 byte

Value	Meaning
00H	320/360 KB 5.25-inch disk
01H	1.2 MB 5.25-inch disk
02H	720 KB 3.5-inch disk
03H	Single-density 8-inch disk
04H	Double-density 8-inch disk
05H	Fixed disk
06H	Tape drive
07H	Other type of block device

Device attributes field: offset 02H, length 1 word

Bit	Value	Meaning
0	0	Removable storage medium
	1	Nonremovable storage medium
1	0	Door lock not supported
	1	Door lock supported
2-15	0	Reserved

Number of cylinders field: offset 04H, length 1 word

Meaning: Maximum number of cylinders supported; set by device driver

Media type field: offset 06H, length 1 byte

Value	Meaning
00H (default)	1.2 MB 5.25-inch disk
01H	320/360 KB 5.25-inch disk

Device BPB field: offset 07H, length 31 bytes

Meaning: See Programmer's Note below.

If bit 0 = 0 in special-functions field, this field contains the new default BPB for the device.

If bit 0 = 1 in special-functions field, BPB in this field is returned by the device driver in response to subsequent Build BPB requests.

Track layout field: offset 26H

Unused

- The device BPB field is a 31-byte data structure. Information contained in the device BPB field describes the current disk and disk control areas. The device BPB field is formatted as follows:

Byte	Meaning
00-01H	Number of bytes per sector
02H	Number of sectors per allocation unit
03-04H	Number of sectors reserved, beginning at sector 0
05H	Number of file allocation tables (FATs)
06-07H	Maximum number of root-directory entries
08-09H	Total number of sectors
0AH	Media descriptor
0B-0CH	Number of sectors per FAT
0D-0EH	Number of sectors per track
0F-10H	Number of heads
11-14H	Number of hidden sectors
15-1FH	Reserved

- Subfunction 0DH minor code 40H performs the complementary action, Set Device Parameters.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

None

Interrupt 21H (33) Function 44H (68) Subfunction 0DH Minor Codes 41H and 61H

IOCTL: Generic I/O Control for Block Devices: Write Track on Logical Drive;
Read Track on Logical Drive

Function 44H Subfunction 0DH minor code 41H writes a track on the logical drive specified in BL and minor code 61H reads a track on the logical drive specified in BL, using information in the parameter block pointed to by DS:DX.

To Call

AH = 44H
AL = 0DH
BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
CH = category code:
 08H disk drive
CL = function (minor) code:
 41H write a track
 61H read a track
DS:DX = segment:offset of parameter block

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 02H invalid drive

Programmer's Notes

- The parameter block is formatted as follows:

Offset	Size	Meaning
00H	Byte	Special-functions field; must be 0.
01H	Word	Head field; contains number of disk head used for read/write.
03H	Word	Cylinder field; contains number of disk cylinder used for read/write.
05H	Word	First-sector field; contains number of first sector to read or write (first sector on track = sector 0).
07H	Word	Number-of-sectors field; contains number of sectors to transfer.
09H	Dword	Transfer address field; contains address of buffer to use for data transfer.

- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

None

Interrupt 21H (33) Function 44H (68) Subfunction 0DH Minor Codes 42H and 62H

IOCTL: Generic I/O Control for Block Devices: Format and Verify Track on Logical Drive; Verify Track on Logical Drive

Function 44H Subfunction 0DH minor code 42H formats and verifies a track on the specified logical drive and minor code 62H verifies a track on the specified logical drive, using information in the parameter block pointed to by DS:DX.

To Call

AH = 44H
AL = 0DH
BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
CH = category code:
 08H disk drive
CL = function (minor) code:
 42H format and verify
 62H verify
DS:DX = segment:offset of parameter block

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 02H invalid drive

Programmer's Notes

- The parameter block is formatted as follows:

Offset	Size	Meaning
00H	Byte	Special-functions field; must be 0.
01H	Word	Head field; contains number of disk head used for format/verify.
03H	Word	Cylinder field; contains number of cylinder used for format/verify.

- This driver subfunction allows the writing of generic formatting programs that are minimally hardware dependent.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

None

Interrupt 21H (33) 3.2 Function 44H (68) Subfunctions 0EH and 0FH

IOCTL: Get Logical Drive Map; Set Logical Drive Map

Function 44H Subfunction 0EH allows a process to determine whether more than one logical drive is assigned to a block device. Subfunction 0FH sets the next logical drive number that will be used to reference a block device.

To Call

AH = 44H
 AL = 0EH get logical drive map
 0FH set logical drive map
 BL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)

Returns

If function is successful:

Carry flag is clear.

AL = mapping code:
 00H only one letter assigned to the block device
 01-1AH logical drive letter (A through Z) mapped to block device

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function
 0FH invalid drive

Programmer's Notes

- If a drive has not been assigned a logical mapping with Function 44H Subfunction 0FH, the logical and physical drive references are the same. (The default is that logical drive A and physical drive A both refer to physical drive A.)
- If this function is used to map logical drives to physical drives, the result is similar to MS-DOS's treatment of a single physical drive as both A and B on a system with one floppy-disk drive. With MS-DOS version 3.2, however, the installable device driver DRIVER.SYS extends this type of physical/logical referencing to other drives. Therefore, processes can prompt for disks themselves, instead of using the prompt provided by MS-DOS.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;   Function 44H, Subfunctions 0EH, 0FH:
;           IOCTL Get/Set Logical Drive Map
;
;   int ioctl_drive_owner(setflag, drv_ltr)
;       int setflag;
;       int drv_ltr;
;
;   Set setflag = 1 to change drive's map, 0 to get
;   current map.
;
;   Returns -1 for all errors, otherwise returns
;   the block device's current logical drive letter.
;
;*****;

cProc   ioctl_drive_owner,PUBLIC
parmB   setflag
parmB   drv_ltr
cBegin
    mov     al,setflag      ; Load setflag.
    and     al,1            ; Keep only lsb.
    add     al,0eh         ; AL = 0EH for get, 0FH for set.
    mov     bl,drv_ltr     ; Get drive letter.
    or      bl,bl          ; Leave 0 alone.
    jz      ido
    and     bl,not 20h     ; Convert letter to uppercase.
    sub     bl,'A'-1       ; Convert to drive number: 'A' = 1,
                          ; 'B' = 2, etc.

ido:
    mov     bh,0
    mov     ah,44h         ; Set function code.
    int     21h           ; Call MS-DOS.
    mov     ah,0           ; Clear high byte.
    jnc     idox           ; Branch if no error.
    mov     ax,-1-'A'      ; Return -1 for errors.

idox:
    add     ax,'A'         ; Return drive letter.

cEnd

```

Interrupt 21H (33) Function 45H (69)

2.0 and later

Duplicate File Handle

Function 45H obtains an additional handle for a currently open file or device.

To Call

AH = 45H
BX = handle for open file or device

Returns

If function is successful:

Carry flag is clear.

AX = new handle number

If function is not successful:

Carry flag is set.

AX = error code:

04H	too many open files
06H	invalid handle

Programmer's Notes

- The file pointer for the new handle is set to the same position as the pointer for the original handle. Any subsequent changes to the file are reflected in both handles. Thus, using either handle for a read or write operation moves the file pointer associated with both.
- Function 45H is often used to duplicate the handle assigned to standard input (0) or standard output (1) before a call to Function 46H (Force Duplicate File Handle). The handle forced by Function 46H can then be used for redirected input or output from or to a file or device.
- Another use for Function 45H is to keep a file open while its directory entry is being updated to reflect a change in length. If a new handle is obtained with Function 45H and then closed with Function 3EH (Close File), the directory and FAT entries for the file are updated. At the same time, because the original handle remains open, the file need not be reopened for additional read or write operations.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

46H (Force Duplicate File Handle)

Example

```
;*****;  
;  
;           Function 45H: Duplicate File Handle           ;  
;  
;           int dup_handle(handle)                       ;  
;               int handle;                             ;  
;  
;           Returns -1 for errors,                       ;  
;           otherwise returns new handle.                ;  
;  
;*****;  
  
cProc  dup_handle,PUBLIC  
parmW  handle  
cBegin  
      mov    bx,handle      ; Get handle to copy.  
      mov    ah,45h        ; Set function code.  
      int    21h           ; Ask MS-DOS to duplicate handle.  
      jnb   dup_ok         ; Branch if copy was successful.  
      mov    ax,-1         ; Else return -1.  
  
dup_ok:  
cEnd
```

Interrupt 21H (33) Function 46H (70)

2.0 and later

Force Duplicate File Handle

Function 46H forces the open handle specified in CX to track the same file or device specified by the handle in BX.

To Call

AH = 46H
BX = open handle to be duplicated
CX = open handle to be forced

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:

04H	too many open files
06H	invalid handle

Programmer's Notes

- The handle in BX must refer either to an open file or to any of the five standard handles reserved by MS-DOS: standard input, standard output, standard error, standard auxiliary, or standard printer.
- If the handle in CX refers to an open file, the file is closed.
- The file pointer for the duplicate handle is set to the same position as the pointer for the original handle. Changing the position of either file pointer moves the pointer associated with the other handle as well.
- When used with Function 45H (Duplicate File Handle), Function 46H can be used to redirect input and output as follows:
 1. Duplicate the handle from which input or output will be redirected with Function 45H (Duplicate File Handle). Save the duplicated handle for later reference (Step 3).
 2. Call Function 46H, with the handle to be redirected from in the CX register and the handle to be redirected to in the BX register.
 3. To restore I/O redirection to its original state, call Function 46H again, with the redirected file handle from Step 2 in the CX register and the duplicated file handle from Step 1 in the BX register.

This procedure is normally used to redirect a standard device, but it can redirect any device referenced by handles.

- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

45H (Duplicate File Handle)

Example

```

;*****;
;
;      Function 46H: Force Duplicate File Handle
;
;      int dup_handle2(existhandle,newhandle)
;          int existhandle,newhandle;
;
;      Returns -1 for errors,
;      otherwise returns newhandle unchanged.
;
;*****;

cProc  dup_handle2,PUBLIC
parmW  existhandle
parmW  newhandle
cBegin
mov    bx,existhandle ; Get handle of existing file.
mov    cx,newhandle   ; Get handle to copy into.
mov    ah,46h         ; Close handle CX and then
int    21h            ; duplicate BX's handle into CX.
mov    ax,newhandle   ; Prepare return value.
jnb    dup2_ok        ; Branch if close/copy was successful.
mov    ax,-1          ; Else return -1.

dup2_ok:
cEnd

```

Interrupt 21H (33) Function 47H (71)

2.0 and later

Get Current Directory

Function 47H returns the path, excluding the drive and leading backslash, of the current directory for the specified drive.

To Call

AH = 47H
DL = drive number (0 = default drive, 1 = drive A, 2 = drive B, and so on)
DS:SI = segment:offset of 64-byte buffer

Returns

If function is successful:

Carry flag is clear.

Buffer is filled in with ASCIIZ pathname.

If function is not successful:

Carry flag is set.

AX = error code:
0FH invalid drive

Programmer's Notes

- The string representing the pathname is returned as a null-terminated ASCII string (ASCIIZ).
- This function does not return an error if the buffer is too small or is incorrectly identified. MS-DOS pathnames can be as long as 64 characters; if the buffer is less than 64 bytes, MS-DOS can overwrite sections of memory outside the buffer.
- The path returned by Function 47H starts at the root directory and fully specifies the path to the current directory but does not include a drive code or a leading backslash (\) character.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

3BH (Change Current Directory)

Example

```

;*****;
;
;      Function 47H: Get Current Directory      ;
;
;      int get_dir(drive_ltr,pbuffer)         ;
;      int drive_ltr;                         ;
;      char *pbuffer;                         ;
;
;      Returns -1 for bad drive,              ;
;      otherwise returns pointer to pbuffer.  ;
;
;*****;

cProc  get_dir,PUBLIC,<ds,si>
parmB  drive_ltr
parmDP pbuffer
cBegin
    loadDP ds,si,pbuffer ; Get pointer to buffer.
    mov    dl,drive_ltr  ; Get drive number.
    or     dl,dl         ; Leave 0 alone.
    jz     gdir
    and    dl,not 20h    ; Convert letter to uppercase
    sub    dl,'A'-1     ; Convert to drive number: 'A' = 1,
                        ; 'B' = 2, etc.

gdir:
    mov    ah,47h       ; Set function code.
    int    21h          ; Call MS-DOS.
    mov    ax,si        ; Return pointer to buffer ...
    jnb   gd_ok
    mov    ax,-1        ; ... unless an error occurred.

gd_ok:
cEnd

```


Interrupt 21H (33) Function 48H (72)

2.0 and later

Allocate Memory Block

Function 48H allocates a block of memory, in paragraphs (1 paragraph = 16 bytes), to the requesting process.

To Call

AH = 48H
BX = number of paragraphs to allocate

Returns

If function is successful:

Carry flag is clear.

AX = segment address of base of allocated block

If function is not successful:

Carry flag is set.

AX = error code:

07H memory control blocks damaged

08H insufficient memory to allocate as requested

BX = size of largest available block (paragraphs)

Programmer's Notes

- If the allocation succeeds, the address returned in AX is the segment of the base of the block. This address would be copied to a segment register (usually DS or ES) to access the memory within the block.
- If the amount of memory requested is greater than the amount in any available contiguous block of memory, the number of paragraphs in the largest available memory block is returned in the BX register.
- The default memory-management strategy in MS-DOS is to choose the first contiguous block of memory that fits the request, no matter how good the fit. With MS-DOS versions 3.0 and later, however, the memory-management strategy can be altered with Function 58H (Get/Set Allocation Strategy).
- If a process actively allocates and frees blocks of memory, the transient program area (TPA) can become fragmented—that is, small blocks of memory can be orphaned because the memory-management strategy seeks contiguous blocks of memory.
- If a process writes to memory outside the limits of the allocated block, it can destroy control structures for other memory blocks. This could result in failure of subsequent memory-management functions, and it will cause MS-DOS to print an error message and halt when the process terminates.

- Initially, the MS-DOS loader allocates all available memory to .COM programs. Function 4AH (Resize Memory Block) can free memory for dynamic reallocation by a process or by its children.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

49H (Free Memory Block)
 4AH (Resize Memory Block)
 58H (Get/Set Allocation Strategy)

Example

```

;*****;
;                                     ;
;      Function 48H: Allocate Memory Block      ;
;                                     ;
;      int get_block(nparas,pblocksegp,pmaxparas) ;
;      int nparas,*pblockseg,*pmaxparas;      ;
;                                     ;
;      Returns 0 if nparas are allocated OK and ;
;      pblockseg has segment address of block, ;
;      otherwise returns error code with pmaxparas ;
;      set to maximum block size available.    ;
;                                     ;
;*****;

cProc  get_block,PUBLIC,ds
parmW  nparas
parmDP pblockseg
parmDP pmaxparas
cBegin
mov    bx,nparas      ; Get size request.
mov    ah,48h         ; Set function code.
int    21h            ; Ask MS-DOS for memory.
mov    cx,bx          ; Save BX.
loadDP ds,bx,pmaxparas
mov    [bx],cx        ; Return result, assuming failure.
jb    gb_err          ; Exit if error, leaving error code
; in AX.

loadDP ds,bx,pblockseg
mov    [bx],ax        ; No error, so store address of block.
xor    ax,ax          ; Return 0.

gb_err:
cEnd

```

Interrupt 21H (33) Function 49H (73)

2.0 and later

Free Memory Block

Function 49H releases a block of memory previously allocated with Function 48H (Allocate Memory Block).

To Call

AH = 49H
ES = segment address of memory block to release

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:

07H memory control blocks damaged
09H incorrect memory segment specified

Programmer's Notes

- The memory segment pointed to by ES:0000H must have been allocated by Function 48H (Allocate Memory Block).
- If a program has inadvertently damaged any of the system's memory control blocks by writing outside an allocated block, an attempt to free allocated memory results in error code 07H (memory control blocks damaged).
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

48H (Allocate Memory Block)
4AH (Resize Memory Block)
58H (Get/Set Allocation Strategy)

Example

```
;*****;  
;  
;           Function 49H: Free Memory Block           ;  
;  
;           int free_block(blockseg)                   ;  
;           int blockseg;                               ;  
;  
;           Returns 0 if block freed OK,               ;  
;           otherwise returns error code.              ;  
;  
;*****;  
  
cProc   free_block,PUBLIC  
parmW   blockseg  
cBegin  
    mov     es,blockseg      ; Get block address.  
    mov     ah,49h          ; Set function code.  
    int     21h             ; Ask MS-DOS to free memory.  
    jb     fb_err          ; Branch on error.  
    xor     ax,ax           ; Return 0 if successful.  
  
fb_err:  
cEnd
```

Interrupt 21H (33) Function 4AH (74)

2.0 and later

Resize Memory Block

Function 4AH adjusts the size of a previously allocated block of memory.

To Call

AH = 4AH
BX = new size of memory block, in paragraphs
ES = segment address of previously allocated memory block

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:

07H memory control blocks damaged
08H insufficient memory to allocate as requested
09H incorrect memory segment specified

BX = maximum number of paragraphs available (if an increase was requested)

Programmer's Notes

- Function 4AH can be used to change the size of a memory block previously allocated with Function 48H (Allocate Memory Block) or to modify the amount of memory originally allocated to a process by MS-DOS.
- If a process is denied an increase in the amount of memory it has been allocated, MS-DOS places the size of the largest contiguous block available in the BX register. The process can then notify the user of the problem and exit, or it can continue to operate in a reduced memory environment.
- Because the MS-DOS loader allocates all available memory to .COM programs, such a program should use Function 4AH immediately (with the segment address of its program segment prefix, or PSP) to release any memory that is not needed. This is mandatory if the .COM program will either allocate memory dynamically or use Function 4BH (Load and Execute Program) to load a child process or overlay.

In addition, if Function 4AH is used to adjust the amount of memory allocated to a .COM program, the stack pointer must be adjusted so that it is within the limits of the program's revised memory allocation.

- If this function is used to shrink an allocated block, any memory above the new limit is not owned by the process and should never be used. If this function is used to expand an allocated block, the contents of memory above the old boundary are unpredictable and the memory should be initialized before use.
- Although it is not possible to predict how much memory-resident software and how many installable device drivers will be used on a computer system, Function 4AH can reliably determine the amount of memory available to an application.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

48H (Allocate Memory Block)
 49H (Free Memory Block)
 58H (Get/Set Allocation Strategy)

Example

```

;*****;
;
;           Function 4AH: Resize Memory Block
;
;           int modify_block (nparas,blockseg,pmxparas)
;           int nparas,blockseg,*pmxparas;
;
;           Returns 0 if modification was a success,
;           otherwise returns error code with pmxparas
;           set to max number of paragraphs available.
;
;*****;

cProc   modify_block,PUBLIC,ds
parmW   nparas
parmW   blockseg
parmDP  pmxparas
cBegin
        mov     es,blockseg      ; Get block address.
        mov     bx,nparas        ; Get nparas.
        mov     ah,4ah           ; Set function code.
        int     21h              ; Ask MS-DOS to change block size.
        mov     cx,bx            ; Save BX.
        loadDP ds,bx,pmxparas
        mov     [bx],cx          ; Set pmxparas, assuming failure.
        jb     mb_exit           ; Branch if size change error.
        xor     ax,ax            ; Return 0 if successful.
mb_exit:
cEnd

```

Interrupt 21H (33) Function 4BH (75)

2.0 and later

Load and Execute Program (EXEC)

Function 4BH, often called EXEC, loads a program file into memory and, optionally, executes the program. This function can also be used to load a program overlay.

To Call

AH	=	4BH	
AL	=	00H	load and execute program
		03H	load overlay
DS:DX	=	segment:offset of ASCIIZ pathname for an executable program file	
ES:BX	=	segment:offset of parameter block	

Returns

If function is successful:

Carry flag is clear.

With MS-DOS versions 2.x, all registers except CS and IP can be destroyed; with MS-DOS versions 3.x, registers are preserved.

If function is not successful:

Carry flag is set.

AX	=	error code:	
		01H	invalid function (AL did not contain 00H or 03H)
		02H	file not found
		03H	path not found
		05H	access denied
		08H	insufficient memory
		0AH	bad environment
		0BH	bad format (AL = 00H only)

Programmer's Notes

- The pathname must be a null-terminated ASCII string (ASCIIZ).
- The handles for any files opened by the parent process before the call to Function 4BH are inherited by the child process, unless the parent specified otherwise in calling Function 3DH (Open File with Handle).

All standard devices also remain open and available to the child process. Thus, the parent process can control the files used by the child process and control redirection for the child process.

- If AL = 00H, the parameter block is 14 bytes long and formatted in four parts, as follows:

Offset	Length	Meaning
00H	Word	Segment address of environment to be passed; 00H indicates child program inherits environment of the current process.
02H	Dword	Segment:offset address of command tail for the new program segment prefix (PSP). Command tail must be 128 bytes or fewer and formatted as a count byte followed by an ASCII string and terminated by a carriage return, as follows: <pre>db 7, 'a:mydoc', 0Dh</pre> The carriage return is not included in the count; the command tail is placed at offset 80H in the new process's PSP.
06H	Dword	Segment:offset address of an FCB to be copied to the default FCB position at offset 5CH in the new process's PSP.
0AH	Dword	Segment:offset address of an FCB to be copied to the default FCB position at offset 6CH in the new process's PSP.

If AL = 03H, the parameter block is 4 bytes long and formatted in two parts, as follows:

Offset	Length	Meaning
00H	Word	Segment address where the overlay is to be loaded.
02H	Word	Relocation factor to be applied to the code image (.EXE files only); not needed if the file is a .COM program or is data.

- The first 2 bytes of the parameter block for Function 4BH Subfunction 00H contain either the segment address for an environment block to be passed to the new process or zero. If the value is zero, the child process inherits an exact copy of the parent process's environment.

The environment block must be aligned on a paragraph boundary (a multiple of 16 bytes). It can be as large as 32 KB, and it consists of a block of ASCIIZ strings, each in the following form:

parameter=value

For example:

```
db    'VERIFY=ON',0
```

The final string in the environment block is followed by a second zero byte. With MS-DOS versions 3.0 and later, the second zero is followed by a word containing a count and an ASCIIZ string containing the drive and pathname of the program file.

The environment passed to the child process allows the parent process to send it messages regarding the system state or control parameters. The pathname included with MS-DOS versions 3.0 and later enables the child process to determine where it was loaded from.

- If AL = 00H, MS-DOS creates a PSP for the new process and sets the terminate and Control-C addresses to the instruction in the parent process that follows the call to Function 4BH. If AL = 03H, no PSP is created.
- Before AL = 00H is used to load and execute a process, the system must contain enough free memory to accommodate the new process. Function 4AH (Resize Memory Block) should be used, if necessary, to reduce the amount of memory allocated to the parent process. If the parent is a .COM program, allocated memory *must* be reduced, because a .COM program is given ownership of all available memory when it is executed.

If Function 4BH is called with AL = 03H, free memory is not a factor, because MS-DOS assumes the new process is being loaded into the calling process's own address space.

- If Function 4BH is called with AL = 00H, the child process remains in control until it executes an exit request, such as Function 4CH (Terminate Process with Return Code), or until Control-C or Control-Break is received or a critical error occurs and the user responds *Abort* to the *Abort, Retry, Ignore?* message.
- With MS-DOS versions 2.x, SS and SP must be saved in the current code segment before Function 4BH is invoked with AL = 00H. When the parent process regains control, all registers other than CS:IP and the stack will most likely have been changed by loading and executing the child process.
- Function 4BH with AL = 03H is useful for loading program overlays or for loading data to be used by the parent process (if that data requires relocation).
- If the child process that is executed attempts to remain resident through either Interrupt 27H or Interrupt 21H Function 31H (Terminate and Stay Resident), system memory becomes permanently fragmented and subsequent processes can fail because of lack of memory.
- The EXEC function (with AL = 00H) is commonly used to load a new copy of COMMAND.COM and then execute an MS-DOS command from within another program.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

31H (Terminate and Stay Resident)
 4CH (Terminate Process with Return Code)
 4DH (Get Return Code of Child Process)

Examples

```

;*****;
;
;      Function 4BH: Load and Execute Program
;
;      int execute(pprogname,pcmdtail)
;      char *pprogname,*pcmdtail;
;
;      Returns 0 if program loaded, ran, and
;      terminated successfully, otherwise returns
;      error code.
;
;*****;

sBegin data
$cmdlen =      126
$cmd      db      $cmdlen+2 dup (?) ; Make space for command line, plus
;      ; 2 extra bytes for length and
;      ; carriage return.

$fcdb      db      0      ; Make dummy FCB.
          db      'dummy fcb'
          db      0,0,0,0

          ; Here's the EXEC parameter block:
$epb      dw      0      ; 0 means inherit environment.
          dw      dataOFFSET $cmd ; Pointer to cmd line.
          dw      seg dgroup
          dw      dataOFFSET $fcdb ; Pointer to FCB #1.
          dw      seg dgroup
          dw      dataOFFSET $fcdb ; Pointer to FCB #2.
          dw      seg dgroup

sEnd      data
sBegin    code

$sp      dw      ?      ; Allocate space in code seg
$ss      dw      ?      ; for saving SS and SP.

Assumes  ES,dgroup

cProc    execute,PUBLIC,<ds,si,di>
parmDP  pprogname
parmDP  pcmdtail
cBegin
mov      cx,$cmdlen      ; Allow command line this long.
loadDP  ds,si,pcmdtail  ; DS:SI = pointer to cmdtail string.

```

(more)

```

        mov     ax,seg dgroup:$cmd      ; Set ES = data segment.
        mov     es,ax
        mov     di,dataOFFSET $cmd+1   ; ES:DI = pointer to 2nd byte of
                                        ; our command-line buffer.
copycmd:
        lodsb                    ; Get next character.
        or      al,al              ; Found end of command tail?
        jz      endcopy            ; Exit loop if so.
        stosb                      ; Copy to command buffer.
        loop   copycmd
endcopy:
        mov     al,13
        stosb                      ; Store carriage return at
                                        ; end of command.

        neg     cl
        add     cl,$cmdlen          ; CL = length of command tail.
        mov     es:$cmd,cl         ; Store length in command-tail buffer.

        loadDP ds,dx,pprograme ; DS:DX = pointer to program name.
        mov     bx,dataOFFSET $epb ; ES:BX = pointer to parameter
                                        ; block.

        mov     cs:$ss,ss          ; Save current stack SS:SP (because
        mov     cs:$sp,sp          ; EXEC function destroys stack).
        mov     ax,4b00h           ; Set function code.
        int     21h                ; Ask MS-DOS to load and execute
                                        ; program.
        cli                      ; Disable interrupts.
        mov     ss,cs:$ss          ; Restore stack.
        mov     sp,cs:$sp
        sti                      ; Enable interrupts.
        jb     ex_err              ; Branch on error.
        xor     ax,ax              ; Return 0 if no error.
ex_err:
cEnd
sEnd    code

```

```

;*****;
;
; Function 4BH: Load an Overlay Program ;
;
; int load_overlay(pfilename,loadseg) ;
; char *pfilename; ;
; int loadseg; ;
;
; Returns 0 if program has been loaded OK, ;
; otherwise returns error code. ;
;
; To call an overlay function after it has been ;
; loaded by load_overlay(), you can use ;
; a far indirect call: ;

```

(more)

```

;
; 1. FTYPE (far *ovlptr)();
; 2. *((unsigned *)&ovlptr + 1) = loadseg;
; 3. *((unsigned *)&ovlptr) = offset;
; 4. (*ovlptr)(arg1,arg2,arg3,...);
;
;
; Line 1 declares a far pointer to a
; function with return type FTYPE.
;
;
; Line 2 stores loadseg into the segment
; portion (high word) of the far pointer.
;
;
; Line 3 stores offset into the offset
; portion (low word) of the far pointer.
;
;
; Line 4 does a far call to offset
; bytes into the segment loadseg
; passing the arguments listed.
;
;
; To return correctly, the overlay must end with a far
; return instruction. If the overlay is
; written in Microsoft C, this can be done by
; declaring the overlay function with the
; keyword "far".
;
;
;*****
sBegin data
; The overlay parameter block:
$lob dw ? ; space for load segment;
dw ? ; space for fixup segment.
sEnd data

sBegin code

cProc load_overlay,PUBLIC,<ds,si,di>
parmDP pfilename
parmW loadseg
cBegin
loadDP ds,dx,pfilename ; DS:DX = pointer to program name.
mov ax,seg dgroup:$lob ; Set ES = data segment.
mov es,ax
mov bx,dataOFFSET $lob ; ES:BX = pointer to parameter
; block.
mov ax,loadseg ; Get load segment parameter.
mov es:[bx],ax ; Set both the load and fixup
mov es:[bx+2],ax ; segments to that segment.

mov cs:$ss,ss ; Save current stack SS:SP (because
mov cs:$sp,sp ; EXEC function destroys stack).
mov ax,4b03h ; Set function code.
int 21h ; Ask MS-DOS to load the overlay.
cli ; Disable interrupts.

```

(more)

```
        mov     ss,cs:$ss      ; Restore stack.
        mov     sp,cs:$sp
        sti                      ; Enable interrupts.
        jb     lo_err          ; Branch on error.
        xor     ax,ax          ; Return 0 if no error.
lo_err:
cEnd
sEnd    code
```

Interrupt 21H (33) Function 4CH (76)

2.0 and later

Terminate Process with Return Code

Function 4CH terminates the current process with a return code and returns control to the calling (parent) process.

To Call

AH = 4CH
AL = return code

Returns

Nothing

Programmer's Notes

- When a process is terminated with Function 4CH, MS-DOS restores the termination-handler (Interrupt 22H), Control-C handler (Interrupt 23H), and critical error handler (Interrupt 24H) addresses from the program segment prefix, or PSP (offsets 0AH, 0EH, and 12H). MS-DOS also flushes the file buffers to disk, updates the disk directory, closes all files with open handles belonging to the terminated process, and then transfers control to the termination-handler address.
- On termination with Function 4CH, all memory owned by the process is freed.
- Function 4CH is the recommended method for terminating all processes — particularly sizable .EXE files — that do not stay resident. This function should be used in preference to the other termination methods (Interrupt 20H, Interrupt 21H Function 00H, near RET for .COM files, or a jump to PSP:0000H). Memory-resident programs should be terminated with Function 31H (Terminate and Stay Resident).
- A return code of 00H is customarily used to indicate that the process executed successfully; a nonzero return code is used to indicate that the process terminated because of an error or lack of resources — for example, the file could not be opened, the process could not be allocated sufficient memory, and so on.
- If the terminated process was invoked by a command line or batch file, control returns to COMMAND.COM and the transient portion of the command interpreter is reloaded, if necessary. If a batch file was in progress, execution continues with the next line of the file and the return code can be tested with an IF ERRORLEVEL statement. Otherwise, the command prompt is issued.

If the terminated process was loaded by a process other than COMMAND.COM, the parent process can retrieve the child's return code with Function 4DH (Get Return Code of Child Process).
- In a networking environment running under MS-DOS version 3.1 or later, all file locks should be removed by the process before it calls Function 4CH to terminate.

Related Functions

- 00H (Terminate Process)
- 31H (Terminate and Stay Resident)
- 4DH (Get Return Code of Child Process)

Example

```

;*****;
;
;      Function 4CH: Terminate Process with Return Code      ;
;
;      int terminate(returncode)                             ;
;      int returncode;                                       ;
;
;      Does NOT return at all!                               ;
;
;*****;

cProc  terminate,PUBLIC
parmB  returncode
cBegin
      mov  al,returncode  ; Set return code.
      mov  ah,4ch        ; Set function code.
      int  21h           ; Call MS-DOS to terminate process.
cEnd

```

Interrupt 21H (33) Function 4DH (77)

2.0 and later

Get Return Code of Child Process

Function 4DH retrieves the return code of a child process that was invoked with Function 4BH (Load and Execute Program) and terminated with either Function 31H (Terminate and Stay Resident) or Function 4CH (Terminate Process with Return Code).

To Call

AH = 4DH

Returns

AH = termination method:

- | | |
|-----|--|
| 00H | normal termination (Interrupt 20H, or Interrupt 21H Function 00H or Function 4CH) |
| 01H | terminated by entry of Control-C |
| 02H | terminated by critical error handler (for example, user responded <i>Abort</i> to <i>Abort, Retry, Ignore?</i> prompt) |
| 03H | terminated and stayed resident (Interrupt 27H or Interrupt 21H Function 31H) |

AL = return code passed by child process

If terminated with Interrupt 20H, Interrupt 21H Function 00H, or Interrupt 27H:

AL = 00H

Programmer's Notes

- Function 4DH can be used only once to retrieve the return code of a terminated process. Subsequent calls do not yield meaningful results.
- Function 4DH does not set the carry flag to indicate an error. If no previous child process exists, the information returned in AH and AL is undefined.

Related Functions

31H (Terminate and Stay Resident)

4CH (Terminate Process with Return Code)

Example

```

;*****;
;
;      Function 4DH: Get Return Code of Child Process
;
;      int child_ret_code()
;
;      Returns the return code of the last
;      child process.
;*****;

cProc  child_ret_code,PUBLIC
cBegin
mov    ah,4dh      ; Set function code.
int    21h        ; Ask MS-DOS to return code.
cbw                   ; Convert AL to a word.
cEnd
```

Interrupt 21H (33) Function 4EH (78)

2.0 and later

Find First File

Function 4EH searches the specified directory for the first matching entry.

To Call

AH = 4EH
 CX = attribute word
 DS:DX = segment:offset of ASCIIZ pathname

Returns

If function is successful:

Carry flag is clear.

Current disk transfer area (DTA) contains the following information about the file:

Offset	Length (bytes)	Value
00H	21	Reserved for use by MS-DOS in subsequent call to Function 4FH (Find Next File)
15H	1	File attribute
16H	2	Time of last write
18H	2	Date of last write
1AH	2	Low word of file size
1CH	2	High word of file size
1EH	13	Filename and extension in ASCIIZ form with blanks removed and period inserted between filename and extension

If function is not successful:

Carry flag is set.

AX = error code:
 02H file not found
 03H path not found
 12H no more files; no match found

Programmer's Notes

- The pathname must be a null-terminated ASCII string (ASCIIZ).

- The filename and extension portions of the pathname can contain the MS-DOS wild-cards ? (match any character) and * (match all remaining characters).
- The DTA should be set with Function 1AH (Set DTA Address) before Function 4EH is called. If no DTA address is set, MS-DOS uses a default 128-byte buffer at offset 80H in the program segment prefix (PSP).
- The attribute word in CX controls the search as follows:
 - If the attribute word is 00H, only normal files are included in the search.
 - If the attribute word has any combination of bits 1, 2, and 4 (hidden, system, and subdirectory bits) set, the search includes normal files as well as files with any of the attributes specified.
 - If the attribute word has bit 3 set (volume-label bit), only a matching volume label is returned.
 - Bits 0 and 5 (read-only and archive bits) are ignored by Function 4EH.
- If Function 4FH (Find Next File) is used in conjunction with Function 4EH, the DTA must be preserved, because the first 21 bytes contain information needed by Function 4FH.
- The time at which the file was last written is returned as a binary value in a word formatted as follows:

Bits	Meaning
0–4	Number of seconds divided by 2
5–10	Minutes (0 through 59)
11–15	Hours, based on a 24-hour clock (0 through 23).

- The date on which the file was last written is returned as a binary value in a word formatted as follows:

Bits	Meaning
0–4	Day of the month
5–8	Month (1 = January, 2 = February, 3 = March, and so on)
9–15	Number of the year minus 1980

- Function 4EH is preferred to Function 11H (Find First File) because it fully supports pathnames.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

- 11H (Find First File)
- 12H (Find Next File)
- 1AH (Set DTA Address)
- 4FH (Find Next File)

Example

```
;*****;  
;  
;           Function 4EH: Find First File           ;  
;  
;           int find_first(ppathname,attr)         ;  
;           char *ppathname;                       ;  
;           int attr;                              ;  
;  
;           Returns 0 if a match was found,       ;  
;           otherwise returns error code.         ;  
;  
;*****;  
  
cProc  find_first,PUBLIC,ds  
parmDP ppathname  
parmW  attr  
cBegin  
    loadDP ds,dx,ppathname ; Get pointer to pathname.  
    mov    cx,attr         ; Get search attributes.  
    mov    ah,4eh          ; Set function code.  
    int    21h             ; Ask MS-DOS to look for a match.  
    jb    ff_err           ; Branch on error.  
    xor    ax,ax           ; Return 0 if no error.  
  
ff_err:  
cEnd
```

Interrupt 21H (33) Function 4FH (79)

2.0 and later

Find Next File

Function 4FH continues a search initiated by a previously successful call to Function 4EH (Find First File). The search is based on the pathname and attributes specified in the call to Function 4EH and uses information left in the current disk transfer area (DTA) by the call to Function 4EH or by a preceding call to Function 4FH.

To Call

AH = 4FH

DTA contains information from prior search with Function 4EH or Function 4FH.

Returns

If function is successful:

Carry flag is clear.

DTA is filled in as for a call to Function 4EH:

Offset	Length (bytes)	Value
00H	21	Reserved for use by MS-DOS in subsequent call to Function 4FH
15H	1	File attribute
16H	2	Time of last write
18H	2	Date of last write
1AH	2	Low word of file size
1CH	2	High word of file size
1EH	13	Filename and extension in ASCII form with blanks removed and period inserted between filename and extension

If function is not successful:

Carry flag is set.

AX = error code:

12H no more files, no match found, or no previous call to Function 4EH

Programmer's Notes

- If multiple calls to Function 4FH are used to find more than one matching file, the DTA setting (Function 1AH) and contents must be preserved because they provide information needed for continuing the search.
- The time at which the file was last written is returned as a binary value in a word formatted as follows:

Bits	Meaning
0–4	Number of seconds divided by 2
5–10	Minutes (0 through 59)
11–15	Hours, based on a 24-hour clock (0 through 23).

- The date on which the file was last written is returned as a binary value in a word formatted as follows:

Bits	Meaning
0–4	Day of the month
5–8	Month (1 = January, 2 = February, 3 = March, and so on)
9–15	Number of the year minus 1980

- Function 4FH is preferred to Function 12H (Find Next File) because it fully supports pathnames.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

11H (Find First File)
 12H (Find Next File)
 1AH (Set DTA Address)
 4EH (Find First File)

Example

```

;*****;
;
;           Function 4FH: Find Next File
;
;           int find_next()
;
;           Returns 0 if a match was found,
;           otherwise returns error code.
;
;*****;

```

(more)

Interrupt 21H Function 4FH

```
cProc  find_next,PUBLIC
cBegin
    mov     ah,4fh           ; Set function code.
    int     21h             ; Ask MS-DOS to look for the next
                           ; matching file.
    jnb     fn_err          ; Branch on error.
    xor     ax,ax           ; Return 0 if no error.
fn_err:
cEnd
```

Interrupt 21H (33) Function 54H (84)

2.0 and later

Get Verify Flag

Function 54H returns the current value of the MS-DOS verify flag.

To Call

AH = 54H

Returns

AL = verify flag:
 00H verify off; no read after write operation
 01H verify on; read after write operation

Programmer's Notes

- The default state of the verify flag is 00H (off).
- The state of the verify flag can be changed either through a call to Function 2EH (Set/Reset Verify Flag) or by the user with the VERIFY ON and VERIFY OFF commands.

Related Function

Function 2EH (Set/Reset Verify Flag)

Example

```

;*****;
;
;      Function 54H: Get Verify Flag      ;
;
;      int get_verify()                  ;
;
;      Returns current value of verify flag.  ;
;
;*****;

cProc  get_verify,PUBLIC
cBegin
    mov     ah,54h          ; Set function code.
    int     21h            ; Read flag from MS-DOS.
    cbw                    ; Clear high byte of return value.

cEnd
    
```


Interrupt 21H (33) Function 56H (86)

2.0 and later

Rename File

Function 56H renames a file and/or moves it to a new location in the hierarchical directory structure.

To Call

AH = 56H
DS:DX = segment:offset of existing ASCIIZ pathname for file
ES:DI = segment:offset of new ASCIIZ pathname for file

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
02H file not found
03H path not found
05H access denied
11H not the same device

Programmer's Notes

- The pathnames must be null-terminated ASCII strings (ASCIIZ).
- The directory paths specified in DS:DX and ES:DI need not be identical. Thus, specifying different directory paths effectively moves a file from one directory to another.
- Function 56H cannot be used to move a file to a different drive. Both the existing pathname and the new one must either contain the same drive identifier or default to the same drive.
- If Function 56H returns error code 05H, the cause can be any of the following:
 - The new pathname would move the file to the root directory, but the root directory is full.
 - A file with the new pathname already exists.
 - The user is on a network and has insufficient access to either the existing file or the new subdirectory.
- Unlike Function 17H (Rename File), Function 56H does not support the use of MS-DOS wildcard characters (? and *).

- Function 56H should not be used to rename open files. An open file should be closed with Function 10H (Close File with FCB) or 3EH (Close File) before Function 56H is called to rename it.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

17H (Rename File)

Example

```

;*****;
;
;           Function 56H: Rename File           ;
;
;           int rename(poldpath,pnewpath)       ;
;           char *poldpath,*pnewpath;          ;
;
;           Returns 0 if file moved OK,         ;
;           otherwise returns error code.       ;
;
;*****;

cProc  rename,PUBLIC,<ds,di>
parmDP poldpath
parmDP pnewpath
cBegin
loadDP es,di,pnewpath ; ES:DI = pointer to newpath.
loadDP ds,dx,poldpath ; DS:DX = pointer to oldpath.
mov    ah,56h         ; Set function code.
int    21h           ; Ask MS-DOS to rename file.
jb     rn_err        ; Branch on error.
xor    ax,ax         ; Return 0 if no error.

rn_err:
cEnd

```

Interrupt 21H (33) Function 57H (87)

2.0 and later

Get/Set Date/Time of File

Function 57H retrieves or sets the date and time of a file's directory entry.

To Call

AH = 57H
 AL = 00H get date and time
 01H set date and time
 BX = handle number

If AL = 01H:

CX = time; binary value formatted as follows:

Bits	Meaning
0-4	Number of seconds divided by 2
5-10	Minutes (0 through 59)
11-15	Hours, based on a 24-hour clock (0 through 23)

DX = date; binary value formatted as follows:

Bits	Meaning
0-4	Day of the month (1 through 31)
5-8	Month (1 = January, 2 = February, 3 = March, and so on)
9-15	Year minus 1980

Returns

If function is successful:

Carry flag is clear.

If AL was 00H on call:

CX = time file was last modified; format as described above

DX = date file was last modified; format as described above

If function is not successful:

Carry flag is set.

AX = error code:

01H invalid function (AL not 00H or 01H)
 06H invalid handle

Programmer's Notes

- Before the date and time in a file's directory entry can be retrieved or changed with Function 57H, a handle must be obtained by opening or creating the file using one of the following functions:
 - 3CH (Create File with Handle)
 - 3DH (Open File with Handle)
 - 5AH (Create Temporary File)
 - 5BH (Create New File)
- Use of Function 57H to retrieve the date and time of a file is preferable to examining the fields of an open FCB directly.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

2AH (Get Date)
 2BH (Set Date)
 2CH (Get Time)
 2DH (Set Time)

Example

```

;*****;
;
;      Function 57H: Get/Set Date/Time of File
;
;      long file_date_time(handle,func,packdate,packtime)
;      int handle,func,packdate,packtime;
;
;      Returns a long -1 for all errors, otherwise packs
;      date and time into a long integer,
;      date in high word, time in low word.
;
;*****;

cProc  file_date_time,PUBLIC
parmW  handle
parmB  func
parmW  packdate
parmW  packtime
cBegin
mov    bx,handle      ; Get handle.
mov    al,func        ; Get function: 0 = read, 1 = write.
mov    dx,packdate    ; Get date (if present).
mov    cx,packtime    ; Get time (if present).
mov    ah,57h         ; Set function code.
int    21h            ; Call MS-DOS.

```

(more)

```
        mov     ax,cx          ; Set DX:AX = date/time, assuming no
                                ; error.
        jnb    dt_ok          ; Branch if no error.
        mov     ax,-1         ; Return -1 for errors.
        cwd                    ; Extend the -1 into DX.
dt_ok:
cEnd
```

Interrupt 21H (33) Function 58H (88)

3.0 and later

Get/Set Allocation Strategy

Function 58H retrieves or sets the method MS-DOS uses to allocate memory blocks for a process that issues a memory-allocation request.

To Call

AH = 58H
 AL = 00H get allocation strategy
 01H set allocation strategy

If AL = 01H:

BX = allocation strategy:
 00H use first (lowest available) block that fits
 01H use block that fits best
 02H use last (highest available) block that fits

Returns

If function is successful:

Carry flag is clear.

If AL was 00H on call:

AX = allocation-strategy code:
 00H first fit
 01H best fit
 02H last fit

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function (AL not 00H or 01H)

Programmer's Notes

- Allocation strategies determine how MS-DOS finds and allocates a block of memory to an application that issues a memory-allocation request with either Function 48H (Allocate Memory Block) or Function 4AH (Resize Memory Block).

The three strategies are carried out as follows:

- First fit (the default): MS-DOS works upward from the lowest available block and allocates the first block it encounters that is large enough to satisfy the request for memory. This strategy is followed consistently, even if the block allocated is much larger than required.

- Best fit: MS-DOS searches all available memory blocks and then allocates the smallest block that satisfies the request, regardless of its location in the empty-block chain. This strategy maximizes the use of dynamically allocated memory at a slight cost in speed of allocation.
- Last fit (the reverse of first fit): MS-DOS works downward from the highest available block and allocates the first block it encounters that is large enough to satisfy the request for memory. This strategy is followed consistently, even if the block allocated is much larger than required.
- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

48H (Allocate Memory Block)

4AH (Resize Memory Block)

Example

```

;*****;
;
;      Function 58H: Get/Set Allocation Strategy      ;
;
;      int alloc_strategy(func, strategy)           ;
;      int func, strategy;                         ;
;
;      Strategies:                                  ;
;      0: First fit                                 ;
;      1: Best fit                                  ;
;      2: Last fit                                  ;
;
;      Returns -1 for all errors, otherwise         ;
;      returns the current strategy.               ;
;
;*****;

cProc  alloc_strategy, PUBLIC
parmB  func
parmW  strategy
cBegin
mov    al, func          ; AL = get/set selector.
mov    bx, strategy     ; BX = new strategy (for AL = 01H).
mov    ah, 58h          ; Set function code.
int    21h              ; Call MS-DOS.
jnb    no_err           ; Branch if no error.
mov    ax, -1           ; Return -1 for all errors.

no_err:
cEnd

```

Interrupt 21H (33) Function 59H (89)

3.0 and later

Get Extended Error Information

Function 59H returns extended error information, including a suggested response, for the function call immediately preceding it.

To Call

AH = 59H
BX = 00H

Returns

AX = extended error code:

00H	no error encountered
01H	invalid function number
02H	file not found
03H	path not found
04H	too many files open; no handles available
05H	access denied
06H	invalid handle
07H	memory control blocks destroyed
08H	insufficient memory
09H	invalid memory-block address
0AH	invalid environment
0BH	invalid format
0CH	invalid access code
0DH	invalid data
0EH	reserved
0FH	invalid disk drive
10H	attempt to remove current directory
11H	device not the same
12H	no more files
13H	write-protected disk
14H	unknown unit
15H	drive not ready
16H	invalid command
17H	data error based on cyclic redundancy check (CRC)
18H	length of request structure invalid
19H	seek error
1AH	non-MS-DOS disk
1BH	sector not found

1CH	printer out of paper
1DH	write fault
1EH	read fault
1FH	general failure
20H	sharing violation
21H	lock violation
22H	invalid disk change
23H	FCB unavailable
24H	sharing buffer exceeded
25-31H	reserved
32H	unsupported network request
33H	remote machine not listening
34H	duplicate name on network
35H	network name not found
36H	network busy
37H	device no longer exists on network
38H	net BIOS command limit exceeded
39H	error in network adapter hardware
3AH	incorrect response from network
3BH	unexpected network error
3CH	remote adapt incompatible
3DH	print queue full
3EH	queue not full
3FH	not enough room for print file
40H	network name deleted
41H	access denied
42H	incorrect network device type
43H	network name not found
44H	network name limit exceeded
45H	net BIOS session limit exceeded
46H	temporary pause
47H	network request not accepted
48H	print or disk redirection paused
49-4FH	reserved
50H	file already exists
51H	reserved
52H	cannot make directory
53H	failure on Interrupt 24H (critical error)
54H	out of structures
55H	already assigned
56H	invalid password
57H	invalid parameter
58H	net write fault

BH = error class:

01H	out of resource (such as storage)
02H	temporary situation, expected to end; not an error
03H	authorization problem
04H	internal error in system software
05H	hardware failure
06H	system-software failure, such as missing or incorrect configuration files; not the fault of the active process
07H	application-program error
08H	file or item not found
09H	file or item of invalid format or type or otherwise unsuitable
0AH	file or item interlocked
0BH	drive contains wrong disk, disk has bad spot, or other problem with storage medium
0CH	already exists
0DH	unknown

BL = suggested action:

01H	perform a reasonable number of retries before prompting user to choose Abort or Ignore in response to error message
02H	perform a reasonable number of retries, with pauses between, before prompting user to choose Abort or Ignore in response to error message
03H	prompt user to enter corrected information, such as drive letter or filename
04H	clean up and exit application
05H	exit immediately without cleanup
06H	ignore; informational error
07H	prompt user to remove cause of error (for example, change disks) and then retry

CH = location of error:

01H	unknown
02H	block device
03H	network
04H	serial device
05H	memory related

Programmer's Notes

- The extended error codes returned by Function 59H correspond to the error values returned in AX by functions in MS-DOS versions 2.0 and later that set the carry flag on error. Versions 2.x of MS-DOS, however, provide a smaller set of error codes (01H through 12H) than do later versions.

Thus, although Function 59H itself is not available in versions of MS-DOS earlier than 3.0, the matching of error codes to earlier versions helps ensure downward compatibility. Function 59H was also designed to be open-ended so that additional error codes could be incorporated as needed. As a result, processes should remain flexible

in their use of this function and should not rely on a fixed set of code numbers for error detection.

- Function 59H is useful in the following situations:
 - When MS-DOS encounters a hardware-related error condition and shifts control to an Interrupt 24H handler that has been created by the programmer
 - When a handle-related function sets the carry flag to indicate an error or when an FCB-related function indicates an error by returning 0FFH in the AL register
- If a function call results in an error, Function 59H returns meaningful information only if it is the next call to MS-DOS. An intervening call to another MS-DOS function, whether explicit or indirect, causes the error value for the unsuccessful function to be lost.
- Unlike most MS-DOS functions, Function 59H alters some registers that are not used to return results: CL, DX, SI, DI, ES, and DS. These registers must be preserved before a call to Function 59H if their contents are needed later.

Related Functions

None

Example

```

;*****;
;
;      Function 59H: Get Extended Error Information
;
;      int extended_error(err,class,action,locus)
;          int *err;
;          char *class,*action,*locus;
;
;      Return value is same as err.
;
;*****;

```

```

cProc   extended_error,PUBLIC,<ds,si,di>
parmDP  perr
parmDP  pclass
parmDP  paction
parmDP  plocus
cBegin
    push    ds            ; Save DS.
    xor     bx,bx
    mov     ah,59h        ; Set function code.
    int     21h          ; Request error info from MS-DOS.
    pop     ds            ; Restore DS.
    loadDP  ds,si,perr    ; Get pointer to err.
    mov     [si],ax       ; Store err.
    loadDP  ds,si,pclass  ; Get pointer to class.
    mov     [si],bh       ; Store class.
    loadDP  ds,si,paction ; Get pointer to action.
    mov     [si],bl       ; Store action.
    loadDP  ds,si,plocus  ; Get pointer to locus.
    mov     [si],ch       ; Store locus.
cEnd

```

Interrupt 21H (33) Function 5AH (90)

3.0 and later

Create Temporary File

Function 5AH uses the system clock to create a unique filename, appends the filename to the specified path, opens the temporary file, and returns a file handle that can be used for subsequent file operations.

To Call

AH = 5AH
 CX = file attribute:
 00H normal file
 01H read-only file
 02H hidden file
 04H system file
 DS:DX = segment:offset of ASCIIZ path, ending with a backslash character (\) and followed by 13 bytes of memory (to receive the generated filename)

Returns

If function is successful:

Carry flag is clear.

AX = handle
 DS:DX = segment:offset of full pathname for temporary file

If function is not successful:

Carry flag is set.

AX = error code:
 03H path not found
 04H too many open files; no handle available
 05H access denied

Programmer's Notes

- Only the drive and path to use for the new file should be specified in the buffer pointed to by DS:DX. The function appends an eight-character filename that is generated from the system time.
- Function 5AH is valuable in such situations as print spooling on a network, where temporary files are created by many users.
- The input string representing the path for the temporary file must be a null-terminated ASCII string (ASCIIZ).
- In networking environments running under MS-DOS version 3.1 or later, MS-DOS opens the temporary file in compatibility mode.

- MS-DOS does not delete temporary files; applications must do this for themselves.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

- 16H (Create File with FCB)
- 3CH (Create File with Handle)
- 5BH (Create New File)

Example

```

;*****;
;
;           Function 5AH: Create Temporary File           ;
;
;           int create_temp(ppathname,attr)               ;
;           char *ppathname;                               ;
;           int attr;                                      ;
;
;           Returns -1 if file was not created,           ;
;           otherwise returns file handle.                ;
;
;*****;

cProc   create_temp,PUBLIC,ds
parmDP  ppathname
parmW   attr
cBegin

    loadDP ds,dx,ppathname ; Get pointer to pathname.
    mov    cx,attr         ; Set function code.
    mov    ah,5ah          ; Ask MS-DOS to make a new file with
                          ; a unique name.
    int    21h             ; Ask MS-DOS to make a tmp file.
    jnb   ct_ok            ; Branch if MS-DOS returned handle.
    mov    ax,-1           ; Else return -1.

ct_ok:
cEnd

```

Interrupt 21H (33) Function 5BH (91)

3.0 and later

Create New File

Function 5BH creates a new file with the specified pathname. This function operates like Function 3CH (Create File with Handle) but fails if the pathname references a file that already exists.

To Call

AH = 5BH
 CX = file attribute:
 00H normal file
 01H read-only file
 02H hidden file
 04H system file
 DS:DX = segment:offset of ASCII pathname

Returns

If function is successful:

Carry flag is clear.

AX = handle

If function is not successful:

Carry flag is set.

AX = error code:
 03H path not found
 04H too many open files; no handle available
 05H access denied
 50H file already exists

Programmer's Notes

- The pathname must be a null-terminated ASCII string (ASCIIZ).
- In networking environments running under MS-DOS version 3.1 or later, the file is opened in compatibility mode. Function 5BH fails, however, if the user does not have Create access to the directory that is to contain the file.
- Function 5BH can be used to implement semaphores in the form of files across a local area network or in a multitasking environment. If the function succeeds, the semaphore has been acquired. To release the semaphore, the application simply deletes the file.

- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

- 16H (Create File with FCB)
- 3CH (Create File with Handle)
- 5AH (Create Temporary File)

Example

```

;*****
;
;      Function 5BH: Create New File
;
;      int create_new(ppathname,attr)
;      char *ppathname;
;      int attr;
;
;      Returns -2 if file already exists,
;              -1 for all other errors,
;              otherwise returns file handle.
;
;*****

cProc   create_new,PUBLIC,ds
parmDP  ppathname
parmW   attr
cBegin
        loadDP  ds,dx,ppathname ; Get pointer to pathname.
        mov     cx,attr         ; Get new file's attribute.
        mov     ah,5bh          ; Set function code.
        int     21h             ; Ask MS-DOS to make a new file.
        jnb    cn_ok           ; Branch if MS-DOS returned handle.
        mov     bx,-2
        cmp     al,80           ; Did file already exist?
        jz     ae_err          ; Branch if so.
        inc     bx              ; Change -2 to -1.
ae_err:
        mov     ax,bx          ; Return error code.
cn_ok:
cEnd

```

Interrupt 21H (33) Function 5CH (92)

3.0 and later

Lock/Unlock File Region

Function 5CH enables a process running in a networking or multitasking environment to lock or unlock a range of bytes in an open file.

To Call

AH	= 5CH	
AL	= 00H	lock region
	01H	unlock region
BX	= handle	
CX:DX	= 4-byte integer specifying beginning of region to be locked or unlocked	(offset in bytes from beginning of file)
SI:DI	= 4-byte integer specifying length of region (measured in bytes)	

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX	= error code:	
	01H	invalid function (AL not 00H or 01H or file sharing not loaded)
	06H	invalid handle
	21H	lock violation
	24H	sharing buffer exceeded

Programmer's Notes

- A process that either closes a file containing a locked region or terminates with the file open leaves the file in an undefined state. Under either condition, MS-DOS might handle the file erratically. If the process can be terminated by Interrupt 23H (Control-C) or 24H (critical error), these interrupts should be trapped so that any locked regions in files can be unlocked before the process terminates.
- Locking a portion of a file with Function 5CH denies all other processes both read and write access to the specified region of the file. This restriction also applies when open file handles are passed to a child process with Function 4BH (Load and Execute Program). Duplicate file handles created with Function 45H (Duplicate File Handle) and 46H (Force Duplicate File Handle), however, are allowed access to locked regions of a file within the current process.
- Locking a region that goes beyond the end of a file does not cause an error.

- Function 5CH is useful primarily in ensuring that competing programs or processes do not interfere while a record is being updated. Locking at the file level is provided by the sharing parameter in Function 3DH (Open File with Handle).
- Function 5CH can also be used to check the lock status of a file. If an attempt to lock a needed portion of a file fails and error code 21H is returned in the AX register, the region is already locked by another process.
- Any region locked with a call to Function 5CH must also be unlocked, and the same 4-byte integer values must be used for each operation. Two adjacent regions of a file cannot be locked separately and then be unlocked with a single unlock call. If the region to unlock does not correspond exactly to a locked region, Function 5CH returns error code 21H.
- The length of time needed to hold locks can be minimized with the transaction-oriented programming model. This concept requires defining and performing an update in a uniform manner: Assert lock, read data, change data, remove lock.
- If file sharing is not loaded, an application receives a 01H (function number invalid) error status when it attempts to lock a file. An immediate call to Function 59H returns the error locus as an unknown or a serial device.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

- 45H (Duplicate File Handle)
- 46H (Force Duplicate File Handle)
- 4BH (Load and Execute Program) [EXEC]

Example

```

;*****;
;
;           Function 5CH: Lock/Unlock File Region           ;
;
;           int locks(handle,onoff,start,length)           ;
;               int handle,onoff;                         ;
;               long start,length;                        ;
;
;           Returns 0 if operation was successful,         ;
;           otherwise returns error code.                 ;
;
;*****;

cProc   locks,PUBLIC,<si,di>
parmW   handle
parmB   onoff
parmD   start
parmD   length

```

(more)

```
cBegin      mov     al,onoff      ; Get lock/unlock flag.
            mov     bx,handle    ; Get file handle.
            les     dx,start     ; Get low word of start.
            mov     cx,es       ; Get high word of start.
            les     di,length    ; Get low word of length.
            mov     si,es       ; Get high word of length.
            mov     ah,5ch      ; Set function code.
            int     21h         ; Make lock/unlock request.
            jb     lk_err       ; Branch on error.
            xor     ax,ax        ; Return 0 if no error.
lk_err:
cEnd
```

Interrupt 21H (33) Function 5EH (94) Subfunction 00H

3.1 and later

Network Machine Name/Printer Setup: Get Machine Name

If Microsoft Networks is running, Function 5EH Subfunction 00H retrieves the network name of the local computer.

To Call

AH = 5EH
AL = 00H
DS:DX = segment:offset of 16-byte buffer

Returns

If function is successful:

Carry flag is clear.

CH = validity of machine name:
00H invalid
nonzero valid
CL = NETBIOS number assigned to machine name
DS:DX = segment:offset of ASCII machine name

If function is not successful:

Carry flag is set.

AX = error code:
01H invalid function; Microsoft Networks not running

Programmer's Notes

- The NETBIOS number in CL and the name at DS:DX are valid only if the value returned in CH is nonzero.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

5FH (Get/Make Assign List Entry)

Example

None

Interrupt 21H (33)

3.1 and later

Function 5EH (94) Subfunctions 02H and 03H

Network Machine Name/Printer Setup: Set Printer Setup;
Get Printer Setup

Function 5EH Subfunctions 02H and 03H respectively set and get the setup string that MS-DOS adds to the beginning of a file sent to a network printer.

To Call

AH = 5EH
 AL = 02H set printer setup string
 03H get printer setup string
 BX = assign-list index number (obtained with Function 5FH Subfunction 02H)

If AL = 02H:

CX = length of setup string in bytes (64 bytes maximum)
 DS:SI = segment:offset of ASCII setup string

If AL = 03H:

ES:DI = segment:offset of 64-byte buffer to receive string

Returns

If function is successful:

Carry flag is clear.

If AL was 03H on call:

CX = length of printer setup string in bytes
 ES:DI = segment:offset of ASCII printer setup string

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid subfunction

Programmer's Notes

- Function 5EH Subfunctions 02H and 03H enable multiple users on a network to configure a shared printer as required. The assign-list number is an index to a table that identifies the printer as a device on the network. A process can determine the assign-list number for the printer by using Function 5FH Subfunction 02H (Get Assign-List Entry).
- Error code 01H in the AX register may indicate either that Microsoft Networks is not running or that an invalid subfunction was selected.

- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

5FH (Get/Make Assign-List Entry)

Example

```

;*****;
;
;      Function 5EH Subfunction 02H:
;      Set Printer Setup
;
;      int printer_setup(index,pstring,len)
;      int  index;
;      char *pstring;
;      int  len;
;
;      Returns 0, otherwise returns -1 for all errors.
;
;*****;

cProc  printer_setup,PUBLIC,<ds,si>
parmW  index
parmDP pstring
parmW  len
cBegin
mov    bx,index      ; BX = index of a net printer.
loadDP ds,si,pstring ; DS:SI = pointer to string.
mov    cx,len        ; CX = length of string.
mov    ax,5e02h      ; Set function code.
int    21h           ; Set printer prefix string.
mov    al,0           ; Assume no error.
jnb    ps_ok          ; Branch if no error,
mov    al,-1          ; Else return -1.
ps_ok:
cbw
cEnd

```

Interrupt 21H (33) Function 5FH (95) Subfunction 02H

3.1 and later

Get/Make Assign-List Entry: Get Assign-List Entry

Function 5FH Subfunction 02H obtains the local and remote (network) names of a device. To find the names, MS-DOS uses the device's user-assigned index number (set with Function 5FH Subfunction 03H) to search a table of redirected devices on the network. Microsoft Networks must be running with file sharing loaded for this subfunction to operate successfully.

To Call

AH = 5FH
 AL = 02H
 BX = assign-list index number
 DS:SI = segment:offset of 16-byte buffer for local (device) name
 ES:DI = segment:offset of 128-byte buffer to receive remote (network) name

Returns

If function is successful:

Carry flag is clear.

BH = device status:
 00H valid device
 01H invalid device
 BL = device type:
 03H printer
 04H drive
 CX = user data
 DS:SI = segment:offset of ASCII string representing local device name
 ES:DI = segment:offset of ASCII string representing network name

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function or Microsoft Networks not running
 12H no more files

Programmer's Notes

- All strings returned by this subfunction are null-terminated ASCII strings (ASCII).
- A successful call to this subfunction destroys the contents of the DX and BP registers.

- Function 59H (Get Extended Error Information) provides further information on any error — in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

5EH Subfunction 00H (Get Machine Name)

Example

```

;*****;
;
;   Function 5FH Subfunction 02H:
;           Get Assign-List Entry
;
;   int get_alist_entry(index,
;           plocalname,premotename,
;           puservalue,ptype)
;   int index;
;   char *plocalname;
;   char *premotename;
;   int *puservalue;
;   int *ptype;
;
;   Returns 0 if the requested assign-list entry is found,
;   otherwise returns error code.
;
;*****;

cProc  get_alist_entry,PUBLIC,<ds,si,di>
parmW  index
parmDP plocalname
parmDP premotename
parmDP puservalue
parmDP ptype
cBegin
mov     bx,index           ; Get list index.
loadDP ds,si,plocalname  ; DS:SI = pointer to local name
                        ; buffer.
loadDP es,di,premotename ; ES:DI = pointer to remote name
                        ; buffer.
mov     ax,5f02h          ; Set function code.
int     21h               ; Get assign-list entry.
jb     ga_err             ; Exit on error.
xor     ax,ax             ; Else return 0.
loadDP ds,si,puservalue  ; Get address of user value.
mov     [si],cx           ; Store user value.
loadDP ds,si,ptype       ; Get address of type.
mov     bh,0
mov     [si],bx           ; Store device type to type.

ga_err:
cEnd

```

Interrupt 21H (33) Function 5FH (95) Subfunction 03H

3.1 and later

Get/Make Assign-List Entry: Make Assign-List Entry

Function 5FH Subfunction 03H redirects a local printer or disk drive to a network device and establishes an assign-list index number for the redirected device. Microsoft Networks must be running with file sharing loaded for this subfunction to operate successfully.

To Call

AH = 5FH
 AL = 03H
 BL = device type:
 03H printer
 04H drive
 CX = user data
 DS:SI = segment:offset of 16-byte ASCIIZ local device name
 ES:DI = segment:offset of 128-byte ASCIIZ remote (network) device name
 and password in the form

machine name\pathname,null,password,null

For example:

```
string db '\\mymach\wp',0,'blibbet',0
```

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function or Microsoft Networks not running
 03H path not found
 05H access denied
 08H insufficient memory
 0FH redirection paused on server
 12H no more files

Programmer's Notes

- The strings used by this subfunction must be null-terminated ASCII strings (ASCIIZ). The ASCIIZ string pointed to by ES:DI (the destination, or remote, device) cannot be more than 128 bytes including the password, which can be a maximum of 8 characters. If the password is omitted, the pathname must be followed by 2 null bytes.

- If BL = 03H, the string pointed to by DS:SI must be one of the following printer names: PRN, LPT1, LPT2, or LPT3. If the call is successful, output is redirected to a network print spooler, which must be named in the destination string. For printer redirection, MS-NET intercepts Interrupt 17H (BIOS Printer I/O). When redirection for a printer is canceled, all printing is sent to the first local printer (LPT1).
If BL = 04H, the string pointed to by DS:SI can be a drive letter followed by a colon, such as E:, or it can be a null string. If the string represents a valid drive, a successful call redirects drive requests to the network directory named in the destination string. If DS:SI points to a null string, MS-DOS attempts to provide access to the network directory named in the destination string without redirecting any device.
- Only printer and disk devices are supported in MS-DOS versions 3.1 and later. COM1 and COM2 are not supported for network redirection, nor are the standard output or standard error devices supported.
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

5EH Subfunction 00H (Get Machine Name)

Example

```

;*****;
;
;   Function 5FH Subfunction 03H:
;   Make Assign-List Entry
;   int add_list_entry(psrcname,pdestname,uservalue,type)
;   char *psrcname,*pdestname;
;   int  uservalue,type;
;
;   Returns 0 if new assign-list entry is made, otherwise
;   returns error code.
;
;*****;

cProc  add_list_entry,PUBLIC,<ds,si,di>
parmDP psrcname
parmDP pdestname
parmW  uservalue
parmW  type
cBegin
    mov    bx,type      ; Get device type.
    mov    cx,uservalue ; Get uservalue.
    loadDP ds,si,psrcname ; DS:SI = pointer to source name.
    loadDP es,di,pdestname ; ES:DI = pointer to destination name.
    mov    ax,5f03h     ; Set function code.
    int    21h          ; Make assign-list entry.
    jb    aa_err        ; Exit if there was some error.
    xor    ax,ax        ; Else return 0.
aa_err:
cEnd

```

Int 21H (33)

3.1 and later

Function 5FH (95) Subfunction 04H

Get/Make Assign-List Entry: Cancel Assign-List Entry

Function 5FH Subfunction 04H cancels the redirection of a local device to a network device previously established with Function 5FH Subfunction 03H (Make Assign-List Entry). Microsoft Networks must be running with file sharing loaded for this subfunction to operate successfully.

To Call

AH = 5FH
 AL = 04H
 DS:SI = segment:offset of ASCIIZ device name or path

Returns

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:

01H	invalid function or Microsoft Networks not running
03H	path not found
05H	access denied
08H	insufficient memory
0FH	redirection paused on server
12H	no more files

Programmer's Notes

- The string pointed to by DS:SI must be a null-terminated ASCII string (ASCIIZ). This string can be any one of the following:
 - The letter, followed by a colon, of a redirected local drive. This function restores the drive letter to its original, physical meaning.
 - The name of a redirected printer: PRN, LPT1, LPT2, LPT3, or its machine-specific equivalent. This function restores the printer name to its original, physical meaning at the local workstation.
 - A string, beginning with two backslashes (\\) followed by the name of a network directory. This function terminates the connection between the local workstation and the directory specified in the string.

- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Function

5EH Subfunction 00H (Get Machine Name)

Example

```

;*****;
;
;   Function 5FH Subfunction 04H:
;           Cancel Assign-List Entry
;
;   int cancel_alist_entry(psrcname)
;           char *psrcname;
;
;   Returns 0 if assignment is canceled, otherwise returns
;   error code.
;
;*****;

cProc   cancel_alist_entry,PUBLIC,<ds,si>
parmDP  psrcname
cBegin
        loadDP  ds,si,psrcname  ; DS:SI = pointer to source name.
        mov     ax,5f04h        ; Set function code.
        int     21h            ; Cancel assign-list entry.
        jnb    ca_err          ; Exit on error.
        xor     ax,ax           ; Else return 0.
ca_err:
cEnd

```

Interrupt 21H (33) Function 62H (98)

3.0 and later

Get Program Segment Prefix Address

Function 62H gets the segment address of the program segment prefix (PSP) for the current process.

To Call

AH = 62H

Returns

BX = segment address of PSP for current process

Programmer's Notes

- The PSP is constructed by MS-DOS at the base of the memory allocated for a .COM or .EXE program being loaded into memory by the EXEC function, 4BH (Load and Execute Program). The PSP is 100H bytes and contains information useful to an executing program, including
 - The command tail
 - Default file control blocks (FCBs)
 - A pointer to the program's environment block
 - Previous addresses for MS-DOS Control-C, critical error, and terminate handlers
- Function 59H (Get Extended Error Information) provides further information on any error—in particular, the code, class, recommended corrective action, and locus of the error.

Related Functions

None

Example

```

;*****;
;
;      Function 62H: Get Program Segment Prefix Address      ;
;
;      int get_psp()                                         ;
;
;      Returns PSP segment.                                 ;
;
;*****;

```

(more)

Interrupt 21H Function 62H

```
cProc  get_psp,PUBLIC
cBegin
      mov     ah,62h      ; Set function code.
      int     21h        ; Get PSP address.
      mov     ax,bx      ; Return it in AX.
cEnd
```

Interrupt 21H (33) Function 63H (99)

2.25

Get Lead Byte Table

Function 63H, available only in MS-DOS version 2.25, includes three subfunctions that support 2-byte-per-character alphabets such as Kanji and Hangeul (Japanese and Korean characters sets). Subfunction 00H obtains the address of the legal lead byte ranges for the character sets; Subfunctions 01H and 02H set or obtain the value of the interim console flag, which determines whether interim characters are returned by certain console system calls.

To Call

AH	= 63H	
AL	= 00H	get lead byte table address
	01H	set or clear interim console flag
	02H	get interim console flag

If AL = 01H:

DL	= interim console flag:
	00H clear
	01H set

Returns

If function is successful:

Carry flag is clear.

If AL was 00H on call:

DS:SI	= segment:offset of lead byte table
-------	-------------------------------------

If AL was 02H on call:

DL	= value of interim console flag
----	---------------------------------

If function is not successful:

Carry flag is set.

AX	= error code:
	01H invalid function

Programmer's Notes

- Function 63H does not necessarily preserve any registers other than SS:SP, so register values should be saved before a call to this function. To avoid saving registers repeatedly, a process can either copy the table or save the pointer to the table for later use.

- The lead byte table contains pairs of bytes that represent the inclusive boundary values for the lead bytes of the specified alphabet. Because of the way bytes are ordered by the 8086 microprocessor family, the values must be read as byte values, not as word values.
- If the interim console flag is set (DL = 01H) by a program through a call to Function 63H, the following functions return interim character information on request:
 - 07H (Character Input Without Echo)
 - 08H (Unfiltered Character Input Without Echo)
 - 0BH (Check Keyboard Status)
 - 0CH (Flush Buffer, Read Keyboard), if Function 07H or 08H is requested in AL

Related Functions

None

Example

```

;*****;
;
; Function 63H: Get Lead Byte Table
;
; char far *get_lead_byte_table()
;
; Returns far pointer to table of lead bytes for multibyte
; characters. Will work only in MS-DOS 2.25!
;
;*****;

cProc get_lead_byte_table,PUBLIC,<ds,si>
cBegin
mov ax,6300h ; Set function code.
int 21h ; Get lead byte table.
mov dx,ds ; Return far pointer in DX:AX.
mov ax,si
cEnd

```

Interrupt 22H (34)

1.0 and later

Terminate Routine Address

The machine interrupt vector for Interrupt 22H (memory locations 0000:0088H through 0000:008BH) contains the address of the routine that receives control when the currently executing program terminates by means of Interrupt 20H, Interrupt 27H, or Interrupt 21H Function 00H, 31H, or 4CH.

To Call

This interrupt should never be issued directly.

Returns

Nothing

Programmer's Note

- The address in this vector is copied into offsets 0AH through 0DH of the program segment prefix (PSP) when a program is loaded but before it begins executing. The address is restored from the PSP (in case it was modified by the application) as part of MS-DOS's termination handling.

Example

None

Interrupt 23H (35)

1.0 and later

Control-C Handler Address

The machine interrupt vector for Interrupt 23H (memory locations 0000:008CH through 0000:008FH) contains the address of the routine that receives control when a Control-C (also Control-Break on IBM PC compatibles) is detected during any character I/O function and, if the Break flag is on, during most other MS-DOS function calls.

To Call

This interrupt should never be issued directly.

Returns

Nothing

Programmer's Notes

- The address in this vector is copied into offsets 0EH through 11H of the program segment prefix (PSP) when a program is loaded but before it begins executing. The address is restored from the PSP (in case it was modified by the application) as part of MS-DOS's termination handling.
- The initialization code for an application can use Interrupt 21H Function 25H (Set Interrupt Vector) to reset the Interrupt 23H vector to point to its own routine for Control-C handling. By installing its own Control-C handler, the program can avoid being terminated as a result of keyboard entry of a Control-C or Control-Break.
- When a Control-C is detected and the program's Interrupt 23H handler receives control, MS-DOS sets all registers to the original values they had when the function call that is being interrupted was made. The program's interrupt handler can then do any of the following:
 - Set a local flag for later inspection by the application (or take any other appropriate action) and then perform a return from interrupt (IRET) to return control to MS-DOS. (All registers must be preserved.) The MS-DOS function in progress is then restarted and proceeds to completion, and control finally returns to the application in the normal manner.
 - Take appropriate action and then perform a far return (RET FAR) to give control back to MS-DOS. MS-DOS uses the state of the carry flag to determine what action to take: If the carry flag is set, the application is terminated; if the carry flag is clear, the application continues in the normal manner.
 - Retain control by transferring to an error-handling routine within the application and then resume execution or take other appropriate action, never performing a RET FAR or IRET to end the interrupt-handling sequence. This option causes no harm to the system.
- Any MS-DOS function call can be used within the body of an Interrupt 23H handler.

Example

None

Interrupt 24H (36)

1.0 and later

Critical Error Handler Address

The machine interrupt vector for Interrupt 24H (memory locations 0000:0090H through 0000:0093H) contains the address of the routine that receives control when a critical error (usually a hardware error) is detected.

To Call

This interrupt should never be issued directly.

Returns

Nothing

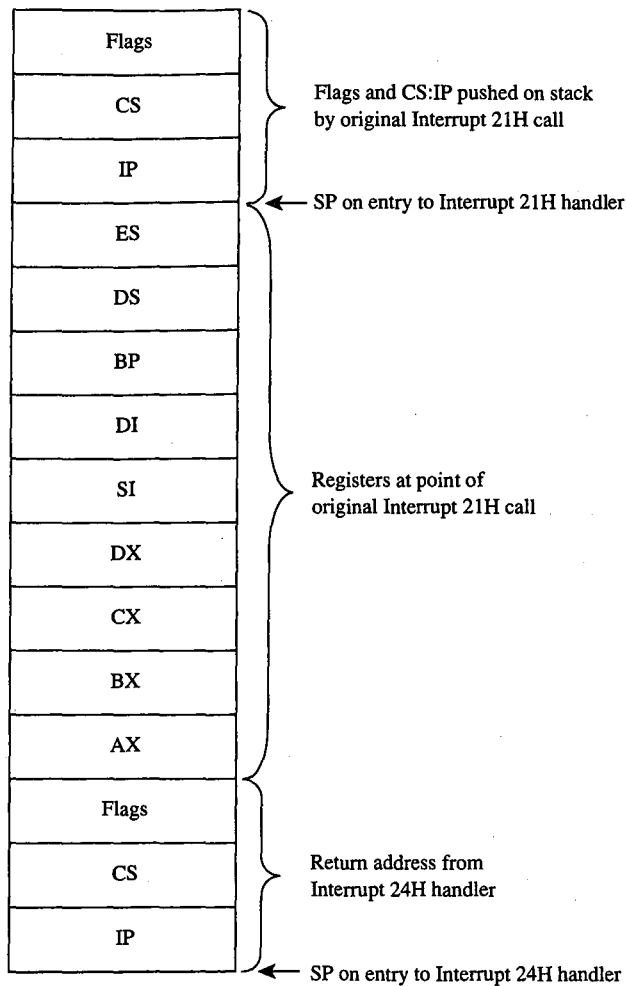
Programmer's Notes

- The address of this vector is copied into offsets 12H through 15H of the program segment prefix (PSP) when a program is loaded but before it begins executing. The address is restored from the PSP (in case it was modified by the application) as part of MS-DOS's termination handling.
- On entry to the critical error interrupt handler, bit 7 of register AH is clear (0) if the error was a disk I/O error; otherwise, it is set (1). BP:SI contains the address of a device-header control block from which additional information can be obtained. Interrupts are disabled. MS-DOS sets up the registers for a retry operation and one of the following error codes is in the lower byte of the DI register (the upper byte is undefined):

Code	Meaning
00H	Write-protect error
01H	Unknown unit
02H	Drive not ready
03H	Unknown command
04H	Data error (bad CRC)
05H	Bad request structure length
06H	Seek error
07H	Unknown media type
08H	Sector not found
09H	Printer out of paper
0AH	Write fault
0BH	Read fault
0CH	General failure
0FH	Invalid disk change

These are the same error codes returned by the device drivers in the request header.

- On a disk error, MS-DOS retries the operation three times before transferring to the Interrupt 24H handler.
- On entry to the Interrupt 24H handler, the stack is set up as follows:



- Interrupt 24H handlers must preserve the SS, SP, DS, ES, BX, CX, and DX registers. Only Interrupt 21H Functions 01H through 0CH, 30H, and 59H can be used by an Interrupt 24H handler; other calls will destroy the MS-DOS stack and its ability to retry or ignore an error.

- Before issuing a RETURN FROM INTERRUPT (IRET), the Interrupt 24H handler should place an action code in AL that will be interpreted by MS-DOS as follows:

Code	Meaning
00H	Ignore error.
01H	Retry operation.
02H	Terminate program through Interrupt 23H.
03H	Fail system call in progress (versions 3.1 and later).

- If an Interrupt 24H routine returns to the user program rather than to MS-DOS, it must restore the user program's registers, removing all but the last three words from the stack, and issue an IRET. Control returns to the instruction immediately following the Interrupt 21H function call that resulted in an error. This leaves MS-DOS in an unstable state until a call is made to an Interrupt 21H function higher than 0CH.

Example

None

Interrupt 25H (37)

1.0 and later

Absolute Disk Read

Interrupt 25H provides direct linkage to the MS-DOS BIOS module to read data from a logical disk sector into a specified memory location.

To Call

AL = drive number (0 = drive A, 1 = drive B, and so on)
CX = number of sectors to read
DX = starting relative (logical) sector number
DS:BX = segment:offset of disk transfer area (DTA)

Returns

If operation is successful:

Carry flag is clear.

If operation is not successful:

Carry flag is set.

AX = error code

Programmer's Notes

- Interrupt 25H might destroy all registers except the segment registers.
- When Interrupt 25H returns, the CPU flags originally pushed onto the stack by the INT 25H instruction are still on the stack. The stack must be cleared by a POPF or ADD SP,2 instruction to prevent uncontrolled stack growth and to make accessible any other values that were pushed onto the stack before the call to Interrupt 25H.
- Logical sector numbers are zero based and are obtained by numbering each disk sector sequentially from track 0, head 0, sector 1 and continuing until the last sector on the disk is counted. The head number is incremented before the track number. Because of interleaving, logically adjacent sectors might not be physically adjacent for some types of disks.
- The lower byte of the error code (AL) is the same error code that is returned in the lower byte of DI when an Interrupt 24H is issued. The upper byte (AH) contains one of the following codes:

Code	Meaning
80H	Device failed to respond
40H	Seek operation failure
20H	Controller failure

(more)

Code	Meaning
10H	Data error (bad CRC)
08H	Direct memory access (DMA) failure
04H	Requested sector not found
03H	Write-protect fault
02H	Bad address mark
01H	Bad command

- **Warning:** Interrupt 25H bypasses the MS-DOS file system. This function must be used with caution to avoid damaging the disk structure.

Example

```

;*****;
;                                           ;
;   Interrupt 25H: Absolute Disk Read      ;
;                                           ;
;   Read logical sector 1 of drive A into the memory area ;
;   named buff. (On most MS-DOS floppy disks, this sector ;
;   contains the beginning of the file allocation table.) ;
;                                           ;
;*****;

    mov     al,0           ; Drive A.
    mov     cx,1          ; Number of sectors.
    mov     dx,1          ; Beginning sector number.
    mov     bx,seg buff   ; Address of buffer.
    mov     ds,bx
    mov     bx,offset buff
    int     25h           ; Request disk read.
    jc     error          ; Jump if read failed.
    add     sp, 2         ; Clear stack.
    .
    .
    .
error:                               ; Error routine goes here.
    .
    .
buff     db     512 dup (?)

```

Interrupt 26H (38)

1.0 and later

Absolute Disk Write

Interrupt 26H provides direct linkage to the MS-DOS BIOS module to write data from a specified memory buffer to a logical disk sector.

To Call

AL = drive number (0 = drive A, 1 = drive B, and so on)
CX = number of sectors to write
DX = starting relative (logical) sector number
DS:BX = segment:offset of disk transfer area (DTA)

Returns

If operation is successful:

Carry flag is clear.

If operation is not successful:

Carry flag is set.

AX = error code

Programmer's Notes

- When Interrupt 26H returns, the CPU flags originally pushed onto the stack by the INT 26H instruction are still on the stack. The stack must be cleared by a POPF or ADD SP,2 instruction to prevent uncontrolled stack growth and to make accessible any other values that were pushed on the stack before the call to Interrupt 26H.
- Logical sector numbers are zero based and are obtained by numbering each disk sector sequentially from track 0, head 0, sector 1 and continuing until the last sector on the disk is counted. The head number is incremented before the track number. Because of interleaving, logically adjacent sectors might not be physically adjacent for some types of disks.
- The lower byte of the error code (AL) is the same error code that is returned in the lower byte of DI when an Interrupt 24H is issued. The upper byte (AH) contains one of the following codes:

Code	Meaning
80H	Device failed to respond
40H	Seek operation failure
20H	Controller failure
10H	Data error (bad CRC)

(more)

Code	Meaning
08H	Direct memory access (DMA) failure
04H	Requested sector not found
03H	Write-protect fault
02H	Bad address mark
01H	Bad command

- **Warning:** Interrupt 26H bypasses the MS-DOS file system. This function must be used with caution to avoid damaging the disk structure.

Example

```

;*****;
;
;   Interrupt 26H: Absolute Disk Write
;
;   Write the contents of the memory area named buff
;   into logical sector 3 of drive C.
;
;   WARNING: Verbatim use of this code could damage
;   the file structure of the fixed disk. It is meant
;   only as a general guide. There is, unfortunately,
;   no way to give a really safe example of this interrupt.
;
;*****;

    mov     al,2           ; Drive C.
    mov     cx,1           ; Number of sectors.
    mov     dx,3           ; Beginning sector number.
    mov     bx,seg buff    ; Address of buffer.
    mov     ds,bx
    mov     bx,offset buff
    int     26h           ; Request disk write.
    jc     error           ; Jump if write failed.
    add     sp,2           ; Clear stack.
    .
    .
error:
    .
    .
buff     db     512 dup (?) ; Data to be written to disk.

```


Interrupt 27H (39)

1.0 and later

Terminate and Stay Resident

Interrupt 27H terminates execution of the currently executing program but reserves part or all of its memory so that it will not be overlaid by the next transient program to be loaded.

To Call

- DX = offset of last byte plus 1 (relative to the program segment prefix, or PSP) of program to be protected
- CS = segment address of PSP

Returns

Nothing

Programmer's Notes

- In response to an Interrupt 27H call, MS-DOS takes the following actions:
 - Restores the termination vector (Interrupt 22H) from PSP:000AH.
 - Restores the Control-C vector (Interrupt 23H) from PSP:000EH.
 - With MS-DOS versions 2.0 and later, restores the critical error handler vector (Interrupt 24H) from PSP:0012H.
 - Transfers to the termination handler address.
- If the program is returning to COMMAND.COM rather than to another program, control transfers first to COMMAND.COM's resident portion, which reloads COMMAND.COM's transient portion (if necessary) and passes it control. If a batch file is in progress, the next line of the file is then fetched and interpreted; otherwise, a prompt is issued for the next user command.
- This interrupt is typically used to allow user-written drivers or interrupt handlers to be loaded as ordinary .COM or .EXE programs and then remain resident. Subsequent entrance to the code is by means of a hardware or software interrupt.
- The maximum amount of memory that can be reserved with this interrupt is 64 KB. Therefore, Interrupt 27H should be used only for applications that must run under MS-DOS versions 1.x.

With versions 2.0 and later, the preferred method to terminate and stay resident is to use Interrupt 21H Function 31H, which allows the program to reserve more than 64 KB of memory and does not require CS to contain the PSP address.

- Interrupt 27H should not be called by .EXE programs that are loaded into the high end of memory (that is, linked with the /HIGH switch), because this would reserve the memory that is ordinarily used by the transient portion of COMMAND.COM. If COMMAND.COM cannot be reloaded, the system will fail.

- Because execution of Interrupt 27H results in the restoration of the terminate routine (Interrupt 22H), Control-C (Interrupt 23H), and critical error (Interrupt 24H) vectors, it cannot be used to permanently install a user-written critical error handler.
- Interrupt 27H does not work correctly when DX contains values in the range FFF1H through FFFFH. In this case, MS-DOS discards the high bit of the contents of DX, resulting in 32 KB less resident memory than was actually requested by the program.

Example

```

;*****;
;
;   Interrupt 27H: Terminate and Stay Resident
;
;   Exit and stay resident, reserving enough memory
;   to protect the program's code and data.
;
;*****;

Start:  .
        .
        mov     dx,offset pgm_end ; DX = bytes to reserve.
        int    27h               ; Terminate, stay resident.
        .
        .
pgm_end equ    $
        end    start

```

Interrupt 2FH (47)

2.0 and later

Multiplex Interrupt

Interrupt 2FH with AH = 01H submits a file to the print spooler, removes a file from the print spooler's queue of pending files, or obtains the status of the printer. Other values for AH are used by various MS-DOS extensions, such as APPEND.

To Call

AH	= 01H	print spooler call
AL	= 00H	get installed status
	01H	submit file to be printed
	02H	remove file from print queue
	03H	cancel all files in queue
	04H	hold print jobs for status read
	05H	end hold for status read

If AL is 01H:

DS:DX = segment:offset of packet address

If AL is 02H:

DS:DX = segment:offset of ASCIIZ file specification

Returns

If operation is successful:

Carry flag is clear.

If AL was 00H on call:

AL	= status:
	00H not installed, OK to install
	01H not installed, not OK to install
	FFH installed

If AL was 04H on call:

DX = error count
DS:SI = segment:offset of print queue

If operation is not successful:

Carry flag is set.

AX	= error code:
	01H function invalid
	02H file not found
	03H path not found

(more)

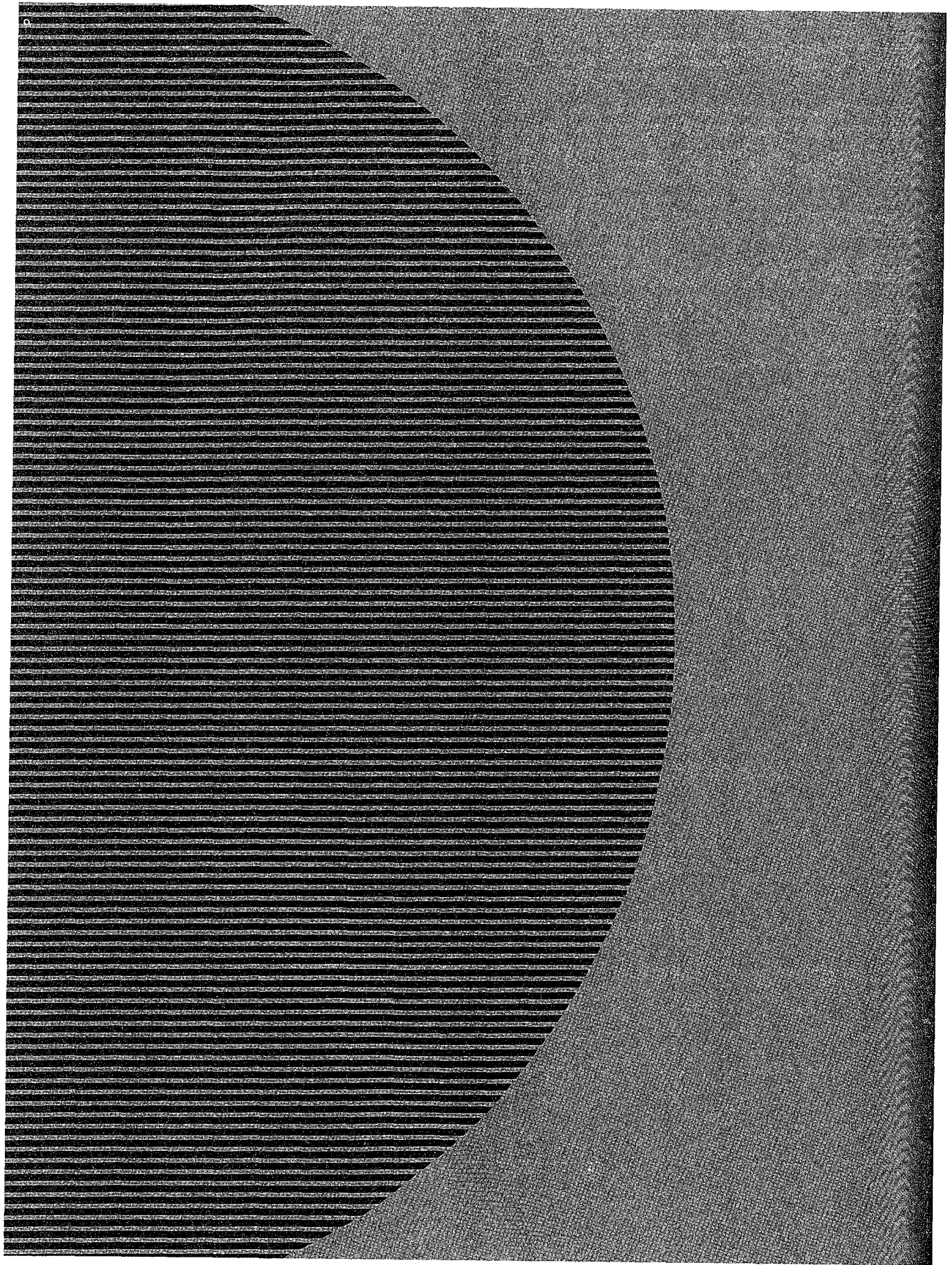
04H	too many open files
05H	access denied
08H	queue full
09H	spooler busy
0CH	name too long
0FH	drive invalid

Programmer's Notes

- For Subfunction 01H, the packet consists of 5 bytes. The first byte contains the level (must be zero), the next 4 bytes contain the doubleword address (segment and offset) of an ASCIIZ file specification. (The filename cannot contain wildcard characters.) If the file exists, it is added to the end of the print queue.
- For Subfunction 02H, wildcard characters (*and ?) are allowed in the file specification, making it possible to delete multiple files from the print queue with one call.
- For Subfunction 04H, the address returned for the print queue points to a series of filename entries. Each entry in the queue is 64 bytes and contains an ASCIIZ file specification. The first file specification in the queue is the one currently being printed. The last slot in the queue has a null (zero) in the first byte.

Example

None



Appendixes

Appendix A

MS-DOS Version 3.3

For the MS-DOS user, version 3.3 incorporates some long-awaited capabilities, runs faster in places, and requires about 9 KB more memory than version 3.2. Its most apparent changes, however, relate to a new, more flexible method of supporting different national languages. For the MS-DOS programmer, version 3.3 offers several enhancements in the areas of file management and internationalization support. This appendix offers an overview of these new features.

Version 3.3 User Considerations

MS-DOS version 3.3 has introduced several changes at the user level. A new external command, FASTOPEN, speeds up the filing system by keeping file locations in memory. A new batch command, CALL, lets a batch file call another batch file and, when that file terminates, continue execution with the next command in the original batch file rather than return to MS-DOS as in previous versions. Two commands previously present only in PC-DOS, COMP and SELECT, have been added to MS-DOS. Five commands have additional capabilities: APPEND, ATTRIB, BACKUP, FDISK, and MODE. In addition, the TIME and DATE commands automatically set the CMOS clock-calendar on the IBM PC/AT and PS/2 machines, making use of the separate SETUP program unnecessary for these functions. Changes to the national language support involve four new commands, three new options to the MODE command, two new or modified system information files, and two new device drivers. Each of these new or modified commands is discussed individually below.

The FASTOPEN command

When MS-DOS searches for a program file, it searches each directory specified in the PATH search path. A lengthy path that has to search many levels of a directory structure can make this a slow process. The FASTOPEN command loads a terminate-and-stay-resident (TSR) program that caches the locations of the most recently accessed directories and files on one or more fixed disks in the system. The number of files and directories to be cached is under the user's control; the default is 10. When it needs a file, MS-DOS looks first in the FASTOPEN list; if the file is found in the list, MS-DOS can bypass inspection of the search path specified by PATH. When the FASTOPEN list is filled and a new file is opened, the new file replaces the least recently used file on the FASTOPEN list.

The improvement in file-system performance depends on the number of open files and the frequency of file access. The FASTOPEN command can be entered only once during a session and, if desired, can be placed in the AUTOEXEC.BAT file.

The FASTOPEN command has two parameters:

FASTOPEN *drive*:[=*entries*][...]

The *drive* parameter is the drive letter, followed by a colon, of a fixed disk for which FASTOPEN is to keep track of the most recently accessed directories and files. More than one drive can be specified by separating the drive identifiers with spaces; the maximum is four drives. A drive associated with a JOIN, SUBST, or ASSIGN command cannot be specified, nor can a drive assigned to a network.

The optional *entries* parameter is the number of directory entries FASTOPEN is to keep in memory. The value of *entries* can be from 10 through 999; the default is 34. If more than one *entries* value is specified, their sum cannot exceed 999. Each entry subtracts 40 bytes from the RAM normally available to run application programs.

Examples: The following command tells MS-DOS to keep track of the last 50 directories and files on drive C:

```
C>FASTOPEN C:=50 <Enter>
```

The next command tells MS-DOS to keep track of the last 34 files on drives C and D:

```
C>FASTOPEN C: D: <Enter>
```

Changes to batch-file processing

Batch-file processing also gains power in MS-DOS version 3.3. The user can now suppress the echo of all batch commands and call one batch file from another without terminating the first batch file.

@

With MS-DOS version 3.3, any line in a batch file preceded by @ is not echoed to the screen when the batch file is executed.

CALL

A batch file no longer needs to load an additional copy of COMMAND.COM in order to execute another batch file and return control to the calling batch file. The CALL command executes a batch file and returns to the next command in the calling batch file.

CALL commands can be nested. If an exit condition is provided, a batch file can even call itself; however, the input or output of a called batch file cannot be redirected or piped.

The CALL command has two parameters:

CALL *batch-file* [*parameters*]

The *batch-file* parameter is the name of the batch file to be executed. The file must be in the current drive and directory or in a drive and/or directory specified in the command path.

The optional *parameters* parameter represents any parameters that may be required by *batch-file*.

Example: Suppose the batch file SORTFILE.BAT accepts one parameter. The following command calls SORTFILE.BAT, specifying NAMES.TXT as the parameter:

```
CALL SORTFILE NAMES.TXT
```

If NAMES.TXT was specified as a command-line parameter to the *calling* batch file, the CALL command could be

```
CALL SORTFILE %1
```

Commands from PC-DOS

Two commands have been added to MS-DOS from earlier versions of PC-DOS: COMP, present in PC-DOS version 1.0, and SELECT, present in PC-DOS version 2.0.

COMP

The COMP command compares two files or sets of files and reports any differences encountered. FC, a similar file-comparison command present in MS-DOS versions 2.0 and later, is still included with MS-DOS 3.3. See USER COMMANDS: COMP; FC.

Syntax for the COMP command is

```
COMP [drive:][filename1] [drive:][filename2]
```

The optional *drive* parameter is the drive letter, followed by a colon, of the drive containing the file to be compared. The *filename1* parameter is the name and location of the file to compare to *filename2*; *filename2* is the name and location of the file to be compared against. Both filenames can be preceded by a path; wildcard characters are permitted in either filename.

Example: The following command tells MS-DOS to compare the file NEWFILE.TXT in the current drive and directory to the file OLDFILE.TXT in the \ARCHIVE directory on drive D and report any differences encountered:

```
C>COMP NEWFILE.TXT D:\ARCHIVE\OLDFILE.TXT <Enter>
```

SELECT

The SELECT command creates a system disk with the time format, date format, and keyboard layout configured for a selected country. The syntax for SELECT is

```
SELECT [[drive1:] [drive2:][path]] [country][keyboard]
```

The optional *drive1* parameter is the drive containing a disk with the MS-DOS operating-system files, the FORMAT program, and the country configuration files. The *drive2* parameter is the drive containing the disk to be formatted with the country-specific information; this drive specifier can be followed by a path. The *country* parameter is a code

that selects the date and time format; the information is taken from the COUNTRY.SYS system file. The *keyboard* parameter is a code that selects the desired keyboard layout. See KEYB below.

The SELECT command

- Formats the target disk.
- Creates CONFIG.SYS and AUTOEXEC.BAT files on the target disk.
- Copies the contents of the source disk to the destination disk.

Example: The following command, which assumes drive A contains a valid system disk and drive B contains the disk to be formatted, creates a bootable system disk that includes country-specific information and keyboard layout for Germany:

```
C>SELECT A: B: 049 GR <Enter>
```

Enhanced commands

Several existing MS-DOS user commands have been given expanded capabilities in version 3.3. These are presented alphabetically in the next few pages. See USER COMMANDS: APPEND; ATTRIB; BACKUP; FDISK; MODE.

APPEND

The APPEND command specifies a search path for data files — files whose extensions are neither .COM, .EXE, nor .BAT — similar to the command path specified by the PATH command, which searches only for executable files *with* those extensions. APPEND has three forms, depending on whether it is being entered for the first time. When it is entered the first time, the APPEND command now has two optional switches:

```
APPEND [/E] [/X]
```

The /E switch makes the data path part of the environment, like the command path. The data path can then be displayed or changed with both the SET and APPEND commands and is inherited by child processes. (However, any changes made to the data path by the child process are lost when the child returns to its parent process.)

The /X switch causes calls to the Find First File functions (Interrupt 21H Functions 11H and 4EH) and the EXEC function (Interrupt 21H Function 4BH) to search the data path. If /X is not specified, only Interrupt 21H Function 0FH (Open File with FCB), Interrupt 21H Function 23H (Get File Size), and Interrupt 21H Function 3DH (Open File with Handle) system calls search the data path.

If either /X or /E is specified the first time APPEND is entered, a pathname cannot be included.

Subsequent uses of the command must take the form

```
APPEND [[drive:]path] [;[drive:]path ...]
```

or

```
APPEND ;
```

The *path* parameter is the name of a directory that is to be made part of the data path. The user can specify as many directory names as will fit in the 128 characters of the command line. Entries must be separated by semicolons. If APPEND is followed only by a semicolon, any previous APPEND paths are deleted.

Example: The following two APPEND commands make the data path part of the environment and put the directories C:\WORD\PROPOSAL, C:\WORD\REPORTS, and C:\123\BUDGET in the data path:

```
C>APPEND /E <Enter>
C>APPEND C:\WORD\PROPOSAL;C:\WORD\REPORTS;C:\123\BUDGET <Enter>
```

Because the data path usually involves frequently used directories, the APPEND command ordinarily is placed in the AUTOEXEC.BAT file.

Note: APPEND is a new command in PC-DOS version 3.3.

ATTRIB

The /S switch has been added to the ATTRIB command so that any attribute changes can be applied to all files in subdirectories contained in the specified directory.

Example: The following command sets the read-only attribute of all files in the directory C:\DOS and in all its subdirectories:

```
C>ATTRIB +R C:/DOS /S <Enter>
```

BACKUP

A formatting parameter has been added to the BACKUP command in MS-DOS version 3.3. The /F switch tells MS-DOS to format the backup diskette if it hasn't been formatted. The /F switch formats the backup diskette to the maximum capacity of the backup drive, so a disk of lower capacity, such as a 360 KB diskette in a 1.2M drive, should not be used. If this switch is used, FORMAT.COM must be available in the current drive and directory or in one of the directories named in the environment's PATH string.

Performance of the BACKUP command has also been improved. Instead of storing each file separately on the backup disk, BACKUP stores only two files: BACKUP.*nnn*, which contains all the backed-up files, and CONTROL.*nnn*, which contains the pathnames of the backed-up files.

FDISK

FDISK can now create a new type of MS-DOS partition called an extended partition on a fixed disk. An extended partition can contain multiple logical drives and allows the use of very large fixed disks. Each logical drive is still limited to 32 MB.

An extended partition is not bootable. In order for the fixed disk to be bootable, it must also contain a primary MS-DOS partition that has been formatted using the FORMAT command with the /S switch so that it contains a system boot record and the operating-system files.

MODE

The MODE command now supports two additional serial ports (COM3 and COM4) and increases the maximum serial transmission rate to 19,200 baud.

Some additional options have been added to MODE to support code-page switching. *See* MODE Command Changes below.

New national language support

The new national language support in MS-DOS version 3.3 replaces the methods used in previous versions to change the keyboard layout and the display and printer character sets so that more than one language could be used. These changes are extensive: four new or modified system files, three new commands, four new options for the MODE command, a new parameter for the GRAFTABL command, and a new parameter for the COUNTRY and DEVICE configuration commands.

Code pages and code-page switching

The key element of the new national language support is the code page, a table of 256 character correspondence codes. MS-DOS recognizes both a hardware code page, which is the character correspondence table built into a device, and a prepared code page, which is an alternate character correspondence table available through MS-DOS. The current code page is the code page most recently selected.

The hardware code page for a device is determined by the country for which the device was manufactured. The user selects a prepared code page, from a list of five included with MS-DOS version 3.3, by using the new CP PREPARE option of the MODE command. *See* MODE Command Changes below.

The new national language support is often referred to as code-page switching because, after the devices and code pages required by the system have been defined, the only commands the user must deal with simply switch from one code page to another. In order to use the new national language support, device drivers must support code-page switching and the devices must be able to display the full character sets.

Code pages are numbered. The identifying numbers have no relationship to the country code introduced with previous versions of MS-DOS and used by the COUNTRY configuration command. Five code pages are included with version 3.3:

Page Number	Configuration
437	United States
850	Multilingual
860	Portugal
863	Canadian French
865	Norway/Denmark

Code page 437 is the character correspondence table used in previous versions of MS-DOS. Its character set supports United States English and includes many accented characters used in other languages. It is the hardware code page for most countries.

Code page 850 replaces two of the four box-drawing sets and some of the mathematical symbols in code page 437 with additional accented characters. It supports English and most Latin-based European languages.

Code page 860 is for Portuguese, code page 863 is for Canadian French, and code page 865 is for Norwegian/Danish. These pages are the hardware code pages for the specified countries.

Setting up the system for code-page switching

Although several commands are required to manage national language support, the process is fairly straightforward. Setting up the system requires the following:

- A DEVICE configuration command in CONFIG.SYS to load a driver for each device that supports code-page switching.
- An NLSFUNC command in AUTOEXEC.BAT to load the memory-resident national language support functions.
- A MODE CP PREPARE command in AUTOEXEC.BAT to prepare code pages for each device that supports code-page switching.
- A CHCP command in AUTOEXEC.BAT to select the initial code page.
- Optionally, a KEYB command in AUTOEXEC.BAT to select the initial keyboard layout.

After starting the system with these commands in CONFIG.SYS and AUTOEXEC.BAT, only a MODE CP SELECT command is required to change to a different language during an MS-DOS session.

The COUNTRY configuration command is still used to control country-specific characteristics such as the time and date format and currency symbol. An added parameter in the COUNTRY command lets the user also specify a code page. *See* Modified National Language Support Commands below.

The system files

MS-DOS version 3.3 includes four system files that support the national language functions: two device drivers and two system information files.

The device drivers are PRINTER.SYS and DISPLAY.SYS. These drivers implement code-page switching for the IBM Proprinter Model 4201 and Quietwriter III Model 5202 printers and for the EGA, PC Convertible LCD, and PS/2 display adapters. They also support all display adapters compatible with the EGA.

The information files are COUNTRY.SYS, which contains information such as time and date formats and currency symbols, and KEYBOARD.SYS, which contains the scan-code-to-ASCII translation tables for the various keyboard layouts.

The new support commands

The new national language support in MS-DOS version 3.3 adds three MS-DOS commands: Change Code Page (CHCP), Keyboard (KEYB), and National Language Support Functions (NLSFUNC).

CHCP

The Change Code Page (CHCP) command tells MS-DOS which code page to use for all devices that support code-page switching.

The NLSFUNC command must be executed before the CHCP command can be used.

CHCP is a system-wide command: It specifies the code page used by MS-DOS and each device attached to the system that supports code-page switching. The CP SELECT option of the MODE command, on the other hand, specifies the code page for a single device.

If the code page specified with CHCP is not compatible with a device, CHCP responds

```
Code page nnn not prepared for all devices
```

If the code page specified with CHCP was not first identified with the CP PREPARE option of the MODE command, CHCP responds

```
Code page nnn not prepared for system
```

The CHCP command has one optional parameter:

CHCP [*code-page*]

The *code-page* parameter is the three-digit number that specifies the code page MS-DOS is to use. If *code-page* is omitted, CHCP displays the current MS-DOS code page.

Examples: The following command changes the system code page to 850:

```
C>CHCP 850 <Enter>
```

If the current code page is 850 and CHCP is entered without parameters, MS-DOS responds:

```
Active code page: 850
```

KEYB

The Keyboard (KEYB) command selects a keyboard layout by changing the scan-code-to-ASCII translation table used by the keyboard driver. It replaces the KEYBxx commands used in earlier versions of MS-DOS to select keyboard layouts.

The first time KEYB is executed, it loads the memory-resident keyboard driver and the translation table, thereby increasing the size of MS-DOS by slightly more than 7 KB. Subsequent executions simply load a different translation table, which replaces the previously loaded translation table and accommodates a different country-specific keyboard layout.

The KEYB command has three optional parameters:

KEYB [*country*],[*code-page*],[*kbdfile*]

The *country* parameter is one of the following two-character country codes:

Country	Code	Country	Code
Australia	US	Netherlands	NL
Belgium	BE	Norway	NO
Canada		Portugal	PO
English	US	Spain	SP
French	CF	Sweden	SV
Denmark	DK	Switzerland	
Finland	SU	French	SF
France	FR	German	SG
Germany	GR	United Kingdom	UK
Italy	IT	United States	US
Latin America	LA		

The *code-page* parameter is the three-digit number that specifies the code page defining the character set that MS-DOS is to use.

If the specified country code and code page aren't compatible, KEYB responds:

```
Code page requested nnn is not valid for given keyboard code
```

If KEYB is entered with no parameters, MS-DOS displays the currently active keyboard country code, keyboard code page, and console device code page.

Examples: The following command selects the French keyboard layout, code page 850, and the keyboard definition file named C:\DOS\KEYBOARD.SYS:

```
C>KEYB FR,850,C:\DOS\KEYBOARD.SYS <Enter>
```

If the code page is omitted but the keyboard definition file is specified, the comma must be included to show the missing parameter:

```
C>KEYB FR,,C:\DOS\KEYBOARD.SYS <Enter>
```

NLSFUNC

The National Language Support Function (NLSFUNC) command loads a memory-resident program that implements code-page switching. It also allows the user to name the file that contains country-specific information — such as date format, time format, and currency symbol — if there is no COUNTRY configuration command in CONFIG.SYS. NLSFUNC must be used before the Change Code Page (CHCP) command.

If national language support is needed for every session, NLSFUNC should be placed in the AUTOEXEC.BAT file.

The NLSFUNC command has one optional parameter:

```
NLSFUNC [country-file]
```


The *country-file* parameter is the name of the country information file (in most implementations of MS-DOS, COUNTRY.SYS). If *country-file* is omitted, MS-DOS defaults to the name of the country information file specified in the COUNTRY configuration command in CONFIG.SYS; if there is no COUNTRY configuration command in CONFIG.SYS, MS-DOS looks for a file named COUNTRY.SYS in the root directory of the current drive.

Example: The following command loads the NLSFUNC program and specifies C:\DOS\COUNTRY.SYS as the country information file:

```
C>NLSFUNC C:\DOS\COUNTRY.SYS <Enter>
```

The modified support commands

The new national language support changes two configuration commands — COUNTRY and DEVICE — and two general MS-DOS commands — GRAFTABL and MODE.

COUNTRY

The COUNTRY configuration command now has three parameters:

COUNTRY=*country-code*, [*code-page*], [*country-file*]

The *country-code* parameter is one of the following three-digit country codes (identical to the specified country's international telephone prefix):

Country	Code	Country	Code
Arabia	785	Latin America	003
Australia	061	Netherlands	031
Belgium	032	Norway	047
Canada		Portugal	351
English	001	Spain	034
French	002	Sweden	046
Denmark	045	Switzerland	
Finland	358	French	041
France	033	German	041
Germany	049	United Kingdom	044
Israel	972	United States	001
Italy	039		

The *code-page* parameter is the three-digit number that specifies the code page defining the character set that MS-DOS is to use.

The *country-file* parameter is the name of the file that contains the country-specific information; the name of the file can be preceded by a drive and/or path. If *country-file* is omitted, MS-DOS defaults to the file COUNTRY.SYS, which it looks for in the root directory of the current drive.

The COUNTRY command is not required; if it is not included in CONFIG.SYS, MS-DOS defaults to country 001 (US), code page 437, and country information file COUNTRY.SYS in the root directory of the current drive.

Example: The following CONFIG.SYS command specifies the French country code, code page 850, and C:\DOS\COUNTRY.SYS as the country information file:

```
COUNTRY=033,850,C:\DOS\COUNTRY.SYS
```

DEVICE

Two options have been added to the DEVICE configuration command that allow the user to specify the display and printer drivers that support code-page switching.

The display driver that supports code-page switching is DISPLAY.SYS. It supports the IBM Enhanced Graphics Adapter (EGA), the IBM Personal System/2 display adapter, and all display adapters compatible with either of these. The Monochrome Display Adapter (MDA) and the Color/Graphics Adapter (CGA) do not support code-page switching.

If the ANSI.SYS display driver is also used, the DEVICE command that defines it must precede the DEVICE command that defines DISPLAY.SYS.

When used to specify the display driver, the DEVICE command has five parameters:

```
DEVICE=driver CON=(type[,hwcp][,prepcp[,sub-fonts]])
```

The *driver* parameter is the name of the file that contains the display driver; the filename can be preceded by a drive and/or path. If *driver* is omitted, MS-DOS defaults to the file DISPLAY.SYS, which it looks for in the root directory of the current drive.

The *type* parameter defines the type of display adapter attached to the system. It must be one of the following:

Code	Adapter
MONO	Monochrome display/printer adapter
CGA	Color/graphics adapter
EGA	Enhanced graphics adapter or IBM Personal System/2 display adapter
LCD	IBM PC Convertible liquid crystal display

The *hwcp* parameter is the three-digit number that specifies the hardware code page supported by the display adapter:

Code	Configuration
437	United States (default)
850	Multilingual
860	Portugal
863	Canadian French
865	Norway/Denmark

The *prepcp* parameter is the number of additional code pages the display can support. These are referred to as prepared code pages and must be defined by the CP PREPARE option of the MODE command. If *type* is either MONO or CGA, *prepcp* must be 0; the default is 0. If *type* is either EGA or LCD, *prepcp* can be any value from 1 through 12; the default is 1. If *hwcp* is 437, *prepcp* should be allowed to default to 1; if *hwcp* is not 437, *prepcp* should be set to 2.

The *sub-fonts* parameter is the number of subfonts supported for each code page. If *type* is either MONO or CGA, *sub-fonts* must be 0; the default is 0. If *type* is EGA, *sub-fonts* can be 1 or 2; the default is 2. If *type* is LCD, *sub-fonts* can be 1 or 2; the default is 1.

Example: The following CONFIG.SYS command specifies C:\DOS\DISPLAY.SYS as the display driver for an EGA whose hardware code page is 437. The parameter for prepared code pages is allowed to default to 1 and the parameter for subfonts is allowed to default to 2.

```
DEVICE=C:\DOS\DISPLAY.SYS CON=(EGA,437)
```

The printer driver that supports code-page switching is PRINTER.SYS. It supports the IBM Proprinter Model 4201, the IBM Quietwriter III Printer Model 5202, and all printers compatible with either of these.

When used to specify the printer driver, the DEVICE configuration command has five parameters:

```
DEVICE=driver port=(type[,hwcp][,prepcp])
```

The *driver* parameter is the name of the file that contains the printer driver; the filename can be preceded by a drive and/or path. If *driver* is omitted, MS-DOS defaults to the file PRINTER.SYS, which it looks for in the root directory of the current drive.

The *port* parameter is the MS-DOS device name of the printer port being defined: LPT1 (or PRN), LPT2, or LPT3. A different set of *type*, *hwcp*, and *prepcp* parameters can be specified for each of the three printer ports.

The *type* parameter defines the type of printer attached to the printer port. It must be one of the following:

Code	Printer
4201	IBM Proprinter Model 4201
5202	IBM Quietwriter III Printer Model 5202

The *hwcp* parameter is a three-digit number that specifies the hardware code page supported by the hardware:

Code	Configuration
437	United States (default)
850	Multilingual
860	Portugal
863	Canadian French
865	Norway/Denmark

If *type* is 5202, two hardware code-page numbers can be specified, enclosed in parentheses and separated by a comma. If two hardware code pages are specified, *prepcp* must be 0.

The *prepcp* parameter is the number of additional code pages (referred to as prepared code pages) for which MS-DOS must reserve buffer space; its value can be from 0 through 12. These additional code pages must be defined by the CP PREPARE option of the MODE command. If *hwcp* is 437, *prepcp* should be set to 1; if *hwcp* is not 437 and only one *hwcp* value is specified, *prepcp* should be set to 2.

Examples: The following CONFIG.SYS command defines C:\DOS\PRINTER.SYS as the printer driver for the PRN device. The printer is an IBM Proprinter Model 4201 whose hardware code page is 437, and MS-DOS is instructed to allow for one prepared code page:

```
DEVICE=C:\DOS\PRINTER.SYS PRN=(4201,437,1)
```

The next CONFIG.SYS command defines C:\DOS\PRINTER.SYS as the printer driver for ports LPT1 and LPT2. The printer attached to LPT1 is the same as in the previous command; the printer attached to LPT2 is an IBM Quietwriter III Printer Model 5202 with two hardware code pages (437 and 850). For the second printer, MS-DOS is instructed to allow for no prepared code pages.

```
DEVICE=C:\DOS\PRINTER.SYS LPT1=(4201,437,1) LPT2=(5202,(437,850),0)
```

GRAFTABL

The GRAFTABL command now has two forms:

```
GRAFTABL [code-page]
```

or

```
GRAFTABL /STATUS
```

The first form of the command loads a code page for the color/graphics adapter (CGA) so that its character set matches that used by MS-DOS and other devices when displaying the upper 128 characters. The *code-page* parameter is the three-digit number that specifies the code page defining the character set that GRAFTABL is to use.

The /STATUS switch causes GRAFTABL to display the name of the graphics character set table currently in use.

MODE

National language support adds four options to the MODE command:

Option	Action
CODEPAGE	Displays the code pages available and active.
CODEPAGE PREPARE	Defines the code pages selected for use.
CODEPAGE REFRESH	Restores code-page contents damaged by hardware error or other causes.
CODEPAGE SELECT	Selects a code page for a particular device.

(CODEPAGE can be abbreviated to CP in the command line.)

When used to display the status of the code pages, the MODE command has one parameter:

MODE *device* CP

The *device* parameter is the name of the device whose code-page status is to be displayed. It can be CON, PRN, LPT1, LPT2, or LPT3.

Example: The following command displays the status of the console device:

```
C>MODE CON CP <Enter>
```

When used to define the code page or pages to be used with a device, the MODE command has three parameters:

MODE *device* CP PREPARE=(*code-page font-file*)

The *device* parameter is the name of the device for which the code page or pages are to be prepared. It can be CON, PRN, LPT1, LPT2, or LPT3.

The *code-page* parameter is one or more of the three-digit numbers, enclosed in parentheses, that specify the code page to be used with *device*. If more than one code-page number is specified, the numbers must be separated with spaces.

The *font-file* parameter is the name of the code-page file that contains the font information for *device*. The files provided for IBM devices include

File	Device
EGA.CPI	IBM Enhanced Graphics Adapter (EGA) and EGA-compatible display adapters
4201.CPI	IBM Proprinter Model 4201
5202.CPI	IBM Quietwriter III Printer Model 5202
LCD.CPI	IBM Convertible liquid crystal display

Example: Assume the display is attached to an EGA. The following command prepares code pages 437 and 850 for the console, specifying C:\DOS\EGA.CPI as the code-page information file:

```
C>MODE CON CP PREPARE=((437 850) C:\DOS\EGA.CPI) <Enter>
```

When used to select a code page for a device, the MODE command has two parameters:

```
MODE device CP SELECT=code-page
```

The *device* parameter is the name of the device for which the code page is to be selected. Permissible values are CON, PRN, LPT1, LPT2, and LPT3.

The *code-page* parameter is the three-digit number that specifies the code page to be used with *device*.

Example: The following command selects code page 850 for the console:

```
C>MODE CON CP SELECT=850 <Enter>
```

Setting up code-page switching for an EGA-only system

Figure A-1 shows the commands required to implement the new national language support for a system that includes only a display attached to an EGA or EGA-compatible adapter. The hardware code page of the EGA is 437 (United States English) and the system is set up to handle code pages 437 and 850. All MS-DOS files are assumed to be in the directory \DOS on the disk in drive C. If the ANSI.SYS driver is not used, the configuration command DEVICE=C:\DOS\ANSI.SYS should be omitted from CONFIG.SYS; if ANSI.SYS is used, however, the DEVICE configuration command that defines it must precede the DEVICE configuration command that defines DISPLAY.SYS.

Commands in CONFIG.SYS:

```
COUNTRY=001,437,C:\DOS\COUNTRY.SYS
DEVICE=C:\DOS\ANSI.SYS
DEVICE=C:\DISPLAY.SYS CON=(EGA,437,1)
```

Commands in AUTOEXEC.BAT:

```
NLSFUNC C:\DOS\COUNTRY.SYS
MODE CON CP PREPARE=((437 850) C:\DOS\EGA.CPI)
MODE CON CP SELECT=437
KEYB US,437,C:\DOS\KEYBOARD.SYS
```

Figure A-1. Setup commands for a system with an EGA only.

When the system is started, code page 437 is selected for MS-DOS, the display, and the keyboard. To change to code page 850 during the session, simply type

```
C>CHCP 850 <Enter>
```

Setting up code-page switching for a PS/2 and printer

Figure A-2 shows the commands required to implement the new national language support for an IBM Personal System/2 or compatible system that includes both a PS/2, EGA, or EGA-compatible display adapter and an IBM Proprinter Model 4201. The hardware code page of both devices is 437 (United States English) and the system is set up to handle code pages 437 and 850.

Commands in CONFIG.SYS:

```
COUNTRY=001,437,C:\DOS\COUNTRY.SYS
DEVICE=C:\DOS\ANSI.SYS
DEVICE=C:\DISPLAY.SYS CON=(EGA,437,1)
DEVICE=C:\DOS\PRINTER.SYS PRN=(4201,437,1)
```

Commands in AUTOEXEC.BAT:

```
NLSFUNC C:\DOS\COUNTRY.SYS
MODE CON CP PREPARE=((437 850) C:\DOS\EGA.CPI)
MODE PRN CP PREPARE=((437 850) C:\DOS\4202.CPI)
MODE CON CP SELECT=850
MODE PRN CP SELECT=850
KEYB US,850,C:\DOS\KEYBOARD.SYS
```

Figure A-2. Setup commands for a PS/2 with display and printer.

Again, all MS-DOS files are assumed to be in the directory \DOS on the disk in drive C. If the ANSI.SYS driver is not used, the configuration command DEVICE=C:\DOS\ANSI.SYS should be omitted from CONFIG.SYS; if ANSI.SYS is used, however, the DEVICE configuration command that defines it must precede the DEVICE configuration command that defines DISPLAY.SYS.

Version 3.3 Programming Considerations

The changes introduced in MS-DOS version 3.3 that are of primary interest to the programmer include

- New Interrupt 21H function calls for file management and internationalization support
- An extension to the definition of the MS-DOS IOCTL function for code-page switching, plus the addition of the underlying device-driver support
- Support for extended MS-DOS partitions on fixed disks

Each of these areas is discussed in detail below.

New file-management functions

MS-DOS version 3.3 includes two new Interrupt 21H file-management functions: Set Handle Count (Function 67H) and Commit File (Function 68H).

Set Handle Count

The Set Handle Count function (Interrupt 21H Function 67H) allows a single process to have more than 20 handles for files or devices open simultaneously. Function 67H is invoked by issuing a software Interrupt 21H with

AH = 67H

BX = number of desired handles

On return,

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code

For each process, the operating system maintains a table that relates handle numbers for the process to MS-DOS's internal global table for all open files in the system. In MS-DOS versions 3.0 and later, the per-process table is ordinarily stored within the reserved area of the program segment prefix (PSP) and has only enough room for 20 handle entries. If 20 or fewer handles are requested in register BX, Function 67H takes no action and returns a success signal. If more than 20 handles are requested, however, Function 67H allocates on behalf of the calling program a new block of memory that is large enough to hold the expanded table of handle numbers and then copies the process's old handle table to the new table. Because the function will fail if the system does not have sufficient free memory to allocate the new block, most programs need to make a call to Interrupt 21H Function 4AH (Resize Memory Block) to "shrink" their initial memory block allocations before calling Function 67H.

Function 67H does not fail if the number requested is larger than the available entries in the system's global table for file and device handles. However, a subsequent attempt to open a file or device or to create a new file will fail if all the entries in the system's global file table are in use, even if the requesting process has not used up all its own handles. (The size of the global table is controlled by the FILES entry in the CONFIG.SYS file. See USER COMMANDS: CONFIG.SYS: FILES; PROGRAMMING IN THE MS-DOS ENVIRONMENT: PROGRAMMING FOR MS-DOS: File and Record Management.)

Example: Set the maximum handle count for the current process to 30, so that the process can have as many as 25 files or devices open simultaneously (5 of the handles are already expended by the MS-DOS standard devices when the process starts up). Note that a FILES=30 (or greater value) entry in the CONFIG.SYS file also is required for the process to successfully open 30 files or devices.


```
.  
. .  
mov    ah,67h      ; Function 67H = set handle count.  
mov    bx,30       ; Maximum number of handles.  
int    21h         ; Transfer to MS-DOS.  
jc     error       ; Jump if function failed.  
. . .
```

Commit File

The Commit File function (Interrupt 21H Function 68H) forces all data in MS-DOS's internal buffers that is associated with a given handle to be written to disk and forces the corresponding disk directory and file allocation table (FAT) information to be updated. By calling this function at appropriate points within its execution, a program can ensure that newly entered data will not be lost if there is a power failure, if the program crashes, or if the user fails to terminate the program properly before turning off the machine. Function 68H is called by issuing a software Interrupt 21H with

AH = 68H

BX = handle for previously opened file.

On return,

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code

The effect of Function 68H is equivalent to closing and reopening the file or to duplicating a file handle with Interrupt 21H Function 45H (Duplicate File Handle) and then closing the duplicate. See PROGRAMMING IN THE MS-DOS ENVIRONMENT: PROGRAMMING FOR MS-DOS: File and Record Management. However, Function 68H has the advantages that the application will not lose control of the file (as could happen with the close-open sequence in a networking environment) and that it will not fail because of a lack of handles (as the duplicate handle method might).

Note: Function 68H operations requested on a handle associated with a character device return a success flag but have no effect.

Example: Assume that the file MYFILE.DAT has been opened previously and that the handle for the file is stored in the variable *handle*. Call Function 68H to ensure that any data in MS-DOS's internal buffers associated with the handle is written out to disk and that the directory and FAT are up-to-date.

```

fname db 'MYFILE.DAT',0 ; ASCIIZ filename.
fhandle dw ? ; Handle from Open operation.
.
.
mov ah,68h ; Function 68H = commit file.
mov bx,fhandle ; Handle from previous open.
int 21h ; Transfer to MS-DOS.
jc error ; Jump if function failed.
.
.

```

New internationalization support functions

MS-DOS version 3.3 includes two new Interrupt 21H internationalization support functions: Get Extended Country Information (Function 65H) and Select Code Page (Function 66H).

Get Extended Country Information

The Get Extended Country Information function (Interrupt 21H Function 65H) returns a superset of the internationalization information obtained with Interrupt 21H Function 38H (Get/Set Current Country). Function 65H is called by issuing a software Interrupt 21H with

```

AH      = 65H
AL      = information ID code:
          01H    get general internationalization information
          02H    get pointer to uppercase table
          04H    get pointer to filename uppercase table
          06H    get pointer to collating sequence table
BX      = code page of interest (active CON device = -1)
CX      = length of buffer to receive information (error returned if less than 5)
DX      = country ID (default = -1)
ES:DI   = address of buffer to receive information

```

On return,

If function is successful:

Carry flag is clear.

Requested data is in calling program's buffer.

If function is not successful:

Carry flag is set.

```
AX      = error code
```

Function 65H may fail if either the country code or the code-page number is invalid or if the code page does not match the country code. If the buffer to receive the information is at least 5 bytes but is too short for the requested information, the data is truncated and no error is returned.

The format of the data returned by Subfunction 01H in the calling program's buffer is

Field	Size
Information ID code (01H)	Byte
Length of following buffer (38 or less)	Word
Country ID	Word
Code-page number	Word
Date format	Word
Currency symbol	5 bytes
Thousands separator	Word
Decimal separator	Word
Date separator	Word
Time separator	Word
Currency format flags	Byte
Digits in currency	Byte
Time format	Byte
Monocase routine entry point	Doubleword
Data list separator	Word
Reserved	10 bytes

See SYSTEM CALLS: INTERRUPT 21H: Function 38H.

The format of the data returned by Subfunctions 02H, 04H, and 06H is

Field	Size
Information ID code (02H, 04H, or 06H)	Byte
Pointer to table	Doubleword

The uppercase and filename uppercase tables are 130 bytes. The first 2 bytes contain the size of the table; the subsequent 128 bytes contain the uppercase equivalents, if any, for character codes 80H through 0FFH. The main use of these tables is to map accented or otherwise modified vowels to their plain vowel equivalents. Text translated using these tables can be sent to devices that do not support the IBM graphics character set or can be used to create filenames that do not require a special keyboard configuration for entry.

The collating table is 258 bytes. The first 2 bytes contain the table length and the next 256 bytes contain the values to be used for the corresponding character codes (0-0FFH) during a sort operation. Among other things, this table maps uppercase and lowercase ASCII characters to the same collating codes (so that sorts will be case insensitive) and maps accented vowels to their plain vowel equivalents.

Note: In some cases, a truncated translation table might be presented to the program by MS-DOS. Applications should always check the length specified at the beginning of the table to be sure the table contains a translation code for the character of interest.

Example: Obtain the extended country information associated with the default country and code page 437.

```
buffer db      41 dup (0)      ; Receives country information.
      .
      .
      .
      mov     ax,6501h         ; Function = get extended info.
      mov     bx,437          ; Code page.
      mov     cx,41           ; Length of buffer.
      mov     dx,-1           ; Default country.
      mov     di,seg buffer    ; ES:DI = buffer address.
      mov     es,di
      mov     di,offset buffer
      int     21h             ; Transfer to MS-DOS.
      jc     error            ; Jump if function failed.
      .
      .
      .
```

In this case, MS-DOS fills the following extended country information into the buffer:

```
buffer db      1              ; Information ID code
      dw      38              ; Length of following buffer
      dw      1              ; Country ID (USA)
      dw      437            ; Code-page number
      dw      0              ; Date format
      db      '$',0,0,0,0    ; Currency symbol
      db      ',',0         ; Thousands separator
      db      '.',0         ; Decimal separator
      db      '-',0         ; Date separator
      db      ':',0         ; Time separator
      db      0              ; Currency format flags
      db      2              ; Digits in currency
      db      0              ; Time format
      dd      026ah:176ch    ; Monocase routine entry point
      db      ',',0         ; Data list separator
      db      10 dup (0)     ; Reserved
```

Example: Obtain the pointer to the uppercase table associated with the default country and code page 437.

```
buffer db      5 dup (0)      ; Receives pointer information.
      .
      .
      .
      mov     ax,6502h         ; Function = get pointer to
      ; uppercase table.
```

(more)

```

mov    bx,437          ; Code page.
mov    cx,5           ; Length of buffer.
mov    dx,-1          ; Default country.
mov    di,seg buffer  ; ES:DI = buffer address.
mov    es,di
mov    di,offset buffer
int    21h            ; Transfer to MS-DOS.
jc     error          ; Jump if function failed.
.
.
.

```

In this case, MS-DOS fills the following values into the buffer:

```

buffer db    2          ; Information ID code
        dw    0204h     ; Offset of uppercase table
        dw    1140h     ; Segment of uppercase table

```

The table at 1140:0204H contains the following data:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
1140:0200					80	00	80	9A	45	41	8E	41	8F	80	45	45EA.A..EE
1140:0210	45	49	49	49	8E	8F	90	92	92	4F	99	4F	55	55	59	99	EIII.....O.OUUY.
1140:0220	9A	9B	9C	9D	9E	9F	41	49	4F	55	A5	A5	A6	A7	A8	A9AIIOU.....
1140:0230	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9
1140:0240	BA	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
1140:0250	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
1140:0260	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9
1140:0270	EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
1140:0280	FA	FB	FC	FD	FE	FF										

Select Code Page

The Select Code Page function (Interrupt 21H Function 66H) queries or selects the current code page. Function 66H is called by issuing a software Interrupt 21H with

AH = 66H

AL = subfunction:

01H get code page

02H select code page

BX = code page to select if AL = 02H

On return,

If function is successful:

Carry flag is clear.

If AL was 01H on call:

BX = active code page

DX = default code page

If function is not successful:

Carry flag is set.

AX = error code

When Subfunction 02H is used, MS-DOS gets the new code page from the COUNTRY.SYS file. The device must be previously prepared for code-page switching by including the appropriate DEVICE command in the CONFIG.SYS file and by issuing the NLSFUNC and MODE CP PREPARE commands (usually by placing them in the AUTOEXEC.BAT file).

Example: Force the active code page to be the same as the system's default code page—that is, return to the code page that was active when the system was first booted.

```

.
.
.
mov     ax,6601h       ; Function = get code page.
int     21h           ; Transfer to MS-DOS.
jc      error         ; Jump if function failed.

mov     bx,dx         ; Force active page = default.

mov     ax,6602h       ; Function = set code page.
int     21h           ; Transfer to MS-DOS.
jc      error         ; Jump if function failed.
.
.
.

```

Extension of IOCTL

The MS-DOS IOCTL service (Interrupt 21H Function 44H) and its device-driver underpinnings have been extended to support code-page switching by the interactive CHCP and MODE commands or by application programs. The relevant IOCTL subfunction is 0CH (Generic IOCTL for Handles). An MS-DOS utility or application program gains access to this subfunction by executing a software Interrupt 21H with

```

AH      = 44H
AL      = 0CH
BX      = handle for character device
CH      = category code:
          00H    unknown
          01H    COM1, COM2, COM3, or COM4
          03H    CON (keyboard and video display)
          05H    LPT1, LPT2, or LPT3

```

(more)

CL = function (minor) code:
 4AH select code page
 4CH start code-page preparation
 4DH end code-page preparation
 6AH query selected code page
 6BH query prepare list
 DS:DX = pointer to Generic IOCTL parameter block

On return,

If function is successful:

Carry flag is clear.

If function is not successful:

Carry flag is set.

AX = error code:
 01H invalid function number
 19H bad data read from font file
 22H unknown command
 26H code page not prepared or selected
 27H code page conflict or device or code page not found in file
 29H device error
 31H file contents not a valid font or no previous "start code-page preparation" call

Additional information about the cause of the error can be obtained with a call to Interrupt 21H Function 59H (Get Extended Error Information).

The parameter blocks for minor codes 4AH, 4DH, and 6AH have the following format:

Field	Size
Length of following data	Word
Code page ID	Word

The parameter block for minor code 4CH has the following format:

Field	Size
Flags	Word
Length of remainder of parameter block (2[n+1])	Word
Number of code pages in the following list (n)	Word

(more)

Field	Size
Code page 1	Word
Code page 2	Word
.	.
Code page n	Word

The parameter block for minor code 6BH has the following format, assuming n hardware code pages and m prepared code pages ($n \leq 12$, $m \leq 12$):

Field	Size
Length of following data ($2[n+m+2]$)	Word
Number of hardware code pages (n)	Word
Hardware code page 1	Word
Hardware code page 2	Word
.	.
Hardware code page n	Word
Number of prepared code pages (m)	Word
Prepared code page 1	Word
Prepared code page 2	Word
.	.
Prepared code page m	Word

After a Start Code-Page Preparation (minor code 4CH) call, the program must write the data defining the code-page font to the driver using one or more IOCTL Send Control Data to Character Device (Interrupt 21H Function 44H Subfunction 03H) calls. The format of the data is both device-specific and driver-specific. After the font data has been written to the driver, the program must issue an End Code-Page Preparation (minor code 4DH) call. If no data is written to the driver between the start and end calls, the driver interprets the newly prepared code pages as hardware code pages.

A special variation of Start Code-Page Preparation, called "refresh," is required to actually load the peripheral device with the prepared code pages. The refresh operation is obtained by calling minor code 4CH with each code-page position in the parameter block set to -1 and then immediately calling minor code 4DH.

The device-driver support that corresponds to IOCTL Subfunction 0CH is invoked by the MS-DOS kernel via the Generic IOCTL function (driver command code 19). The category (major) and function (minor) codes described above, along with a pointer to the parameter block, are passed to the driver in the request header. *See* PROGRAMMING IN THE MS-DOS ENVIRONMENT: CUSTOMIZING MS-DOS: Installable Device Drivers.

Extended MS-DOS partitions

An extended MS-DOS partition is indicated by a system indicator byte value of 05 in the partition table of the fixed disk's master boot record. *See* PROGRAMMING IN THE MS-DOS ENVIRONMENT: STRUCTURE OF MS-DOS: MS-DOS Storage Devices. An extended partition is not bootable and can be created on a bootable fixed-disk drive only if that drive already contains a primary MS-DOS partition (system indicator type 01 or 04). Fixed disks that are not bootable can contain an extended partition without a primary partition.

An extended partition is subdivided into extended logical disk volumes, each consisting of an extended boot record and a logical block device. The extended boot record is analogous in structure to the partition table for the fixed disk as a whole; it contains a logical drive table describing the volume and a pointer to the next extended logical volume. The logical block device is an image of a normal MS-DOS disk, including a master block (logical sector 0 containing the BPB describing the device), root directory, FAT, and files area. Each extended volume must start and end on a cylinder boundary.

*Van Wolverton
Ray Duncan*

Appendix B

Critical Error Codes

Critical errors are returned via Interrupt 24H. If register AL bit 7 is 0, then the error was a disk error; if register AL bit 7 is 1, then the error was a nondisk error. The upper half of DI is undefined; the lower half of DI contains one of the following error-condition codes:

Code	Description
00H	Attempt to write on write-protected disk
01H	Unknown drive or unit
02H	Drive not ready
03H	Invalid command
04H	Data error (CRC failed)
05H	Bad request structure length
06H	Seek error
07H	Unknown media type
08H	Sector not found
09H	Printer out of paper
0AH	Write fault
0BH	Read fault
0CH	General failure
0FH	Invalid disk change

Appendix C

Extended Error Codes

The extended error codes used by Interrupt 21H functions consist of four separate codes in the AX, BH, BL, and CH registers. These codes give as much detail as possible about the error and suggest how the issuing program should respond.

AX—Extended Error Code

If an error condition occurs in response to an Interrupt 21H function call, the carry flag is set and one of the following error codes is returned in AX:

Error	Description	Error	Description
01H	Invalid function code	16H	Invalid disk command
02H	File not found	17H	CRC error
03H	Path not found	18H	Invalid length (disk operation)
04H	Too many open files (no handles left)	19H	Seek error
05H	Access denied	1AH	Not an MS-DOS disk
06H	Invalid handle	1BH	Sector not found
07H	Memory control blocks destroyed	1CH	Out of paper
08H	Insufficient memory	1DH	Write fault
09H	Invalid memory block address	1EH	Read fault
0AH	Invalid environment	1FH	General failure
0BH	Invalid format	20H	Sharing violation
0CH	Invalid access code	21H	Lock violation
0DH	Invalid data	22H	Wrong disk
0EH	Reserved	23H	FCB unavailable
0FH	Invalid drive	24H	Sharing buffer overflow
10H	Attempt to remove the current directory	25–31H	Reserved
11H	Not same device	32H	Network request not supported
12H	No more files	33H	Remote computer not listening
13H	Disk is write-protected	34H	Duplicate name on network
14H	Bad disk unit	35H	Network path not found
15H	Drive not ready	36H	Network busy
		37H	Network device no longer exists
		38H	Net BIOS command limit exceeded

(more)

Error	Description	Error	Description
39H	Network adapter hardware error	45H	Net BIOS session limit exceeded
3AH	Incorrect response from network	46H	Sharing temporarily paused
3BH	Unexpected network error	47H	Network request not accepted
3CH	Incompatible remote adapter	48H	Print or disk redirection paused
3DH	Print queue full	49-4FH	Reserved
3EH	Print queue not full	50H	File exists
3FH	Print file was canceled (not enough space)	51H	Reserved
40H	Network name was deleted	52H	Cannot make directory entry
41H	Access denied	53H	Fail on Interrupt 24H
42H	Network device type incorrect	54H	Out of network structures
43H	Network name not found	55H	Device already assigned
44H	Network name limit exceeded	56H	Invalid password
		57H	Invalid parameter
		58H	Network data fault

BH — Error Class

BH returns a code that describes the class of error that occurred:

Class	Description
01H	Out of a resource, such as storage or channels
02H	Not an error, but a temporary situation (such as a locked region in a file) that can be expected to end
03H	Authorization problem
04H	An internal error in system software
05H	Hardware failure
06H	A system software failure not the fault of the active process (could be caused by missing or incorrect configuration files, for example)
07H	Application program error
08H	File or item not found
09H	File or item of invalid format or type or otherwise invalid or unsuitable
0AH	File or item interlocked
0BH	Wrong disk in drive, bad spot on disk, or other problem with storage medium
0CH	Other error

BL — Suggested Action

BL returns a code that suggests how the program should respond to the error:

Action	Description
01H	Retry, then prompt user.
02H	Retry after a pause.
03H	If the user entered data such as a drive letter or filename, prompt for it again.
04H	Terminate with cleanup.
05H	Terminate immediately. The system is so unhealthy that the program should exit as soon as possible without taking the time to close files and update indexes.
06H	Error is informational.
07H	Prompt the user to perform some action, such as changing disks, then retry the operation.

CH — Locus

CH returns a code that provides additional information to help locate the area involved in the failure. This code is particularly useful for hardware failures (BH = 05H).

Locus	Description
01H	Unknown
02H	Related to random-access block devices, such as a disk drive
03H	Related to network
04H	Related to serial-access character devices, such as a printer
05H	Related to random-access memory

Procedure

Programs should handle errors by noting the error returned in AX from the original system call and then invoking Interrupt 21H Function 59H to get the extended error information. If no extended error information is provided, the program should respond to the original error code.

The Function 59H system call is available during Interrupt 24H.

Appendix D

ASCII Character Set and IBM Extended Character Set

Char	Number		Control	Char	Number		Control
	Dec	Hex			Dec	Hex	
	0	00	NUL (Null)	#	35	23	
☉	1	01	SOH (Start of heading)	\$	36	24	
☺	2	02	STX (Start of text)	%	37	25	
♥	3	03	ETX (End of text)	&	38	26	
♦	4	04	EOT (End of transmission)	'	39	27	
♣	5	05	ENQ (Enquiry)	(40	28	
♠	6	06	ACK (Acknowledge))	41	29	
•	7	07	BEL (Bell)	*	42	2A	
■	8	08	BS (Backspace)	+	43	2B	
○	9	09	HT (Horizontal tab)	,	44	2C	
◐	10	0A	LF (Linefeed)	-	45	2D	
♂	11	0B	VT (Vertical tab)	.	46	2E	
♀	12	0C	FF (Formfeed)	/	47	2F	
♪	13	0D	CR (Carriage return)	0	48	30	
♪	14	0E	SO (Shift out)	1	49	31	
*	15	0F	SI (Shift in)	2	50	32	
▶	16	10	DLE (Data link escape)	3	51	33	
◀	17	11	DC1 (Device control 1)	4	52	34	
‡	18	12	DC2 (Device control 2)	5	53	35	
‡	19	13	DC3 (Device control 3)	6	54	36	
‡	20	14	DC4 (Device control 4)	7	55	37	
§	21	15	NAK (Negative acknowledge)	8	56	38	
-	22	16	SYN (Synchronous idle)	9	57	39	
‡	23	17	ETB (End transmission block)	:	58	3A	
↑	24	18	CAN (Cancel)	;	59	3B	
↓	25	19	EM (End of medium)	<	60	3C	
→	26	1A	SUB (Substitute)	=	61	3D	
←	27	1B	ESC (Escape)	>	62	3E	
┌	28	1C	FS (File separator)	?	63	3F	
↔	29	1D	GS (Group separator)	@	64	40	
▲	30	1E	RS (Record separator)	A	65	41	
▼	31	1F	US (Unit separator)	B	66	42	
<space>	32	20		C	67	43	
!	33	21		D	68	44	
"	34	22		E	69	45	
				F	70	46	
				G	71	47	
				H	72	48	

(more)

Appendix D: ASCII Character Set and IBM Extended Character Set

Char	Number		Char	Number		Control	Char	Number	
	Dec	Hex		Dec	Hex			Dec	Hex
I	73	49	z	122	7A			171	AB
J	74	4A	{	123	7B			172	AC
K	75	4B		124	7C			173	AD
L	76	4C	}	125	7D			174	AE
M	77	4D	~	126	7E			175	AF
N	78	4E		127	7F	DEL		176	B0
O	79	4F		128	80			177	B1
P	80	50		129	81			178	B2
Q	81	51		130	82			179	B3
R	82	52		131	83			180	B4
S	83	53		132	84			181	B5
T	84	54		133	85			182	B6
U	85	55		134	86			183	B7
V	86	56		135	87			184	B8
W	87	57		136	88			185	B9
X	88	58		137	89			186	BA
Y	89	59		138	8A			187	BB
Z	90	5A		139	8B		¡	188	BC
[91	5B		140	8C		¢	189	BD
\	92	5C		141	8D		£	190	BE
]	93	5D		142	8E		¤	191	BF
^	94	5E		143	8F		¥	192	C0
_	95	5F		144	90		¦	193	C1
`	96	60		145	91		§	194	C2
a	97	61		146	92		¨	195	C3
b	98	62		147	93		©	196	C4
c	99	63		148	94		ª	197	C5
d	100	64		149	95		«	198	C6
e	101	65		150	96		¬	199	C7
f	102	66		151	97		­	200	C8
g	103	67		151	98		®	201	C9
h	104	68		152	99		¯	202	CA
i	105	69		154	9A		°	203	CB
j	106	6A		155	9B		±	204	CC
k	107	6B		156	9C		²	205	CD
l	108	6C		157	9D		³	206	CE
m	109	6D		158	9E		´	207	CF
n	110	6E		159	9F		µ	208	D0
o	111	6F		160	A0		¶	209	D1
p	112	70		161	A1		·	210	D2
q	113	71		162	A2		¸	211	D3
r	114	72		163	A3			212	D4
s	115	73		164	A4			213	D5
t	116	74		165	A5			214	D6
u	117	75		166	A6			215	D7
v	118	76		167	A7			216	D8
w	119	77		168	A8			217	D9
x	120	78		169	A9			218	DA
y	121	79		170	AA			219	DB

(more)

Char	Number		Char	Number		Char	Number	
	Dec	Hex		Dec	Hex		Dec	Hex
■	220	DC	ϕ	232	E8	∫	244	F4
▮	221	DD	⊖	233	E9	∫	245	F5
▮	222	DE	Ω	234	EA	+	246	F6
■	223	DF	δ	235	EB	≈	247	F7
α	224	E0	∞	236	EC	•	248	F8
β	225	E1	φ	237	ED	•	249	F9
Γ	226	E2	ε	238	EE	•	250	FA
π	227	E3	∩	239	EF	√	251	FB
Σ	228	E4	≡	240	F0	η	252	FC
σ	229	E5	±	241	F1	∴	253	FD
μ	230	E6	≥	242	F2	•	254	FE
τ	231	E7	≤	243	F3	•	255	FF

Appendix E

EBCDIC Character Set

Number			Number			Number		
Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
NUL	0	00		41	29		82	52
SOH	1	01	SM	42	2A		83	53
STX	2	02	CU2	43	2B		84	54
ETX	3	03		44	2C		85	55
PF	4	04	ENQ	45	2D		86	56
HT	5	05	ACK	46	2E		87	57
LC	6	06	BEL	47	2F		88	58
DEL	7	07		48	30		89	59
GE	8	08		49	31	!	90	5A
RLF	9	09	SYN	50	32	\$	91	5B
SMM	10	0A		51	33	*	92	5C
VT	11	0B	PN	52	34)	93	5D
FF	12	0C	RS	53	35	;	94	5E
CR	13	0D	UC	54	36	┌	95	5F
SO	14	0E	EOT	55	37	-	96	60
SI	15	0F		56	38	/	97	61
DLE	16	10		57	39		98	62
DC1	17	11		58	3A		99	63
DC2	18	12	CU3	59	3B		100	64
TM	19	13	DC4	60	3C		101	65
RES	20	14	NAK	61	3D		102	66
NL	21	15		62	3E		103	67
BS	22	16	SUB	63	3F		104	68
IL	23	17	Sp	64	40		105	69
CAN	24	18		65	41	!	106	6A
EM	25	19		66	42	,	107	6B
CC	26	1A		67	43	%	108	6C
CU1	27	1B		68	44	┌	109	6D
IFS	28	1C		69	45	>	110	6E
IGS	29	1D		70	46	?	111	6F
IRS	30	1E		71	47		112	70
IUS	31	1F		72	48		113	71
DS	32	20		73	49		114	72
SOS	33	21	¢	74	4A		115	73
FS	34	22	.	75	4B		116	74
	35	23	<	76	4C		117	75
BYP	36	24	(77	4D		118	76
LF	37	25	+	78	4E		119	77
ETB	38	26		79	4F		120	78
ESC	39	27	&	80	50		121	79
	40	28		81	51		122	7A

Appendix E: EBCDIC Character Set

Number			Number			Number		
Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
#	123	7B	y	168	A8	N	213	D5
@	124	7C	z	169	A9	O	214	D6
'	125	7D		170	AA	P	215	D7
=	126	7E		171	AB	Q	216	D8
"	127	7F		172	AC	R	217	D9
	128	80		173	AD		218	DA
a	129	81		174	AE		219	DB
b	130	82		175	AF		220	DC
c	131	83		176	B0		221	DD
d	132	84		177	B1		222	DE
e	133	85		178	B2		223	DF
f	134	86		179	B3	\	224	E0
g	135	87		180	B4		225	E1
h	136	88		181	B5	S	226	E2
i	137	89		182	B6	T	227	E3
	138	8A		183	B7	U	228	E4
	139	8B		184	B8	V	229	E5
	140	8C		185	B9	W	230	E6
	141	8D		186	BA	X	231	E7
	142	8E		187	BB	Y	232	E8
	143	8F		188	BC	Z	233	E9
	144	90		189	BD		234	EA
j	145	91		190	BE		235	EB
k	146	92		191	BF	d	236	EC
l	147	93	{	192	C0		237	ED
m	148	94	A	193	C1		238	EE
n	149	95	B	194	C2		239	EF
o	150	96	C	195	C3	0	240	F0
p	151	97	D	196	C4	1	241	F1
q	152	98	E	197	C5	2	242	F2
r	153	99	F	198	C6	3	243	F3
	154	9A	G	199	C7	4	244	F4
	155	9B	H	200	C8	5	245	F5
	156	9C	I	201	C9	6	246	F6
	157	9D		202	CA	7	247	F7
	158	9E		203	CB	8	248	F8
	159	9F	J	204	CC	9	249	F9
	160	A0		205	CD		250	FA
~	161	A1	Y	206	CE		251	FB
s	162	A2		207	CF		252	FC
t	163	A3	}	208	D0		253	FD
u	164	A4	J	209	D1		254	FE
v	165	A5	K	210	D2	EO	255	FF
w	166	A6	L	211	D3			
x	167	A7	M	212	D4			

Appendix F

ANSI.SYS Key and Extended Key Codes

The following escape sequence allows redefinition of keyboard keys to a specified *string*:

ESC[*code*;*string*;*...p*

where:

string is either the ASCII code for a single character or a string contained in quotation marks. For example, both 65 and "A" can be used to represent an uppercase A.

code is one or more of the following values that represent keyboard keys. Semi-colons shown in this table must be entered in addition to the required semi-colons in the command line.

Key	Code			
	Alone	Shift-	Ctrl-	Alt-
F1	0;59	0;84	0;94	0;104
F2	0;60	0;85	0;95	0;105
F3	0;61	0;86	0;96	0;106
F4	0;62	0;87	0;97	0;107
F5	0;63	0;88	0;98	0;108
F6	0;64	0;89	0;99	0;109
F7	0;65	0;90	0;100	0;110
F8	0;66	0;91	0;101	0;111
F9	0;67	0;92	0;102	0;112
F10	0;68	0;93	0;103	0;113
Home	0;71	55	0;119	—
Up Arrow	0;72	56	—	—
Pg Up	0;73	57	0;132	—
Left Arrow	0;75	52	0;115	—
Down Arrow	0;77	54	0;116	—
End	0;79	49	0;117	—
Down Arrow	0;80	50	—	—
Pg Dn	0;81	51	0;118	—
Ins	0;82	48	—	—
Del	0;83	46	—	—
PrtSc	—	—	0;114	—
A	97	65	1	0;30

(more)

Key	Code			
	Alone	Shift-	Ctrl-	Alt-
B	98	66	2	0;48
C	99	67	3	0;46
D	100	68	4	0;32
E	101	69	5	0;18
F	102	70	6	0;33
G	103	71	7	0;34
H	104	72	8	0;35
I	105	73	9	0;23
J	106	74	10	0;36
K	107	75	11	0;37
L	108	76	12	0;38
M	109	77	13	0;50
N	110	78	14	0;49
O	111	79	15	0;24
P	112	80	16	0;25
Q	113	81	17	0;16
R	114	82	18	0;19
S	115	83	19	0;31
T	116	84	20	0;20
U	117	85	21	0;22
V	118	86	22	0;47
W	119	87	23	0;17
X	120	88	24	0;45
Y	121	89	25	0;21
Z	122	90	26	0;44
1	49	33	-	0;120
2	50	64	-	0;121
3	51	35	-	0;122
4	52	36	-	0;123
5	53	37	-	0;124
6	54	94	-	0;125
7	55	38	-	0;126
8	56	42	-	0;127
9	57	40	-	0;128
0	48	41	-	0;129
-	45	95	-	0;130
=	61	43	-	0;131
Tab	9	0;15	-	-
Null	0;3	-	-	-

Appendix G

File Control Block (FCB) Structure

Figures G-1 and G-2 (memory block diagrams) and Tables G-1 and G-2 describe the structure of normal and extended file control blocks (FCBs).

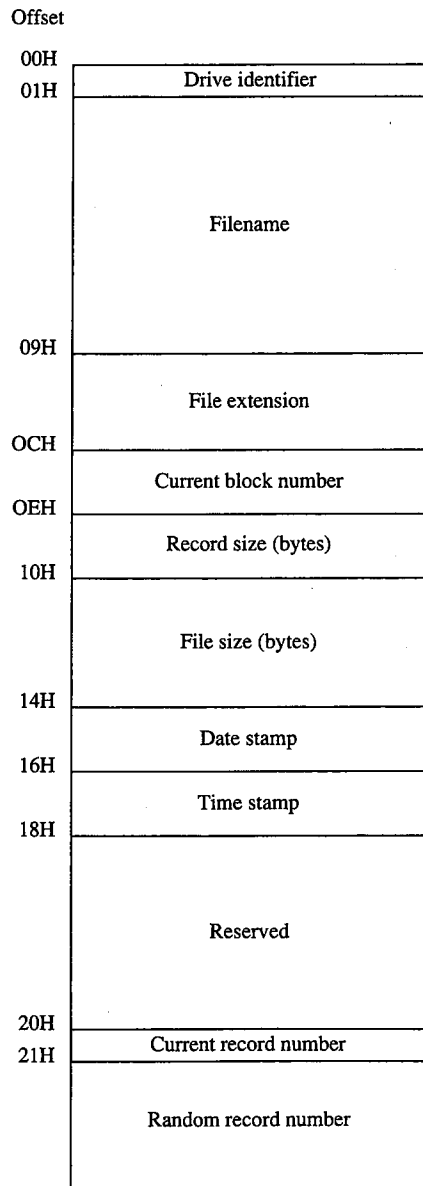


Figure G-1. Structure of a normal file control block.

Table G-1. Elements of a Normal File Control Block.

Element	Maintained by	Comments								
Drive identifier	Program	Designates the drive on which the file to be opened or created resides (0 = default drive, 1 = drive A, 2 = drive B, and so on). If the application supplies a zero in this byte, MS-DOS alters the byte during the open or create operation to reflect the actual drive used.								
Filename	Program	Standard eight-character filename; must be left justified and must be padded with blanks if fewer than eight characters. A device name (for example, PRN) can be used; there is no colon after a device name.								
File extension	Program	Three-character file extension; must be left justified and must be padded with blanks if fewer than three characters.								
Current block number	Program	Zero when the file is opened; the current block number and the current record number combined make up the record pointer during sequential file access.								
Record size	Program	Set to 128 when the file is opened or created; the program can modify the field afterward to any desired record size.*								
File size	MS-DOS	The size of the file in bytes; the first 2 bytes of this 4-byte field are the least significant bytes of the file size.								
Date stamp	MS-DOS	The date of the last write operation on the file; follows the same format used by Interrupt 21H file handle Function 57H (Get/Set Time and Date):								
		<table border="0"> <thead> <tr> <th>Bits</th> <th>Contents</th> </tr> </thead> <tbody> <tr> <td>9-15</td> <td>Year (relative to 1980)</td> </tr> <tr> <td>5-8</td> <td>Month (1-12)</td> </tr> <tr> <td>0-4</td> <td>Day of month (1-31)</td> </tr> </tbody> </table>	Bits	Contents	9-15	Year (relative to 1980)	5-8	Month (1-12)	0-4	Day of month (1-31)
Bits	Contents									
9-15	Year (relative to 1980)									
5-8	Month (1-12)									
0-4	Day of month (1-31)									
Time stamp	MS-DOS	The time of the last write operation on the file; follows the same format used by Interrupt 21H file handle Function 57H (Get/Set Time and Date):								
		<table border="0"> <thead> <tr> <th>Bits</th> <th>Contents</th> </tr> </thead> <tbody> <tr> <td>11-15</td> <td>Hours (0-23)</td> </tr> <tr> <td>5-10</td> <td>Minutes (0-59)</td> </tr> <tr> <td>0-4</td> <td>Number of 2-second increments (0-29)</td> </tr> </tbody> </table>	Bits	Contents	11-15	Hours (0-23)	5-10	Minutes (0-59)	0-4	Number of 2-second increments (0-29)
Bits	Contents									
11-15	Hours (0-23)									
5-10	Minutes (0-59)									
0-4	Number of 2-second increments (0-29)									

(more)

Table G-1. *Continued.*

Element	Maintained by	Comments
Current record number	Program	Limited to the range 0 through 127; there are 128 records per block. The beginning of a file is record 0 of block 0. Together with the current block number, this field constitutes the record pointer used during sequential read and write operations. MS-DOS does not automatically initialize this field when a file is opened.
Random record pointer	Program	Identifies the record to be transferred by the Interrupt 21H random record functions 21H, 22H, 27H, and 28H; if the record size is 64 bytes or larger, only the first 3 bytes of this field are used. MS-DOS updates this field after random block reads and writes (Functions 27H and 28H) but not after random record reads and writes (Functions 21H and 22H).

* If the record size is made larger than 128 bytes, the default data transfer area (DTA) in the program segment prefix (PSP) cannot be used because it will collide with the program's own code or data.

Table G-2. Additional Elements of an Extended File Control Block.

Element	Maintained by	Comments																		
Extended FCB flag	Program	0FFH tells MS-DOS this is an extended (44-byte) FCB.																		
File attribute byte	Program	Must be initialized by the application when an extended FCB is used to open or create a file. The bits of this field have the following significance: <table border="1" style="margin-left: 40px;"> <thead> <tr> <th>Bit</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Read-only</td> </tr> <tr> <td>1</td> <td>Hidden</td> </tr> <tr> <td>2</td> <td>System</td> </tr> <tr> <td>3</td> <td>Volume label</td> </tr> <tr> <td>4</td> <td>Directory</td> </tr> <tr> <td>5</td> <td>Archive</td> </tr> <tr> <td>6</td> <td>Reserved</td> </tr> <tr> <td>7</td> <td>Reserved</td> </tr> </tbody> </table>	Bit	Meaning	0	Read-only	1	Hidden	2	System	3	Volume label	4	Directory	5	Archive	6	Reserved	7	Reserved
Bit	Meaning																			
0	Read-only																			
1	Hidden																			
2	System																			
3	Volume label																			
4	Directory																			
5	Archive																			
6	Reserved																			
7	Reserved																			

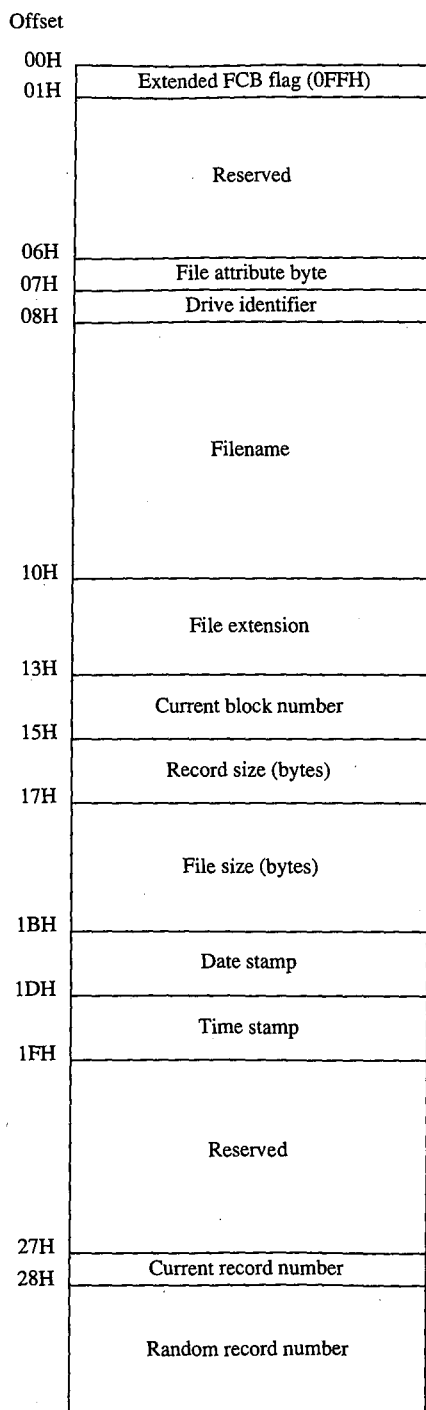


Figure G-2. Structure of an extended file control block.

Appendix H

Program Segment Prefix (PSP) Structure

Offset	Size (in bytes)	Contents
00H (0)	2	INT 20H instruction
02H (2)	2	Address of last segment allocated to program
04H (4)	1	Reserved; normally 0
05H (5)	5	Long call to MS-DOS function dispatcher
0AH (10)	4	Terminate program interrupt vector (Interrupt 22H)
0EH (14)	4	Ctrl-C handler interrupt vector (Interrupt 23H)
12H (18)	4	Critical error handler interrupt vector (Interrupt 24H)
16H (22)	22	Reserved
2CH (44)	2	Segment address of environment
2EH (46)	34	Reserved
50H (80)	3	INT 21H, RETF instructions
53H (83)	9	Reserved
5CH (92)	16	Default file control block 1
6CH (108)	20	Default file control block 2 (overlaid if FCB 1 opened)
80H (128)	127	Command tail and default DTA
FFH (255)		

Figure H-1 (memory block diagram) illustrates the structure of the program segment prefix (PSP).

Figure H-1. Structure of the program segment prefix.

Appendix I

8086/8088/80286/80386 Instruction Sets

The 8086/8088 Instruction Set

Mnemonic	Description	Mnemonic	Description
AAA	ASCII adjust after addition	JB	Jump on below
AAD	ASCII adjust before division	JBE	Jump on below or equal
AAM	ASCII adjust after multiplication	JC	Jump on carry
AAS	ASCII adjust after subtraction	JCXZ	Jump on CX zero
ADC	Add with carry	JE	Jump on equal
ADD	Add	JG	Jump on greater
AND	Logical AND	JGE	Jump on greater or equal
CALL	Call procedure	JL	Jump on less than
CBW	Convert byte to word	JLE	Jump on less than or equal
CLC	Clear carry flag	JMP	Jump unconditionally
CLD	Clear direction flag	JNA	Jump on not above
CLI	Clear interrupt flag	JNAE	Jump on not above or equal
CMC	Complement carry flag	JNB	Jump on not below
CMP	Compare	JNBE	Jump on not below or equal
CMPS	Compare string	JNC	Jump on no carry
CMPSB	Compare byte string	JNE	Jump on not equal
CMPSW	Compare word string	JNG	Jump on not greater
CWD	Convert word to doubleword	JNGE	Jump on not greater or equal
DAA	Decimal adjust for addition	JNL	Jump on not less than
DAS	Decimal adjust for subtraction	JNLE	Jump on not less than or equal
DEC	Decrement by 1	JNO	Jump on not overflow
DIV	Unsigned divide	JNP	Jump on not parity
ESC	Escape	JNS	Jump on not sign
HLT	Halt	JNZ	Jump on not zero
IDIV	Integer divide	JO	Jump on overflow
IMUL	Integer multiply	JP	Jump on parity
IN	Input from port	JPE	Jump on parity even
INC	Increment by 1	JPO	Jump on parity odd
INT	Call to interrupt procedure	JS	Jump on sign
INTO	Interrupt on overflow	JZ	Jump on zero
IRET	Interrupt on return	LAHF	Load AH with flags
JA	Jump on above	LDS	Load pointer into DS
JAE	Jump on above or equal	LEA	Load effective address

(more)

Mnemonic	Description	Mnemonic	Description
LES	Load pointer into ES	REPNE	Repeat while not equal
LOCK	Lock the bus	REPNZ	Repeat while not zero
LODS	Load string	REPZ	Repeat while zero
LODSB	Load byte (string)	RET	Return
LODSW	Load word (string)	ROL	Rotate left
LOOP	Loop	ROR	Rotate right
LOOPE	Loop while equal	SAHF	Store AH into flags
LOOPNE	Loop while not equal	SAL	Shift arithmetic left
LOOPNZ	Loop while not zero	SAR	Shift arithmetic right
LOOPZ	Loop while zero	SBB	Subtract with borrow
MOV	Move data	SCAS	Scan string
MOVS	Move data from string to string	SCASB	Scan byte (string)
MOVSB	Move byte (string)	SCASW	Scan word (string)
MOVSW	Move word (string)	SHL	Shift logical left
MUL	Multiply	SHR	Shift logical right
NEG	Negate	STC	Set carry flag
NOP	No operation	STD	Set direction flag
NOT	Logical NOT	STI	Set interrupt flag
OR	Logical OR	STOS	Store string
OUT	Output to port	STOSB	Store byte (string)
POP	Pop top of stack	STOSW	Store word (string)
POPF	Pop stack into flags	SUB	Subtract
PUSH	Push onto stack	TEST	Logical compare
PUSHF	Push flags onto stack	WAIT	Enter wait state
RCL	Rotate through carry left	XCHG	Exchange
RCR	Rotate through carry right	XLAT	Translate
REP	Repeat	XOR	Exclusive OR
REPE	Repeat while equal		

The 80286 Instruction Set

Mnemonic	Description	Mnemonic	Description
AAA	ASCII adjust after addition	AND	Logical AND
AAD	ASCII adjust before division	ARPL	Adjust RPL field of selector
AAM	ASCII adjust after multiplication	BOUND	Check array index against bounds
AAS	ASCII adjust after subtraction	CALL	Call procedure
ADC	Add with carry	CBW	Convert byte to word
ADD	Add	CLC	Clear carry flag

(more)

Mnemonic	Description	Mnemonic	Description
CLD	Clear direction flag	JNE	Jump on not equal
CLI	Clear interrupt flag	JNG	Jump on not greater
CLTS	Clear task switched flag	JNGE	Jump on not greater or equal
CMC	Complement carry flag	JNL	Jump on not less than
CMP	Compare	JNLE	Jump on not less than or equal
CMPS	Compare string	JNO	Jump on not overflow
CMPSB	Compare byte string	JNP	Jump on not parity
CMPSW	Compare word string	JNS	Jump on not sign
CWD	Convert word to doubleword	JNZ	Jump on not zero
DAA	Decimal adjust for addition	JO	Jump on overflow
DAS	Decimal adjust for subtraction	JP	Jump on parity
DEC	Decrement by 1	JPE	Jump on parity even
DIV	Unsigned divide	JPO	Jump on parity odd
ENTER	Make stack frame (for procedure parameters)	JS	Jump on sign
ESC	Escape	JZ	Jump on zero
HLT	Halt	LAHF	Load AH with flags
IDIV	Integer divide	LAR	Load access-rights byte
IMUL	Integer multiply	LDS	Load pointer into DS
IN	Input from port	LEA	Load effective address
INC	Increment by 1	LEAVE	High-level procedure exit
INS	Input string from port	LES	Load pointer into ES
INT	Call to interrupt procedure	LGDT	Load global descriptor table
INTO	Interrupt on overflow	LIDT	Load interrupt descriptor table
IRET	Interrupt on return	LLDT	Load local descriptor table
JA	Jump on above	LMSW	Load machine status word
JAE	Jump on above or equal	LOCK	Lock the bus
JB	Jump on below	LODS	Load string
JBE	Jump on below or equal	LODSB	Load byte (string)
JC	Jump on carry	LODSW	Load word (string)
JCXZ	Jump on CX zero	LOOP	Loop
JE	Jump on equal	LOOPE	Loop while equal
JG	Jump on greater	LOOPNE	Loop while not equal
JGE	Jump on greater or equal	LOOPNZ	Loop while not zero
JL	Jump on less than	LOOPZ	Loop while zero
JLE	Jump on less than or equal	LSL	Load segment limit
JMP	Jump unconditionally	LTR	Load task register
JNA	Jump on not above	MOV	Move data
JNAE	Jump on not above or equal	MOVS	Move data from string to string
JNB	Jump on not below	MOVSB	Move byte (string)
JNBE	Jump on not below or equal	MOVSW	Move word (string)
JNC	Jump on no carry	MUL	Multiply
		NEG	Negate

(more)

Mnemonic	Description	Mnemonic	Description
NOP	No operation	SCAS	Scan string
NOT	Logical NOT	SCASB	Scan byte (string)
OR	Logical OR	SCASW	Scan word (string)
OUT	Output to port	SGDT	Store global descriptor table
OUTS	Output string to port	SHL	Shift logical left
POP	Pop top of stack	SHR	Shift logical right
POPA	Pop eight 16-bit registers	SIDT	Store interrupt descriptor table
POPF	Pop stack into flags	SLDT	Store local descriptor table
PUSH	Push onto stack	SMSW	Store machine status word
PUSHA	Push eight 16-bit registers	STC	Set carry flag
PUSHF	Push flags onto stack	STD	Set direction flag
RCL	Rotate through carry left	STI	Set interrupt flag
RCR	Rotate through carry right	STOS	Store string
REP	Repeat	STOSB	Store byte (string)
REPE	Repeat while equal	STOSW	Store word (string)
REPNE	Repeat while not equal	STR	Store task register
REPNZ	Repeat while not zero	SUB	Subtract
REPZ	Repeat while zero	TEST	Logical compare
RET	Return	VERR	Verify a segment for reading
ROL	Rotate left	VERW	Verify a segment for writing
ROR	Rotate right	WAIT	Enter wait state
SAHF	Store AH into flags	XCHG	Exchange
SAL	Shift arithmetic left	XLAT	Translate
SAR	Shift arithmetic right	XOR	Exclusive OR
SBB	Subtract with borrow		

The 80386 Instruction Set

Mnemonic	Description	Mnemonic	Description
AAA	ASCII adjust after addition	BSF	Bit scan forward
AAD	ASCII adjust before division	BSR	Bit scan reverse
AAM	ASCII adjust after multiplication	BT	Bit test
AAS	ASCII adjust after subtraction	BTC	Bit test and complement
ADC	Add with carry	BTR	Bit test and reset
ADD	Add	BTS	Bit test and set
AND	Logical AND	CALL	Call procedure
ARPL	Adjust RPL field of selector	CBW	Convert byte to word
BOUND	Check array index against bounds	CDQ	Convert doubleword to quad word

(more)

Mnemonic	Description	Mnemonic	Description
CLC	Clear carry flag	JMP	Jump unconditionally
CLD	Clear direction flag	JNA	Jump on not above
CLI	Clear interrupt flag	JNAE	Jump on not above or equal
CLTS	Clear task switched flag	JNB	Jump on not below
CMC	Complement carry flag	JNBE	Jump on not below or equal
CMP	Compare	JNC	Jump on no carry
CMPS	Compare string	JNE	Jump on not equal
CMPSB	Compare byte string	JNG	Jump on not greater
CMPSD	Compare doubleword string	JNGE	Jump on not greater or equal
CMPSW	Compare word string	JNL	Jump on not less than
CWD	Convert word to doubleword	JNLE	Jump on not less than or equal
DAA	Decimal adjust for addition	JNO	Jump on not overflow
DAS	Decimal adjust for subtraction	JNP	Jump on not parity
DEC	Decrement by 1	JNS	Jump on not sign
DIV	Unsigned divide	JNZ	Jump on not zero
ENTER	Make stack frame (for procedure parameters)	JO	Jump on overflow
ESC	Escape	JP	Jump on parity
HLT	Halt	JPE	Jump on parity even
IDIV	Integer divide	JPO	Jump on parity odd
IMUL	Integer multiply	JS	Jump on sign
IN	Input from port	JZ	Jump on zero
INC	Increment by 1	LAHF	Load AH with flags
INS	Input string from port	LAR	Load access-rights byte
INSD	Input doubleword from port	LDS	Load pointer into DS
INT	Call to interrupt procedure	LEA	Load effective address
INTO	Interrupt on overflow	LEAVE	High-level procedure exit
IRET	Interrupt on return	LES	Load pointer into ES
IRETD	Interrupt return to virtual 8086 mode	LFS	Load pointer into FS
JA	Jump on above	LGDT	Load global descriptor table
JAE	Jump on above or equal	LGS	Load pointer into GS
JB	Jump on below	LIDT	Load interrupt descriptor table
JBE	Jump on below or equal	LLDT	Load local descriptor table
JC	Jump on carry	LMSW	Load machine status word
JCXZ	Jump on CX zero	LOCK	Lock the bus
JE	Jump on equal	LODS	Load string
JECXZ	Jump on ECX zero	LODSB	Load byte (string)
JG	Jump on greater	LODSD	Load doubleword (string)
JGE	Jump on greater or equal	LODSW	Load word (string)
JL	Jump on less than	LOOP	Loop
JLE	Jump on less than or equal	LOOPE	Loop while equal
		LOOPNE	Loop while not equal
		LOOPNZ	Loop while not zero

(more)

Mnemonic	Description	Mnemonic	Description
LOOPZ	Loop while zero	ROL	Rotate left
LSL	Load segment limit	ROR	Rotate right
LSS	Load pointer into SS	SAHF	Store AH into flags
LTR	Load task register	SAL	Shift arithmetic left
MOV	Move data	SAR	Shift arithmetic right
MOVS	Move data from string to string	SBB	Subtract with borrow
MOVSB	Move byte (string)	SCAS	Scan string
MOVSD	Move doubleword (string)	SCASB	Scan byte (string)
MOVSW	Move word (string)	SCASD	Scan doubleword (string)
MOVSBX	Move with sign extend	SCASW	Scan word (string)
MOVZXB	Move with zero extend	SET	Byte set on condition
MUL	Multiply	SGDT	Store global descriptor table
NEG	Negate	SHL	Shift logical left
NOP	No operation	SHLD	Double precision shift left
NOT	Logical NOT	SHR	Shift logical right
OR	Logical OR	SHRD	Double precision shift right
OUT	Output to port	SIDT	Store interrupt descriptor table
OUTS	Output string to port	SLDT	Store local descriptor table
POP	Pop top of stack	SMSW	Store machine status word
POPA	Pop eight 16-bit registers	STC	Set carry flag
POPAD	Pop eight 32-bit registers	STD	Set direction flag
POPF	Pop stack into flags	STI	Set interrupt flag
POPFD	Loads doubleword into EFLAGS	STOS	Store string
PUSH	Push onto stack	STOSB	Store byte (string)
PUSHA	Push eight 16-bit registers	STOSD	Store doubleword (string)
PUSHAD	Push eight 32-bit registers	STOSW	Store word (string)
PUSHED	Push EFLAGS	STR	Store task register
PUSHF	Push flags onto stack	SUB	Subtract
RCL	Rotate through carry left	TEST	Logical compare
RCR	Rotate through carry right	VERR	Verify a segment for reading
REP	Repeat	VERW	Verify a segment for writing
REPE	Repeat while equal	WAIT	Enter wait state
REPNE	Repeat while not equal	XCHG	Exchange
REPNZ	Repeat while not zero	XLAT	Translate
REPZ	Repeat while zero	XOR	Exclusive OR
RET	Return		

Appendix J

Common MS-DOS Filename Extensions

The Microsoft systems programs and language products commonly use the following filename extensions:

Extension	Program/System	Description
.@@@	MS-DOS	Backup ID file
.\$\$\$	EDLIN	Backup filename if out of disk space; error condition
.ASC	Generic	ASCII text file
.ASM	MASM	Assembly-language source code
.BAK	Generic	Backup file
.BAS	BASIC	BASIC language source code
.BAT	MS-DOS	Batch file (contains MS-DOS command lines)
.BIN	Generic	Binary file
.C	C	C language source code
.CAL	Windows	Calendar file
.COB	COBOL	COBOL language source code
.COD	Generic	Object listing file
.COM	MS-DOS	Executable program file
.CRD	Windows	Cardfile file
.CRF	MASM	Cross-reference file
.DAT	Generic	Data file
.DBG	COBOL	Debug file
.DEF	Windows	Module definition file
.DOC	Generic	Documentation or document file
.DRV	Generic	Driver file
.ERR	Generic	Error file
.EXE	MS-DOS	Executable program file
.FNT	Generic	Font file
.FON	Generic	Font file
.FOR	FORTRAN	FORTRAN language source code
.GRB	Windows	Grab file (snapshot)
.H	C	Include file
.HEX	MS-DOS	INTEL hexadecimal format file
.HLP	Generic	Help file
.INC	Generic	Include file
.INI	Windows	Initialization file

(more)

Extension	Program/System	Description
.INT	COBOL	Object file
.LIB	Generic	Library file
.LST	Generic	List file
.MAP	Generic	Address map file
.MOD	Generic	Module file
.MSG	COBOL	Message file
.MSP	Windows	Windows Paint file
.OBJ	Generic	Relocatable object module
.OVL	Generic	Overlay file
.OVR	COBOL	Compiler overlay file
.PAS	PASCAL	PASCAL language source code
.PIF	Windows	Program information file
.QLB	Generic	Library file for Microsoft's Quick products
.RC	Windows	Resource script file
.REF	CREF	Cross-reference listing file
.RES	Windows	Compiled resource file
.SCR	Generic	Script file
.SYM	Generic	Symbol file
.SYS	Generic	System file or device driver
.TMP	Generic	Temporary file
.TRM	Windows	Terminal file
.TXT	Generic	Text file or Windows Notepad file
.WRI	Windows	Write file

Appendix K

Segmented (New) .EXE File Header Format

Microsoft Windows requires much more information about a program than is available in the format of the .EXE executable file supported by MS-DOS. For example, Windows needs to identify the various segments of a program as code segments or data segments, to identify exported and imported functions, and to store the program's resources (such as icons, cursors, menus, and dialog-box templates). Windows must also support dynamically linkable library modules containing routines that programs and other library modules can call. For this reason, Windows programs use an expanded .EXE header format called the New Executable file header format. This format is used for Windows programs, Windows library modules, and resource-only files such as the Windows font resource files.

The Old Executable Header

The New Executable file header format incorporates the existing MS-DOS executable file header format. In fact, the beginning of a New Executable file is simply a normal MS-DOS .EXE header. The 4 bytes at offset 3CH are a pointer to the beginning of the New Executable header. (Offsets are from the beginning of the Old Executable header.)

Offset	Length (bytes)	Contents
00H	1	Signature byte <i>M</i>
01H	1	Signature byte <i>Z</i>
3CH	4	Offset of New Executable header from beginning of file

This normal MS-DOS .EXE header can contain size and relocation information for a non-Windows MS-DOS program that is contained within the .EXE file along with the Windows program. This program is run when the .EXE file is executed from the MS-DOS command line. Most Windows programmers use a standard program that simply prints the message *This program requires Microsoft Windows.*

The New Executable Header

The beginning of the New Executable file header contains information about the location and size of various tables within the header. (Offsets are from the beginning of the New Executable header.)

Offset	Length (bytes)	Contents
00H	1	Signature byte <i>N</i>
01H	1	Signature byte <i>E</i>
02H	1	LINK version number
03H	1	LINK revision number
04H	2	Offset of beginning of entry table relative to beginning of New Executable header
06H	2	Length of entry table
08H	4	32-bit checksum of entire contents of file, using zero for these 4 bytes
0CH	2	Module flag word (<i>see</i> below)
0EH	2	Segment number of automatic data segment (0 if neither SINGLEDATA nor MULTIPLEDATA flag is set in flag word)
10H	2	Initial size of local heap to be added to automatic data segment (0 if there is no local heap)
12H	2	Initial size of stack to be added to automatic data segment (0 for library modules)
14H	2	Initial value of instruction pointer (IP) register on entry to program
16H	2	Initial segment number for setting code segment (CS) register on entry to program
18H	2	Initial value of stack pointer (SP) register on entry to program (0 if stack segment is automatic data segment; stack should be set above static data area and below local heap in automatic data segment)

(more)

Offset	Length (bytes)	Contents
1AH	2	Segment number for setting stack segment (SS) register on entry to program (0 for library modules)
1CH	2	Number of entries in segment table
1EH	2	Number of entries in module reference table
20H	2	Number of bytes in nonresident names table
22H	2	Offset of beginning of segment table relative to beginning of New Executable header
24H	2	Offset of beginning of resource table relative to beginning of New Executable header
26H	2	Offset of beginning of resident names table relative to beginning of New Executable header
28H	2	Offset of beginning of module reference table relative to beginning of New Executable header
2AH	2	Offset of beginning of imported names table relative to beginning of New Executable header
2CH	4	Offset of nonresident names table relative to beginning of file
30H	2	Number of movable entry points listed in entry table
32H	2	Alignment shift count (0 is equivalent to 9)
34H	12	Reserved for expansion

The module flag word at offset 0CH in the New Executable header is defined as shown in Figure K-1.

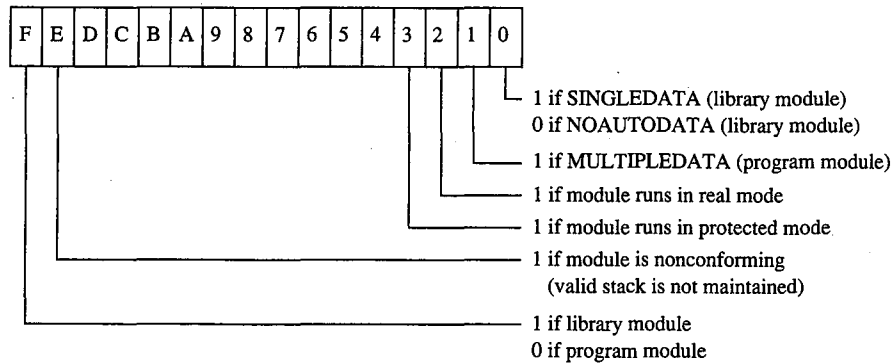


Figure K-1. The module flag word.

The segment table

This table contains one 8-byte record for every code and data segment in the program or library module. Each segment has an ordinal number associated with it. For example, the first segment has an ordinal number of 1. These segment numbers are used to reference the segments in other sections of the New Executable file. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	2	Offset of segment relative to beginning of file after shifting value left by alignment shift count
02H	2	Length of segment (0000H for segment of 65536 bytes)
04H	2	Segment flag word (<i>see below</i>)
06H	2	Minimum allocation size for segment; that is, amount of space Windows reserves in memory for segment (0000H for minimum allocation size of 65536 bytes)

The segment flag word is defined as shown in Figure K-2.

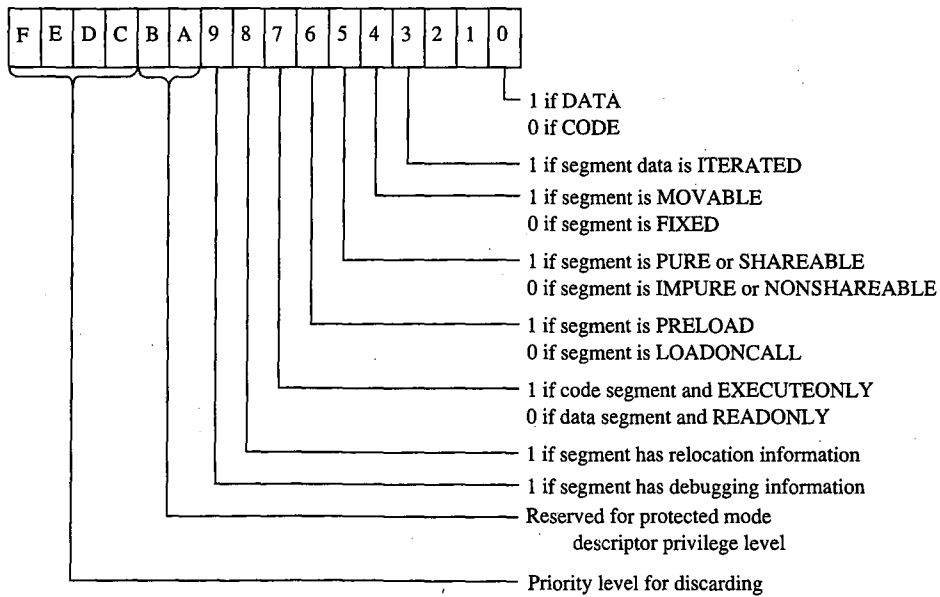


Figure K-2. The segment flag word.

The resource table

Resources are segments that contain data but are not included in a program's normal data segments. Resources are commonly used in Windows programs to store menus, dialog-box templates, icons, cursors, and text strings, but they can also be used for any type of read-only data. Each resource has a type and a name, both of which can be represented by either a number or an ASCII name.

The resource table begins with a resource shift count used for adjusting other values in the table. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	2	Resource shift count

This is followed by one or more resource groups, each defining one or more resources. (Offsets are from the beginning of the group.)

Offset	Length (bytes)	Contents
00H	2	Resource type (0 if end of table) If high bit set, type represented by predetermined number (high bit not shown): <ul style="list-style-type: none"> 1 Cursor 2 Bitmap 3 Icon 4 Menu template 5 Dialog-box template 6 String table 7 Font directory 8 Font 9 Keyboard-accelerator table If high bit not set, type is ASCII text string and this value is offset from beginning of resource table, pointing to 1-byte value with number of bytes in string followed by string itself.
02H	2	Number of resources of this type
04H	4	Reserved for run-time use
08H	12 each	Resource description

Each resource description requires 12 bytes. (Offsets are from the beginning of the description.)

Offset	Length (bytes)	Contents
00H	2	Offset of resource relative to beginning of file after shifting left by resource shift count
02H	2	Length of resource after shifting left by resource shift count
04H	2	Resource flag word (<i>see</i> below)
06H	2	Resource name If high bit set, represented by a number; otherwise, type is ASCII text string and this value is offset from beginning of resource table, pointing to 1-byte value with number of bytes in string followed by string itself.
08H	4	Reserved for run-time use

The resource flag word is defined as shown in Figure K-3.

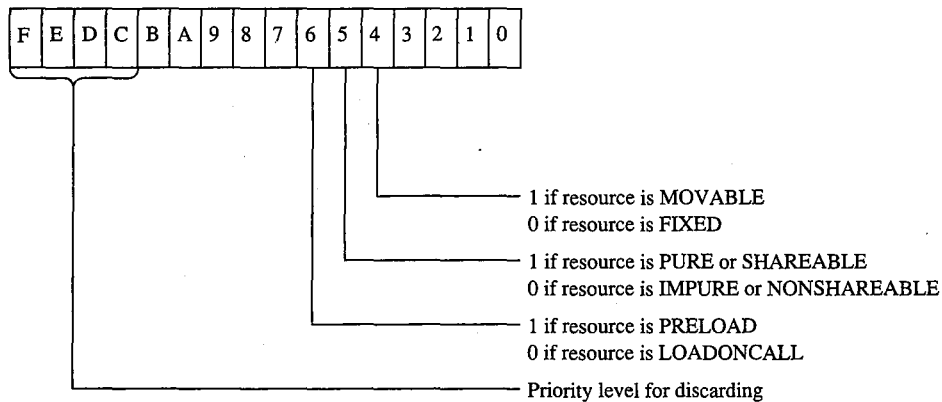


Figure K-3. The resource flag word.

The resident names table

This table contains a list of ASCII strings. The first string is the module name given in the module definition file. The other strings are the names of all exported functions listed in the module definition file that were not given explicit ordinal numbers or that were explicitly specified in the file as resident names. (Exported functions with explicit ordinal numbers in the module definition file are listed in the nonresident names table.)

Each string is prefaced by a single byte indicating the number of characters in the string and is followed by a word (2 bytes) referencing an element in the entry table, beginning at 1. The word that follows the module name is 0. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in string (0 if end of table)
01H	<i>n</i>	ASCII string, not null-terminated
<i>n</i> +1	2	Index into entry table

The module reference table

The module reference table contains 2 bytes for every external module the program uses. These 2 bytes are an offset into the imported names table.

The imported names table

The imported names table contains a list of ASCII strings. These strings are the names of all other modules that are referenced through imported functions. The strings are prefaced with a single byte indicating the length of the string.

For most Windows programs, the imported names table includes KERNEL, USER, and GDI, but it can also include names of other modules, such as KEYBOARD and SOUND. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in name string
01H	<i>n</i>	ASCII name string, not null-terminated

These strings do not necessarily start at the beginning of the imported names table; the names are referenced by offsets specified in the module reference table.

The entry table

This table contains one member for every entry point in the program or library module. (Every public FAR function or procedure in a module is an entry point.) The members in the entry table have ordinal numbers beginning at 1. These ordinal numbers are referenced by the resident names table and the nonresident names table.

LINK versions 4.0 and later bundle the members of the entry table. Each bundle begins with the following information. (Offsets are from the beginning of the bundle.)

Offset	Length (bytes)	Contents
00H	1	Number of entry points in bundle (0 if end of table)
01H	1	Segment number of entry points if entry points in bundle are in single fixed segment; 0FFH if entry points in bundle are in movable segments

For a bundle containing entry points in fixed segments, each entry point requires 3 bytes. (Offsets are from the beginning of the entry description.)

Offset	Length (bytes)	Contents
00H	1	Entry-point flag byte (<i>see</i> below)
01H	2	Offset of entry point in segment

For bundles containing entry points in movable segments, each entry point requires 6 bytes. (Offsets are from the beginning of the entry description.)

Offset	Length (bytes)	Contents
00H	1	Entry-point flag byte (<i>see</i> below)
01H	2	Interrupt 3FH instruction: CDH 3FH
03H	1	Segment number of entry point
04H	2	Offset of entry-point segment

The entry-point flag byte is defined as shown in Figure K-4.

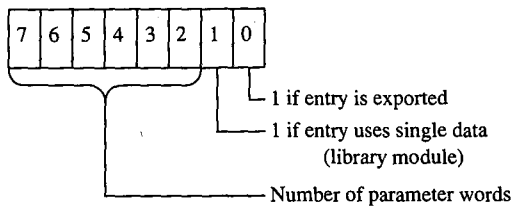


Figure K-4. The entry-point flag.

The nonresident names table

This table contains a list of ASCII strings. The first string is the module description from the module definition file. The other strings are the names of all exported functions listed in the module definition file that have ordinal numbers associated with them. (Exported functions without ordinal numbers in the module definition file are listed in the resident names table.)

Each string is prefaced by a single byte indicating the number of characters in the string and is followed by a word (2 bytes) referencing a member of the entry table, beginning at 1. The word that follows the module description string is 0. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in string (0 if end of table)
01H	<i>n</i>	ASCII string, not null-terminated
<i>n</i> +1	2	Index into entry table

The code and data segment

Following the various tables in the New Executable file header are the code and data segments of the program or library module.

If the code or data segment is flagged in the segment flag word as ITERATED, the segment is organized as follows. (Offsets are from the beginning of the segment.)

Offset	Length (bytes)	Contents
00H	2	Number of iterations of data
02H	2	Number of bytes of data
04H	<i>n</i>	Data

Otherwise, the size of the segment data is given by the length of the segment field in the segment table.

If the segment is flagged in the segment flag word as containing relocation information, then the relocation table begins immediately after the segment data. Windows uses the relocation table to resolve references within the segments to functions in other segments in the same module and to imported functions in other modules. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	2	Number of relocation items

Each relocation item requires 8 bytes. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
00H	1	Type of address to insert in segment: 01H Offset only 02H Segment only 03H Segment and offset

(more)

Offset	Length (bytes)	Contents
01H	1	Relocation type: 00H Internal reference 01H Imported ordinal 02H Imported name If bit 2 set, relocation type is additive (<i>see</i> below)
02H	2	Offset of relocation item within segment

The next 4 bytes depend on the relocation type. If the relocation type is an internal reference to a segment in the same module, these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	1	Segment number for fixed segment; 0FFH for movable segment
05H	1	0
06H	2	If MOVABLE segment, ordinal number referenced in entry table; if FIXED segment, offset into segment

If the relocation type is an imported ordinal to another module, then these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	2	Index into module reference table
06H	2	Function ordinal number

Finally, if the relocation type is an imported name of a function in another module, these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	2	Index into module reference table
06H	2	Offset within imported names table to name of imported function

If the ADDITIVE flag of the relocation type is set, the address of the external function is added to the contents of the address in the target segment. If the ADDITIVE flag is not set, then the target contains an offset to another target within the same segment that requires the same relocation address. This defines a chain of target addresses that get the same address. The chain is terminated with a -1 entry.

Charles Petzold



Appendix L

Intel Hexadecimal Object File Format

The MCS-86 hexadecimal object file format provides a means of recording a program's binary (compiled or assembled) image in a text-only (printable) file format. This format makes it easy to transfer the program between computers over telephone lines without using special communications software. More important, it provides a ready means of transferring programs between computers and the various types of laboratory equipment typically used during the development of specialized programs.

The MCS-86 hexadecimal file format is a superset of Intel's older Intellec-8 hexadecimal object file format. Intel originally designed the Intellec-8 format for use with its 8-bit microprocessor line. The format rapidly gained acceptance among other microprocessor manufacturers. When Intel subsequently developed the MCS-86 microprocessor family, it also expanded the Intellec-8 hexadecimal file format into the MCS-86 hexadecimal file format to support the new microprocessors' extended addressing capabilities.

The MCS-86 hexadecimal object file format should not be confused with the object (.OBJ) files produced by the Microsoft Macro Assembler (MASM) and language compilers. The MCS-86 hexadecimal object file format is referred to as an *absolute* object file format because the code contained within the file has been completely linked and all address references have already been resolved. The object modules produced by the assembler and compilers (.OBJ files) are referred to as *relocatable* object modules because they contain the information necessary to relocate the enclosed code to any memory address for execution.

The MCS-86 hexadecimal object file format consists of four types of ASCII text records:

- Data record
- End-of-file record
- Extended-address record
- Start-address record

All records begin with a *record mark* consisting of a single ASCII colon character (:). The remainder of the record consists of a variable number of ASCII hexadecimal digit pairs (00–0FH), each representing an unsigned byte value (0–255 decimal). The first digit represents the value of the high nibble (bits 7–4) of the byte; the second digit represents the value of the low nibble (bits 3–0). These digit pairs begin immediately after the record mark and continue through the end of the record without any separation between them.

All records have the following fields, in the order listed:

- A fixed-length *record length* field
- A fixed-length *address* field (optional)
- A fixed-length *record type* field

- A fixed-length or variable-length *data* field
- A fixed-length *checksum* field

The fixed-length *record length* field consists of the first digit pair following the record mark and gives the length of the record-type-dependent variable-length data field.

The optional fixed-length *address* field consists of the second and third digit pairs following the record mark. The first digit pair of this field (second digit pair of the record) gives the high byte of a word address value (bits 15–8); the second digit pair (third digit pair of the record) gives the low byte of a word address value (bits 7–0). If the record type does not use the address field, then the field contains a fill-in value consisting of the four-character ASCII string *0000*.

The fixed-length *record type* field consists of the fourth digit pair of the record and indicates the type of data the record contains. The valid record-type values are

Value	Type
00H	Data record
01H	End-of-file record
02H	Extended-address record
03H	Start-address record

All records end with a fixed-length *checksum* field. This field contains the negative of the sum of all byte values represented by the digit pairs in the record, from the record length field through the last digit pair before the checksum field. The checksum field is used to determine whether an error occurred during the transmission of a record between computers or other pieces of equipment.

(The receiving equipment can easily perform this error checking as each record is received. It only has to add all digit pairs of the record, including the checksum, and ignore any overflow beyond 8 bits. The total should be 00H, because the checksum is the negative of the summation of all preceding digit pairs.)

The variable-length *data* field of the data record contains the actual data bytes of the program's image. In data records, the record length field indicates the number of bytes, each represented as a digit pair, contained within the data field; the address field gives the offset within the current memory segment at which to load the record's data into memory.

The fixed-length data field of the extended-address record establishes the memory segment into which subsequent data records are to be loaded. In extended-address records, the data field consists of a single field identical to the address field. The address field of an extended-address record always contains the ASCII 0000 filler, and the record length field always contains ASCII 02, which reflects the fixed length of the data field. The memory segment (also known as the memory frame) established by an extended-address record remains in effect until the next extended-address record is encountered; thus, all data

records following the most recent extended-address record are loaded in the established memory segment. See PROGRAMMING IN MS-DOS: PROGRAMMING TOOLS: The Microsoft Object Linker.

Figures L-1 and L-2 show how the extended-address record and the data record combine to load the byte values 0FDH, 0B9H, 75H, 31H, 0ECH, 0A8H, 64H, and 20H into memory starting at address 9A6EH:429FH.

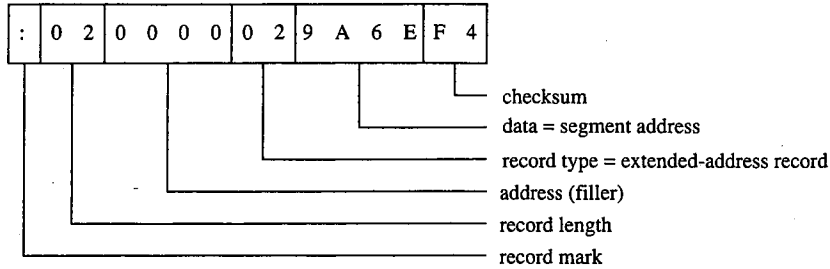


Figure L-1. The extended-address record.

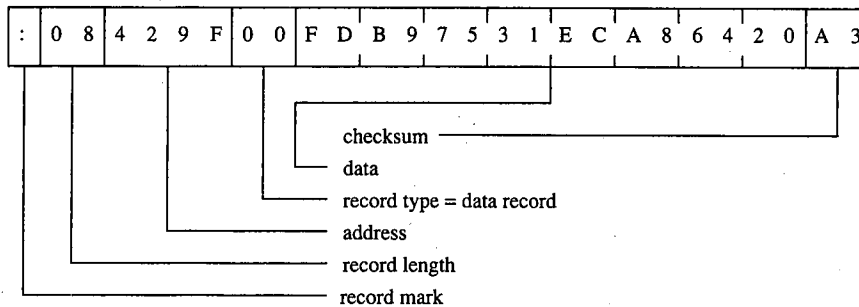


Figure L-2. The data record.

The start-address record provides the CS and IP register values at which program execution begins. This record contains the register values within the fixed-length data field. The address field of a start-address record always contains the ASCII 0000 filler, and the record length field always contains ASCII 04, which reflects the fixed length of the data field. The example in Figure L-3 shows a CS:IP setting (program entry point) of F924H:E69AH.

The end-of-file record marks the end of an MCS-86 hexadecimal file. Under the MCS-86 hexadecimal file definition, the end-of-file record does not contain any variable-value fields; the record always appears as shown in Figure L-4.

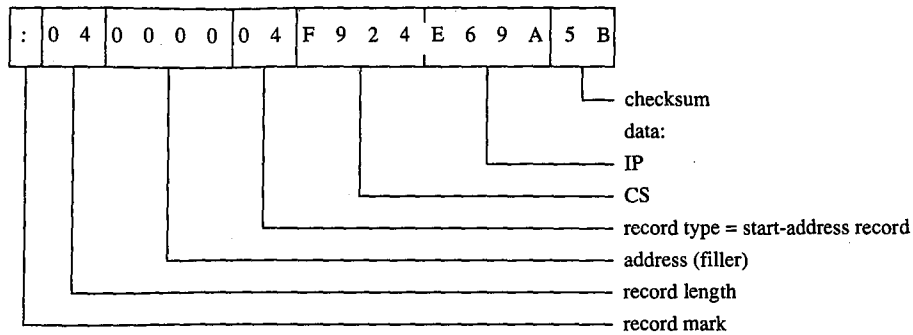


Figure L-3. The start-address record.

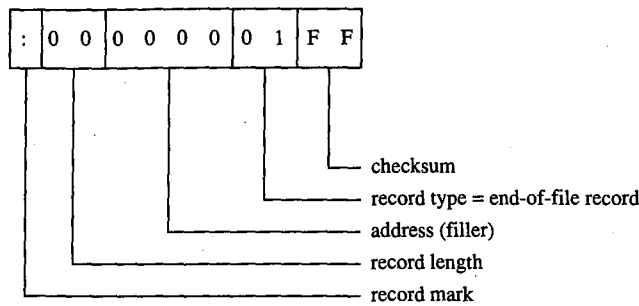


Figure L-4. The end-of-file record.

Traditionally, development equipment and programs that accept the MCS-86 hexadecimal file format as input also recognize an alternate end-of-file record. The alternate record consists of a data record that contains no data; therefore, its record length field contains 00. Figure L-5 shows this alternate end-of-file record.

DEBUG is the only program supplied with MS-DOS that accepts the MCS-86 hexadecimal file format. Even then, DEBUG only loads hexadecimal files into memory; it does not save a program back to disk as a hexadecimal file. (The same applies for SYMDEB and for CodeView.)

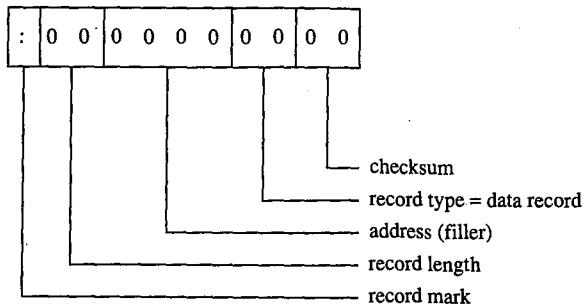


Figure L-5. The alternate end-of-file record.

While loading a hexadecimal file, DEBUG actually processes only data records and end-of-file records; it ignores both start-address records and any extended-address records. Thus, DEBUG actually supports only the older Intellec-8 hexadecimal file format but will not reject the file if it also contains the newer MCS-86 hexadecimal file records.

DEBUG does not support MCS-86 records because it must operate within the MS-DOS environment and MS-DOS does not support the loading of programs into absolute memory locations—a restriction imposed by most general-purpose operating systems. Because DEBUG cannot load the data records into the absolute segments indicated by the extended-address records, it simply loads the program image contained within the data records in a manner similar to that in which a .COM program is loaded. See PROGRAMMING IN THE MS-DOS ENVIRONMENT: PROGRAMMING FOR MS-DOS: Structure of an Application Program. DEBUG uses the address field for the data records as the offset into the .COM program segment at which to load the contents of the records.

The sample QuickBASIC (versions 3.0 and later) program shown in Figure L-6 converts binary files, including .COM files, into limited MCS-86 hexadecimal files that DEBUG can load. Examining this program can provide additional understanding of the structure of Intel hexadecimal files.

```
'Binary-to-Hex file conversion utility.
'Requires Microsoft QuickBASIC version 3.0 or later.

DEFINT A-Z                                ' All variables are integers
                                           ' unless otherwise declared.
CONST FALSE = 0                            ' Value of logical FALSE.
CONST TRUE = NOT FALSE                     ' Value of logical TRUE.

DEF FNHXB$(X) = RIGHT$(HEX$(&H100 + X), 2) ' Return 2-digit hex value for X.
DEF FNHW$(X!) = RIGHT$("000" + HEX$(X!), 4) ' Return 4-digit hex value for X!.
DEF FNMOD(X, Y) = X! - INT(X!/Y) * Y        ' X! MOD Y (the MOD operation is
                                           ' only for integers).
CONST SRCCNL = 1                            ' Source (.BIN) file channel.
CONST TGTCNL = 2                            ' Target (.HEX) file channel.

LINE INPUT "Enter full name of source .BIN file      : ";SRCFIL$
OPEN SRCFIL$ FOR INPUT AS SRCCNL             ' Test for source (.BIN) file.
SRCSIZ! = LOF(SRCCNL)                       ' Save file's size.
CLOSE SRCCNL
IF (SRCSIZ! > 65536) THEN                    ' Reject if file exceeds 64 KB.
    PRINT "Cannot convert file larger than 64 KB."
    END
END IF

LINE INPUT "Enter full name of target .HEX file     : ";TGTFIL$
OPEN TGTFIL$ FOR OUTPUT AS TGTCNL           ' Test target (.HEX) filename.
CLOSE TGTCNL
```

Figure L-6. QuickBASIC binary-to-hexadecimal file conversion utility.

(more)

```

DO
LINE INPUT "Enter starting address of .BIN file in HEX : ";L$
ADRBN! = VAL("&H" + L$)           ' Convert ASCII HEX address value
                                   ' to binary value.
IF (ADRBN! < 0) THEN               ' HEX values 8000-FFFFH convert
  ADRBN! = 65536 + ADRBN!         ' to negative values.
END IF
ADREN! = ADRBN! + SRCIZ! - 1      ' Calculate resulting end address.
IF (ADREN! > 65535) THEN           ' Reject if address exceeds FFFFH.
  PRINT "Entered start address causes end address to exceed FFFFH."
END IF
LOOP UNTIL (ADRFLD! >= 0) AND (ADRFLD! <= 65535) AND (ADREN! <= 65535)

DO
LINE INPUT "Enter byte count for each record in HEX : ";L$
SRCRLN = VAL("&H" + L$)           ' Convert ASCII HEX max record
                                   ' length value to binary value.
IF (SRCRLN < 0) THEN               ' HEX values 8000-FFFFH convert
  SRCRLN = 65536 + SRCRLN         ' to negative values.
END IF
LOOP UNTIL (SRCRLN > 0) AND (SRCRLN < 256) ' Ask again if not 1-255.

OPEN SRCFIL$ AS SRCCLN LEN = SRCRLN ' Reopen source for block I/O.
FIELD#SRCCLN,SRCRLN AS SRCBLK$
OPEN TGTFIL$ FOR OUTPUT AS TGTCNL  ' Reopen target for text output.
SRCREC = 0                          ' Starting source block # minus 1.

FOR ADRFLD! = ADRBN! TO ADREN! STEP SRCRLN ' Convert one block per loop.
  SRCREC = SRCREC + 1                ' Next source block.
  GET SRCCLN,SRCREC                  ' Read the source block.
  IF (ADRFLD! + SRCRLN > ADREN!) THEN ' If last block less than full
    BLK$=LEFT$(SRCBLK$,ADREN!-ADRFLD!+1) ' size: trim it.
  ELSE                                ' Else:
    BLK$ = SRCBLK$                  ' Use full block.
  END IF

  PRINT#TGTCNL, ":";                 ' Write record mark.

  PRINT#TGTCNL, FNHB$(LEN(BLK$));    ' Write data field size.
  CHKSUM = LEN(BLK$)                 ' Initialize checksum accumulate
                                   ' with first value.
  PRINT#TGTCNL, FNHW$(ADRFLD!);     ' Write record's load address.

  ' The following "AND &HFF" operations limit CHKSUM to a byte value.
  CHKSUM = CHKSUM + INT(ADRFLD!/256) AND &HFF ' Add hi byte of adrs to csum.
  CHKSUM = CHKSUM + FNMOD(ADRFLD!,256) AND &HFF ' Add lo byte of adrs to csum.

  PRINT#TGTCNL, FNHB$(0);           ' Write record type.

```

Figure L-6. Continued.

(more)

```
' Don't bother to add record type byte to checksum since it's 0.
FOR IDX = 1 TO LEN(BLK$)      ' Write all bytes.
  PRINT#TGTCNL,FNHB$(ASC(MID$(BLK$,IDX,1)));      ' Write next byte.
  CHKSUM = CHKSUM + ASC(MID$(BLK$,IDX,1)) AND &HFF ' Incl byte in csum.
NEXT IDX

CHKSUM = 0 - CHKSUM AND &HFF      ' Negate checksum then limit
                                  ' to byte value.
PRINT #TGTCNL,FNHB$(CHKSUM)      ' End record with checksum.

NEXT ADRFLD!

PRINT#TGTCNL, ":00000001FF"      ' Write end-of-file record.

CLOSE TGTCNL                      ' Close target file.
CLOSE SRCCNL                      ' Close source file.

END
```

Figure L-6. Continued.

Keith Burgoyne

Appendix M

8086/8088 Software Compatibility Issues

In general, the Intel 80286 microprocessor running in real mode executes 8086/8088 software correctly. The following is a list of the actions to take to compensate for the minor differences between the 8086/8088 and real mode of the 80286.

- *Do not rely on 8086/8088 instruction clock counts.* The 80286 takes fewer clocks for most instructions than the 8086/8088. The areas to look into are delays between I/O operations and assumed delays when the 8086/8088 is operating in parallel with an 8087 coprocessor.
- *Note that divide exceptions point to the DIV instruction.* Any interrupt on the 80286 always leaves the saved CS:IP value pointing to the instruction that failed. On the 8086/8088, the CS:IP value saved for a divide exception points to the next instruction.
- *Set up numeric exception handlers to allow prefixes.* The saved CS:IP value in the NPX environment save area points to any ESC instruction prefixes. On 8086/8088 systems, this value points only to the ESC instruction.
- *Do not attempt undefined 8086/8088 operations.* 8086/8088 instructions like POP CS or MOV CS,op either invoke exception 06H (Invalid Opcode) or perform a protection setup operation like LIDT on the 80286. Undefined bit encodings for bits 5–3 of the second byte of POP MEM or PUSH MEM invoke exception 13H on the 80286.
- *Do not rely on the value written by PUSH SP.* The 80286 pushes a different value on the stack for PUSH SP than does the 8086/8088. If the value pushed is important, replace PUSH SP instructions with the following instructions:

```
PUSH      BP
MOV       BP, SP
XCHG     BP, [BP]
```

This code functions like the 8086/8088 PUSH SP instruction on the 80286.

- *Do not shift or rotate by more than 31 bits.* The 80286 masks all SHIFT/ROTATE counts to the low 5 bits. This MOD 32 operation limits the count to a maximum of 31 bits. With this change, the longest SHIFT/ROTATE instruction is 39 clocks. Without this change, the longest SHIFT/ROTATE instruction is 264 clocks, which delays interrupt response until the instruction completes execution.
- *Do not duplicate prefixes.* The 80286 sets an instruction-length limit of 10 bytes. The only way to exceed this limit is to include the same prefix two or more times before an instruction. Exception 06H occurs if the instruction-length limit is violated. The 8086/8088 has no instruction-length limit.
- *Do not rely on odd 8086/8088 LOCK characteristics.* The LOCK prefix and its corresponding output signal should be used only to prevent other bus masters from interrupting a data movement operation. The 80286 always asserts LOCK during an XCHG instruction with memory (even if the LOCK prefix was not used). LOCK should be

used only with the XCHG, MOV, MOVS, INS, and OUTS instructions. The 80286 LOCK signal will *not* go active during an instruction prefetch.

- *Do not rely on IDIV exceptions for quotients of 80H or 8000H.* The 80286 can generate the largest negative number as a quotient for IDIV instructions. The 8086/8088 generates exception 00H (Divide by Zero) instead.
- *Do not rely on address space wraparound.*
- *Do not use I/O ports 0F8–0FFH.* These are reserved for controlling the 80287 and future microprocessor extensions.

Appendix N

An Object Module Dump Utility

The program OBJDUMP.C displays the contents of an object file as individual object records. It can be used to study the structure of object modules as well as to verify the output of a language translator. The program recognizes all of the object record types discussed in PROGRAMMING IN THE MS-DOS ENVIRONMENT: PROGRAMMING TOOLS: Object Modules.

OBJDUMP.C should be executed with the following syntax:

```
OBJDUMP filename
```

where *filename* is a complete filename specification. For example, to dump the contents of the object file MYPROG.OBJ, the user would type

```
C>OBJDUMP MYPROG.OBJ <Enter>
```

The following is a typical object record as displayed by OBJDUMP:

```
Record 9: 96h LNames
96 002Eh 00 06 44 47 52 4F 55 50 05 5F 54 45 58 54 04 43  ..DGROUP._TEXT.C
          4F 44 45 05 5F 44 41 54 41 04 44 41 54 41 05 43  ODE._DATA.DATA.C
          4F 4E 53 54 04 5F 42 53 53 03 42 53 53 3F      ONST._BSS.BSS?
```

This sample LNames record defines a null name and eight names used in subsequent SEGDEF and GRPDEF records. The first 3 bytes of the record (the identifying byte and the 2-byte record length) are displayed to the left of the hexadecimal and ASCII listings of the contents of the record.

```

/*****
*
* OBJDUMP.C -- display contents of an object file
*
*
*   Compile:  msc objdump;   (Microsoft C version 4.0 or later)
*   Link:     link objdump;
*   Execute:  objdump <filename>
*
*****/

#include      <fcntl.h>

#define      TRUE      1
#define      FALSE     0
```

(more)

```
main( argc, argv )
int     argc;
char    **argv;
{
    unsigned char    CurrentByte;
    int     ObjFileHandle;
    int     CurrentLineLength;          /* length of output line */
    int     ObjRecordNumber = 0;
    int     ObjRecordLength;
    int     ObjRecordOffset = 0;      /* offset into current object record */
    char    ASCIIEquiv[17];
    char    FormatString[24];
    char    *ObjRecordName();
    char    *memset();

    /* open the object file */

    ObjFileHandle = open( argv[1],O_BINARY );

    if( ObjFileHandle == -1 )
    {
        printf( "\nCan't open object file\n" );
        exit( 1 );
    }

    /* process the object file character by character */

    while( read( ObjFileHandle, &CurrentByte, 1 ) )
    {
        switch( ObjRecordOffset ) /* action depends on offset into record */
        {
            case(0):                /* start of object record */
                printf( "\n\nRecord %d: %02Xh %s",
                    ++ObjRecordNumber, CurrentByte, ObjRecordName( CurrentByte ) );
                printf( "\n%02X ", CurrentByte );
                ++ObjRecordOffset;
                break;

            case(1):                /* first byte of length field */
                ObjRecordLength = CurrentByte;
                ++ObjRecordOffset;
                break;

            case(2):                /* second byte of length field */
                ObjRecordLength += CurrentByte << 8; /* compute record length */
                printf( "%04Xh ", ObjRecordLength ); /* show length */
                CurrentLineLength = 0;
                memset( ASCIIEquiv, '\0', 17 ); /* zero this string */
                ++ObjRecordOffset;
                break;
        }
    }
}
```

(more)

```

default:                /* remaining bytes in object record */
printf( "%02X ", CurrentByte );                /* hex */

if( CurrentByte < 0x20 || CurrentByte > 0x7F ) /* ASCII */
    CurrentByte = '.';
ASCIIEquiv[CurrentLineLength++] = CurrentByte;

if( CurrentLineLength == 16 || /* if end of output line ... */
    ObjRecordOffset == ObjRecordLength+2 )
{
    /* ... display it */
    sprintf( FormatString, "%%ds%s\n",
              3*(16-CurrentLineLength)+2 );
    printf( FormatString, " ", ASCIIEquiv );
    memset( ASCIIEquiv, '\0', 17 );
    CurrentLineLength = 0;
}

if( ++ObjRecordOffset == ObjRecordLength+3 ) /* if done ... */
    ObjRecordOffset = 0; /* ... process another record */
break;
}
}

if( CurrentLineLength ) /* display remainder of last output line */
    printf( " %s", ASCIIEquiv );

close( ObjFileHandle );

printf( "\n%d object records\n", ObjRecordNumber );

return( 0 );
}

```

```

char *ObjRecordName( n )                /* return object record name */
int     n;                               /* n = record type */
{
    int     i;

    static struct
    {
        int     RecordNumber;
        char     *RecordName;
    } RecordStruct[] =
    {
        { 0x80, "THEADR",
          0x88, "COMENT",
          0x8A, "MODEND",
          0x8C, "EXTDEF",
          0x8E, "TYPDEF",
          0x90, "PUBDEF",

```

(more)

```
        0x94, "LINNUM",
        0x96, "LNAMES",
        0x98, "SEGDEF",
        0x9A, "GRPDEF",
        0x9C, "FIXUPP",
        0xA0, "LEDATA",
        0xA2, "LIDATA",
        0xB0, "COMDEF",
        0x00, "*****"
    };

    int    RecordTableSize = sizeof(RecordStruct)/sizeof(RecordStruct[0]);

    for( i=0; i<RecordTableSize-1; i++ )        /* scan table for name */
        if ( RecordStruct[i].RecordNumber == n )
            break;

    return( RecordStruct[i].RecordName );
}
```

Richard Wilton

Appendix O

IBM PC ROM BIOS Calls

To invoke an IBM PC BIOS routine, set register AH to the desired function and execute the software interrupt (INT) for the desired routine.

Graphics pixel coordinates and cursor row and column coordinates are always zero based.

Interrupt 10H: Video Services

Function 00H: Set Video Mode

To call:

AH	= 00H			
AL	= mode:			
	00H	16-shade gray text EGA: 64-color	40 by 25	B000:8000H
	01H	16/8-color text EGA: 64-color	40 by 25	B000:8000H
	02H	16-shade gray text EGA: 64-color	80 by 25	B000:8000H
	03H	16/8-color text EGA: 64-color	80 by 25	B000:8000H
	04H	4-color graphics	320 by 200	B000:8000H
	05H	4-shade gray graphics	320 by 200	B000:8000H
	06H	2-shade gray graphics	640 by 200	B000:8000H
	07H	monochrome text	80 by 25	B000:0000H
	08H	16-color graphics	160 by 200	B000:0000H
	09H	16-color graphics	320 by 200	B000:0000H
	0AH	4-color graphics	640 by 200	B000:0000H
	0BH	Reserved		
	0CH	Reserved		
	0DH	16-color graphics	320 by 200	A000:0000H
	0EH	16-color graphics	640 by 200	A000:0000H
	0FH	monochrome graphics	640 by 350	A000:0000H
	10H	16/64-color graphics	640 by 350	A000:0000H

Returns:

Nothing

Function 01H: Set Cursor Size and Shape

To call:

AH = 01H
CH = starting scan line
CL = ending scan line

Note: CH < CL gives normal one-part cursor; CH > CL gives two-part cursor; CH = 20H gives no cursor.

Returns:

Nothing

Function 02H: Set Cursor Position

To call:

AH = 02H
BH = display page (0 in graphics)
DH = row number
DL = line number

Returns:

Nothing

Function 03H: Read Cursor Position, Size, and Shape

To call:

AH = 03H
BH = display page

Returns:

CH = starting scan line
CL = ending scan line
DH = row number
DL = column number

Function 04H: Read Light-Pen Position

To call:

AH = 04H

Returns:

AH = status:
 01H pen triggered
 00H not triggered
 BX = pixel column number
 CH = pixel line number
 CX = pixel line number for some EGA modes
 DH = character row number
 DL = character column number

Function 05H: Select Active Page**To call:**

AH = 05H
 AL = page number:
 00-07H 40-column text modes
 00-03H 80-column text modes
 varies EGA graphics modes

Note: Each page = 2 KB in 40-column text mode, 4 KB in 80-column text mode.

Returns:

Nothing

Function 06H: Scroll Window Up**Function 07H: Scroll Window Down****To call:**

AH = 06H scroll up
 = 07H scroll down
 AL = number of lines to scroll (00H blanks screen)
 BH = display attributes for blank lines
 CH = row number of upper left corner
 CL = column number of upper left corner
 DH = row number of lower right corner
 DL = column number of lower right corner

Returns:

Nothing

Function 08H: Read Character and Attribute at Cursor**To call:**

AH = 08H
 BH = display page (for text mode only)

Returns:

If text mode:

AH = color attributes of character
AL = ASCII character from current location

If graphics mode:

AL = ASCII character (00H if unmatched)

Function 09H: Write Character and Attribute

To call:

AH = 09H
AL = ASCII character to write
BH = display page
BL = text attribute or graphics foreground color
CX = number of times to write character (must be > 0)

Returns:

Nothing

Note: Cursor position unchanged.

Function 0AH: Write Character Only

To call:

AH = 0AH
AL = ASCII character to write
BH = display page
BL = graphics foreground color (unused in text modes)
CX = number of times to write character (must be > 0)

Returns:

Nothing

Note: Cursor position unchanged.

Function 0BH: Select Color Palette

To call:

AH = 0BH
BH = palette color ID
BL = color or palette value

Returns:

Nothing

Function 0CH: Write Pixel Dot**To call:**

AH = 0CH
AL = color attribute of pixel
CX = pixel column number
DX = pixel raster line number

Returns:

Nothing

Function 0DH: Read Pixel Dot**To call:**

AH = 0DH
CX = pixel column number (0-based)
DX = pixel raster line number (0-based)

Returns:

AL = pixel color attribute

Function 0EH: Write Character as TTY**To call:**

AH = 0EH
AL = ASCII character
BH = display page
BL = foreground color of character (unused in text mode)

Returns:

Nothing

Note: Cursor position advanced; beep, backspace, linefeed, and carriage return active; all other characters displayed.

Function 0FH: Get Current Video Mode**To call:**

AH = 0FH

Returns:

AH = characters per line (20, 40, or 80)
AL = current video mode (see Interrupt 10H Function 00H)
BH = active display page

Function 13H: Write Character String

To call:

AH	= 13H
AL	= subfunction number:
	00H string shares attribute in BL, cursor unchanged
	01H string shares attribute in BL, cursor advanced
	02H each character has attribute, cursor unchanged
	03H each character has attribute, cursor advanced
BH	= active display page
BL	= string attribute (for AL = 00H or 01H only)
CX	= length of character string
DH	= starting row number
DL	= starting column number
ES:BP	= address of string to be displayed

Note: For AL = 00H or 01H, string = (*char, char, char, ...*). For AL = 02H or 03H, string = (*char, attr, char, attr, ...*).

Returns:

Nothing

Note: For AL = 01H or 03H, cursor position set to location following last character output.

Interrupt 11H: Get Peripheral Equipment List

Returns:

AX	= equipment list code word (bit settings PPMURRRUFFVVUUCI):
	PP number of printers installed
	M 1 if internal modem installed
	RRR number of RS-232 ports installed
	U unused
	FF number of floppy-disk drives minus 1 (0 = one drive)
	VV initial video mode:
	00 = reserved
	01 = 40-by-25 color
	10 = 80-by-25 color
	11 = 80-by-25 monochrome
	U unused
	C 1 if math coprocessor installed
	I 1 if IPL (Initial Program Load) diskette installed

Interrupt 12H: Get Usable Memory Size (KB)

Returns:

AX = available memory size in KB

Interrupt 13H: Disk Services

Function 00H: Reset Disk System

To call:

AH	= 00H	
AL	= drive number:	
	00-7FH	floppy disk
	80-FFH	fixed disk

Returns:

CF	= 0	no error
	1	error
AH	= error code	(see Interrupt 13H Function 01H)

Function 01H: Get Disk Status

To call:

AH = 01H

Returns:

AH	= 00H	
AL	= disk status of previous disk operation:	
	00H	no error
	01H	invalid command
	02H	address mark not found
	03H	write attempt on write-protected disk (F)
	04H	sector not found
	05H	reset failed (H)
	06H	floppy disk removed (F)
	07H	bad parameter table (H)
	08H	DMA overflow (F)
	09H	DMA crossed 64 KB boundary
	0AH	bad sector flag (H)
	10H	uncorrectable CRC or ECC data error
	11H	ECC corrected data error (H)
	20H	controller failed

(more)

40H	seek failed
80H	time out
AAH	drive not ready (H)
BBH	undefined error (H)
CCH	write fault (H)
E0H	status error (H)

Note: H = fixed disk only, F = floppy disk only.

Function 02H: Read Disk Sectors

Function 03H: Write Disk Sectors

Function 04H: Verify Disk Sectors

Function 05H: Format Disk Tracks

To call:

AH	= 02H	read disk sectors
	03H	write disk sectors
	04H	verify disk sectors
	05H	format disk track
AL	=	number of sectors
CH	=	cylinder number
CL	=	sector number (unused if AH = 05H)
DH	=	head number
DL	=	drive number
ES:BX	=	buffer address (unused if AH = 04H)

Returns:

CF	= 0	no error
	1	error
AH	=	error code (<i>see</i> Interrupt 13H Function 01H)

If AH was 05H on call:

ES:BX	=	4-byte address field entries, 1 per sector:
	byte 0	cylinder number
	byte 1	head number
	byte 2	sector number
	byte 3	sector-size code:
	00H	128 bytes per sector
	01H	256 bytes per sector
	02H	512 bytes per sector (standard)
	03H	1024 bytes per sector

Function 08H: Get Current Drive Parameters

To call:

AH	= 08H
DL	= drive number

Returns:

AX = 00H
BH = 00H
BL = drive type
CH = low-order 8 bits of 10-bit maximum number of cylinders
CL = bits 7 and 6 high-order 2 bits of 10-bit maximum number of cylinders
bits 5-0 maximum number of sectors/track
DH = maximum head number
DL = number of drives installed
ES:DI = address of floppy-disk-drive parameter table

Function 09H: Initialize Hard-Disk Parameter Table**To call:**

AH = 09H

Returns:

Nothing

Function 0AH: Read Long

Reads 512-byte sector plus 4-byte ECC code.

To call:

See Interrupt 13H Function 02H.

Returns:

See Interrupt 13H Function 02H.

Function 0BH: Write Long

Writes 512-byte sector plus 4-byte ECC code.

To call:

See Interrupt 13H Function 03H.

Returns:

See Interrupt 13H Function 03H.

Function 0CH: Seek to Head

Positions head but does not transfer data.

To call:

See Interrupt 13H Functions 02H and 03H.

Returns:

See Interrupt 13H Functions 02H and 03H.

Function 0DH: Alternate Disk Reset

To call:

AH = 0DH
DL = drive number

Returns:

Nothing

Function 10H: Test for Drive Ready

To call:

AH = 10H
DL = drive number

Returns:

AH = status

Function 11H: Recalibrate Drive

To call:

AH = 11H
DL = drive number

Returns:

AH = status

Function 14H: Controller Diagnostic

To call:

AH = 14H

Returns:

AH = status

Function 15H: Get Disk Type

To call:

AH = 15H
DL = drive number

Returns:

AH = drive type code:
00H no drive present
01H cannot sense when floppy disk is changed

(more)

02H can sense when floppy disk is changed
 03H fixed disk

If AH = 03H:

CX:DX = number of sectors

Function 16H: Check for Change of Floppy Disk Status

To call:

AH = 16H
 DL = drive number to check

Returns:

AH = 00H no change
 06H floppy-disk change

Function 17H: Set Disk Type

To call:

AH = 17H
 DL = drive number
 AL = floppy-disk type code

Returns:

Nothing

Interrupt 14H: Serial Port Services

Function 00H: Initialize Port Parameters

To call:

AH = 00H
 AL = serial port parameters (bit settings BBBPPSCC):
 BBB baud rate:
 000 110 baud
 001 150 baud
 010 300 baud
 011 600 baud
 100 1200 baud
 101 2400 baud
 110 4800 baud
 111 9600 baud

(more)

PP	parity code:
	00 none
	01 odd
	10 none
	11 even
S	number of stop bits code:
	0 one stop bit
	1 two stop bits
CC	character size:
	00 unused
	01 unused
	10 7-bit character size
	11 8-bit character size
DX	= serial port number (0 = first port)

Returns:

Nothing

Function 01H: Send One Character

To call:

AH	= 01H
AL	= character to send
DX	= serial port number (0 = first port)

Returns:

AH	= error status (<i>see</i> Interrupt 14H Function 03H):
	00H no error

Function 02H: Receive One Character

To call:

AH	= 02H
DX	= serial port number (0 = first port)

Returns:

AL	= character received
AH	= error status (<i>see</i> Interrupt 14H Function 03H):
	00H no error

Function 03H: Get Port Status

To call:

AH	= 03H
DX	= serial port number (0 = first port)

Returns:

AX	= serial port status:
8000H	time out
4000H	transfer shift register empty
2000H	transfer holding register empty
1000H	break detect
0800H	framing error
0400H	parity error
0200H	overrun error
0100H	data ready
0080H	received line signal detect
0040H	ring indicator
0020H	data set ready
0010H	clear to send
0008H	delta receive line signal detect
0004H	trailing edge ring detector
0002H	delta data set ready
0001H	delta clear to send

Note: Multiple conditions can be active simultaneously.

Interrupt 15H: Miscellaneous System Services

Function 00H: Turn On Cassette Motor

Function 01H: Turn Off Cassette Motor

To call:

AH	= 00H	turn on cassette motor
	01H	turn off cassette motor

Returns:

Nothing

Function 02H: Read Data from Cassette

To call:

AH	= 02H
CX	= number of bytes to read
ES:BX	= buffer address

Returns:

CF = 0 no error
1 error
AH = error status (if needed):
01H CRC error
02H bit signals scrambled
03H no data found
DX = number of bytes read
ES:BX = location following last byte read

Function 03H: Write Data to Cassette

To call:

AH = 03H
CX = number of bytes to write
ES:BX = buffer address

Note: Blocking factor = 256 bytes/block.

Returns:

CX = 00H
ES:BX = location following last byte written

Interrupt 16H: Keyboard Services

Function 00H: Read Next Character

To call:

AH = 00H

Returns:

If ASCII characters:

AH = standard PC keyboard scan code
AL = ASCII character

If extended ASCII codes:

AH = extended ASCII code
AL = 00H

Note: Does not return until character is read; removes character from keyboard buffer.

Function 01H: Report If Character Ready**To call:**

AH = 01H

Returns:

ZF = 0 character ready
 1 character not ready
 AH = *see* Interrupt 16H Function 00H
 AL = *see* Interrupt 16H Function 00H

Note: Returns immediately; does not remove character from keyboard buffer.

Function 02H: Get Shift Status**To call:**

AH = 02H

Returns:

AL = shift status:
 01H right shift active
 02H left shift active
 04H Ctrl active
 08H Alt active
 10H Scroll Lock active
 20H Num Lock active
 40H Caps Lock active
 80H insert state active

Note: Multiple states can be active simultaneously.

Interrupt 17H: Printer Services**Function 00H: Send Byte to Printer****To call:**

AH = 00H
 AL = character to be printed
 DX = printer number

Returns:

AH = status (*see* Interrupt 17H Function 02H)

Function 01H: Initialize Printer

To call:

AH = 01H
DX = printer number

Returns:

AH = status (*see* Interrupt 17H Function 02H)

Function 02H: Get Printer Status

To call:

AH = 02H
DX = printer number

Returns:

AH = status:

01H	time out
02H	unused
04H	unused
08H	I/O error
10H	printer selected
20H	out of paper
40H	printer acknowledgment
80H	printer not busy (bit off, 0, = busy)

Note: Multiple states can be active simultaneously.

Interrupt 18H: Transfer Control to ROM-BASIC

Interrupt 19H: Reboot Computer (Warm Start)

Interrupt 1AH: Get/Set Time/Date

Function 00H: Read Current Clock Count

To call:

AH = 00H

Returns:

AL = midnight signal
CX = high-order word of tick count
DX = low-order word of tick count

Function 01H: Set Current Clock Count**To call:**

AH = 01H
CX = high-order word of tick count
DX = low-order word of tick count

Returns:

Nothing

Function 02H: Read Real-Time Clock**To call:**

AH = 02H

Returns:

CF = 0 clock running
 1 clock stopped
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD

Function 03H: Set Real-Time Clock**To call:**

AH = 03H
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = 00H standard time
 01H daylight saving time

Returns:

Nothing

Function 04H: Read Date from Real-Time Clock**To call:**

AH = 04H

Returns:

CF = 0 clock running
 1 clock stopped
CH = century in BCD (19 or 20)
CL = year in BCD
DH = month in BCD
DL = day in BCD

Function 05H: Set Date in Real-Time Clock

To call:

AH = 05H
CH = century in BCD (19 or 20)
CL = year in BCD
DH = month in BCD
DL = day in BCD

Returns:

Nothing

Function 06H: Set Alarm

To call:

AH = 06H
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD

Returns:

CF = status:
 0 operation successful
 1 alarm already set or clock stopped

Function 07H: Reset Alarm (Turn Alarm Off)

To call:

AH = 07H

Returns:

Nothing

Indexes



Subject

Symbols and Numerals

! (exclamation point)
SYMDEB 1154-55

(number sign). *See also* EDLIN commands
CREF 967

•(asterisk)
EDLIN 829, 832
SYMDEB 1156
wildcard 813

- (hyphen)
DEBUG prompt 1020-21, 1046
SYMDEB prompt 1055

. (period). *See also* EDLIN commands
SYMDEB 1151

. and .. (directory aliases) 103, 282, 283

/ (slash)
directories 280, 284
SYMDEB 1150

: (colon)
EDLIN 832
hexadecimal object file format 1499
SYMDEB 1059

;(semicolon), APPEND 739

<, >, and >> (redirection symbols) 67, 753
ECHO 759
filters and 430
PAUSE 766
REM 768
SYMDEB 1143-45

= (equal sign), SYMDEB 1146

? (question mark)
PROMPT 904, 905
SYMDEB 1152-53

@ (at sign) 1434

\ (backslash)
directories 284

{ } (braces), SYMDEB 1147-48

;(piping character) 67, 753
ECHO 759
REM 768

~ (tilde), SYMDEB 1149

86-DOS operating system 12-13, 27
as basis for MS-DOS 15-19

4004. *See* Intel 4004 chip

8008. *See* Intel 8008 chip

8080. *See* Intel 8080 chip

8086. *See* Intel 8086 chip

8250. *See* INS8250 Universal Asynchronous Receiver Transmitter (UART)

8259. *See* Intel 8259A Programmable Interrupt Controller (PIC)

80186. *See* Intel 80186 chip

80188. *See* Intel 80188 chip

80286. *See* Intel 80286 chip

80386. *See* Intel 80386 chip

A

Absolute Disk Read. *See* Interrupt 25H

Absolute Disk Write. *See* Interrupt 26H

Address, defined 1058

Advanced run length limited (ARLL) encoding 87

align type parameters 125-27

Allen, Paul 8(fig.), 16(fig.)
in the development of early BASIC 3-8
in the development of MS-DOS 14-15, 30, 34

Allocate Memory Block. *See* Interrupt 21H
Function 48H

Alphabetic Sort Filter (SORT) 935-37

Altair computer, and BASIC language 3-8

Alternate Disk Reset. *See* Interrupt 13H Function 0DH

ANSI Console Driver. *See* ANSI.SYS

ANSI.SYS 152, 731-38
AUTOEXEC.BAT and 755
controlling the screen with 158-59
key and extended key codes 1471-72

APPEND command 739-40
MS-DOS version 3.3 1436-37

Append Lines from Disk (EDLIN A) 834

Application programs
structure of 107-48
.COM programs 142-48
.EXE programs 107-42
as transient 447
writing for upward compatibility 489-97
hardware issues 489-92
operating-system issues 492-97

Applications Program Interface. *See* Family API

Arithmetic, hexadecimal 1035

ASCII format 872
character set 1465-67
cross-reference listing 967
display content of memory in 1077-78
display lookup table 629-40
entering strings 1093-96
escape sequences 731

- ASCII format (*continued*)
 make files, and MAKE utility 999–1003
 strings with environmental variables 930
 text files 752, 788, 829, 935, 947
- ASCIIZ strings 65
- ASCTBL.C program 545
 correct code 639(fig.)
 correction of 631–39
 expected output 630(fig.)
 incorrect code 630–31
- Assemble Machine Instructions
 DEBUG A 1024–25
 SYMDEB A 1063–64
- Assembly-language programs
 acceptance/translation of 1024, 1063
 active TSR (video buffer dump) 360–80
 block-device driver 478–85
 character-device driver 471–77
 character-oriented filter 431–33
 communications device driver 182–200
 communications port monitor 558–63
 disassembling machine instructions into
 1051, 1132
 filter as child process 442–46
 handler for UART interrupts 216–21
 line-oriented filter 434–35
 lowercase filter 437–39
 message program 651
 modem engine 207–8
 MS-DOS shell substitute 81–83
 parent and child examples 329–34
 passive TSR (pop-up) 357–59
 replacement Interrupt 00H handler 420–24
 replacement Interrupt 24H handler 395–98
 root and overlay examples 337–42
 support files for terminal emulator 223–30
 symbol cross-referencing in, with CREF 967
 test program for communications port monitor
 580–81
 translation into relocatable object module (*see*
 Microsoft Macro Assembler)
 volume label updating program 292–96
- ASSIGN command 741–42
 APPEND and 739
 BACKUP and 747
 CHKDSK and 775
 DISKCOMP and 818
 DISKCOPY and 822
 JOIN and 877
 LABEL and 882
 MKDIR/MD 885
- Assign Drive Alias (ASSIGN) 741–42
 Assign Standard Input/Output Device (CTTY) 810
 Asynchronous, defined 171–72
- AT address parameter 128
 AT Probe hardware debugging aid 641
 ATTRIB command 743–44
 MS-DOS version 3.3 1437
- AUTOEXEC.BAT file (BATCH) 755–57
 environments and 65
 MODE and 887
 VER and 952
- AUX (auxiliary input/output) 22, 59, 62, 151. *See also*
 COM1; Serial communications ports
 filters and 429
 implementing modem engine with MS-DOS
 functions 168–70
 I/O 161–62
 opening 76
- Auxiliary Input. *See* Interrupt 21H Function 03H
 Auxiliary Output. *See* Interrupt 21H Function 04H
- ## B
- Background program 900
- BACKUP command 745–51
 ASSIGN and 741
 ATTRIB and 743
 JOIN and 877
 MS-DOS version 3.3 1437
 RESTORE and 918
- Back Up Files (BACKUP) 745–51
- BACKUPID.@@@ control file 746–47
- BADSCOP.ASM program 544
 correction of 593–600
 incorrect version of 587–93
- BASIC (language), role of, in development of
 MS-DOS 3–8, 12, 14
- Batch file(s) 26
 AUTOEXEC.BAT 755–57
 COMMAND.COM and 64, 66–67, 78, 753, 755
 directives 730, 752–69, 1434
 @ command 1434
 CALL command 1434–35
 ECHO command 758–59
 FOR command 760–61
 GOTO command 762–63
 IF command 764–65
 PAUSE command 766–77
 REM command 768
 SHIFT command 769
 executing commands stored in 752
 MS-DOS version 3.3 1434–35
 suspend execution of 766
- .BAT file. *See* Batch file(s)
- Baud rate 170, 222, 892

- BDOS (Basic Disk Operating System), CP/M 10
 - Bebic, Mark 39
 - Binary operators, SYMDEB 1059
 - Binary-to-hexadecimal file conversion utility
 - program 1503-5
 - BIOS (Basic Input/Output System)
 - CP/M 10
 - MS-DOS 52-53, 61-62
 - ROM 62 (*see also* Interrupts 10H through 1AH)
 - BIOS parameter block (BPB) 70, 71(fig.), 93
 - build function, in device drivers 459-60
 - format 460(table)
 - Bit bucket. *See* NUL device
 - Bit parity 222
 - Bit rate divisor table for 8250 IBM UART chip
 - 175(table)
 - Bits per second (bps) 170
 - Block device(s) 57, 62. *See also* Fixed disk; Floppy disk; RAMdisk
 - critical error handling 392-93
 - drivers 450-52
 - file system and 54-55
 - layout of a physical 86-90
 - partition layout 90-92
 - setting highest logical 803
 - setting parameters 797-98
 - Bootable devices, loading 70, 71(fig.)
 - Boot sector 94-96
 - hexadecimal dump of 96(fig.)
 - map of 95(fig.)
 - Bootstrapping, operating system 52, 68-72
 - BOUND Range Exceeded exception. *See* Interrupt 05H
 - BREAK command 770-71
 - BREAK command (CONFIG.SYS) 788, 790
 - BREAK condition 172
 - Breakpoints 1033
 - clearing 1065-66
 - DEBUG use of 578-79, 584-85
 - disabling 1067-68
 - enabling 1069-70
 - hardware 640, 641
 - listing 1071
 - setting 1072-73
 - SYMDEB use of 608-9
 - trapping 400
 - Breakpoint Trap exception. *See* Interrupt 03H
 - Brock, Rod 12, 15
 - Buffered Keyboard Input. *See* Interrupt 21H
 - Function 0AH
 - BUFFERS command (CONFIG.SYS) 788, 791
 - Byte(s)
 - displaying 1079-80
 - Byte(s) (*continued*)
 - entering 1095-96
 - BYTE alignment 125-26
- C**
- CALL command (BATCH) 1434-35
 - Calls menu (CodeView) 1162
 - Cancel Assign-List Entry 1411-12
 - Cassette/Network Service. *See* Interrupt 15H
 - CAV (constant angular velocity) disks 87
 - C Compiler, Microsoft
 - environmental variables in 931, 980
 - general structure of C program 139(fig.)
 - memory model use with 137-40
 - utilities supplied with 974, 977, 987, 999
 - CCP (Console Command Processor), CP/M 10
 - CD command. *See* CHDIR/CD command
 - CD ROM storage 103
 - CDVUTL.C communications driver-status
 - utility 209-15
 - code 209-14
 - program functions 214(table)
 - Central processing unit (CPU), speed of, and compatibility issues 491
 - CH1.ASM program 215-22
 - exception handler module 223-24
 - module functions 221(table)
 - set_mdm()* parameter coding 222(table)
 - CH2.ASM program 225-30
 - Change Code Page (CHCP) 1440
 - Change Current Directory. *See* Interrupt 21H
 - Function 3BH
 - Change Current Directory (CHDIR or CD) 772-73
 - Change File Attributes (ATTRIB) 743-44
 - Change Filename (RENAME or REN) 912-13
 - Change Sharing Retry Count 1337-38
 - Character-device input/output 149-66. *See also* Display output; Graphics; Input/output (I/O); Parallel port; Printer; Screen; Serial communications ports
 - accessing character devices 150-51
 - background information on 149-50
 - basic MS-DOS devices 151
 - display 157-61
 - keyboard 154-57
 - parallel port and printer 163-64
 - raw versus cooked mode 153-54
 - serial communications ports 161-62
 - standard devices 152-53

Character-device input/output (*continued*)
 basic MS-DOS devices (*continued*)
 standard devices as support for filters
 429–30
 copying files 806–9
 critical error handling 393
 defined keyboard 879
 device drivers 448–50
 IOCTL subfunctions 164–66
 screen dump in graphics mode to printer
 874–76
 specify for standard input/output 810
 system calls for 1182

Character-device management commands 728
 CLS 781
 CTTY 810
 GRAFTABL 872–73
 KEYB:xx 879–81
 MODE 887–95
 PRINT 899–903

Character Input with Echo. *See* Interrupt 21H
 Function 01H

Character Input Without Echo. *See* Interrupt 21H
 Function 08H

Character Output. *See* Interrupt 21H Function 02H

Character string, finding 863–64

CHCP command 1440

CHDIR/CD command 281, 772–73

Check Disk Status (CHKDSK) 774–80

Check for Change of Floppy Disk Status. *See*
 Interrupt 13H Function 16H

Check If Block Device Is Remote. *See* Interrupt 21H
 Function 44H Subfunction 09H

Check If Block Device Is Removable. *See* Interrupt
 21H Function 44H Subfunction 08H

Check If Handle Is Remote. *See* Interrupt 21H
 Function 44H Subfunction 0AH

Check Input Status. *See* Interrupt 21H Function 44H
 Subfunction 06H

Check Keyboard Status. *See* Interrupt 21H
 Function 0BH

Check Output Status. *See* Interrupt 21H Function 44H
 Subfunction 07H

CHILD.ASM program 334–35

Child program(s)
 filters used as 441–46
 using EXEC to load/run 321
 examining return codes 328
 parent and child program example 329–35
 preparing parameters for 323–26
 running child programs 327

CHKDSK command 101, 774–80, 941

C language programs
 ASCII lookup program 639

C language programs (*continued*)
 attribute listing program 291–92
 character-oriented filter 433
 control program for communications port
 monitor 565–66
 debugging with SYMDEB 600–618
 demonstration Windows program 513–15
 driver-status utility 209–14
 line-oriented filter 436
 lowercase filter 438–39
 new FIND filter program 439–41
 object module dump utility 1509–12
 terminal emulator 230–41

class type parameters 128–30

Clear Breakpoints (SYMDEB BC) 1065–66

Clear Screen (CLS) 781

Clipboard (Windows) 537–38

Clock
 setting date 811
 setting system time 942

CLOCK\$ 57, 59, 62, 151

Closed-loop servomechanism 89

Close File. *See* Interrupt 21H Function 3EH

Close File with FCB. *See* Interrupt 21H Function 10H

CLPBRD utility (Windows) 506

CLS command 781

Clusters, file data 94

CLV (constant linear velocity) disks 87

Cmacros 1178–81

CMACROX.INC 1179–81

COBOL (language) 14

Code-page switching 1438–48, 1451–58

CodeView utility 573, 619–40, 1157–73
 description 1158–59
 dialog window commands 1163–65
 display window commands 1159–62
 Calls menu 1162
 File menu 1159
 Help menu 1162
 Language menu 1161
 Options menu 1161
 Run menu 1160
 Search menu 1160
 View menu 1160
 Watch menu 1161
 instrumentation debugging with 619–29
 key commands 1163
 messages 1166–73
 screen 1159(fig.)
 screen output debugging with 629–40

Cold boot 68

Color capabilities, of display 733

Color/Graphics Adapter (CGA) 157

- COM1 (first serial communications port) 151, 161–62
- COM2 (second serial communications port) 151, 161–62
- combine* type parameters 127–28
- COMDEF Communal Names object record 651, 698–700
- COMDVR. ASM communications device driver 182–206
 - buffering 203
 - code 182–200
 - debugging techniques 205–6
 - definitions 200–201
 - headers and structure tables 201
 - Initialization Request routine 204–5
 - interrupt service routine 203–4
 - Start_output* routine 204
 - strategy and request routines 180
 - using 205
- COMMENT Comment object record 651, 658–60
- Command(s) 725–30. *See individual command names*
 - defining command search path 897
 - execution of, with COMMAND.COM 64–65
 - by functional group 728–30
 - internal, external, and batch 76–79
 - interpreting text file of, with MAKE 999
 - PC-DOS, added to MS-DOS version 3.3 1435–36
- COMMAND.COM 20, 63–68, 782–84
 - batch files and 64, 66–67, 78, 753, 755–56
 - command execution with 64–65
 - define prompt 904
 - escape to 1154–55
 - EXEC use with 329–30
 - I/O redirection in 67–68
 - loading 76–79
 - MS-DOS environments and 65–66
 - parts of 76
 - specifying/replacing, with SHELL 79–83, 804
 - split personality of 64
 - SYS and 940
 - terminating 853
 - transient/resident portions of 24
- COMMAND command 782–84. *See also* COMMAND.COM
- Command processor. *See* COMMAND.COM; SHELL command
- Command Processor (COMMAND) 782–84
- Command tail
 - in child program execution 327
 - DEBUG initializing of 582–83
 - FCB functions and 267–68
 - name parameters 1040, 1116
- COMMDDUMP.BAS program 543–44, 569–72
- Comment line
 - including with REM 768
 - in make files 1001
 - SYMDEB 1156
- Commit File 1450–51
- COMMON parameter 128
- COMMSCMD.BAS program 543, 567–69
- COMMSCMD.C program 543
 - as a .COD file for SYMDEB debugging 601–6
 - correction of 606–18
 - stopping a trace in 565–66
- COMMSCOP.ASM program 542–43, 558–63
- Communications, interrupt-driven 167–246, 412
 - device driver 180
 - hardware for 170–80
 - 8250 UART architecture 172–80
 - modem 170–71
 - serial port 171–72
 - memory-resident device driver 182–215
 - COMDRV. ASM 182–206
 - driver-status utility CDVUTL.C 209–15
 - modem engine 206–9
 - vs* traditional method 181
 - program, purpose of 167–68
 - traditional device driver 215–46
 - exception handler module 223–25
 - hardware ISR module 215–22
 - smart terminal emulator CTERM.C 230–46
 - video display module 225–30
 - using simple MS-DOS functions 168–70
- Compact memory model 138
- COMPAQ-DOS operating system 27
- Compare Files (COMP) 785–87
- Compare Files (FC) 854–57
- Compare Floppy Disks (DISKCOMP) 818–21
- Compare Memory Areas
 - DEBUG C 1026
 - SYMDEB C 1074
- Compatibility issues
 - 8086/8088 and 80286 1507–8
 - MS-DOS and MS OS/2 489–97
 - hardware 489–92
 - operating system 492–97
- COMP command 785–87
 - MS-DOS version 3.3 1435
- Compress .EXE File (EXEPACK) 977–79
- .COM program files 23, 64, 142–47, 974
 - converting .EXE programs to executable
 - 971–72
 - creating 144–46
 - vs* .EXE programs 147–48
 - giving control to 143

- .COM program files (*continued*)
 - memory allocated for 142, 300–302
 - memory map with register pointers 143(fig.)
 - patching using DEBUG 146
 - terminating 144
- COMSPEC variable 930
- CON (console input/output) 22, 59, 62, 151, 157. *See also* Display output; Screen
 - batch commands for 66–67
 - filter and 429
 - opening 76
- Conditional execution, using IF to perform 764–65
- CONFIG.SYS system configuration 63, 448, 788–89
 - configuring Control-C checking 790
 - configuring internal disk buffers 791–92
 - configuring internal stacks 805
 - environments and 65
 - installing device drivers 149, 795–96
 - setting block-device parameters 797–98
 - setting country code 793–94
 - setting highest logical drive 803
 - setting maximum open files with FCBs 799–800
 - setting maximum open files with handles 801–2
 - specifying command processor 804
- Configurable External-Disk-Drive Driver (DRIVER.SYS) 826–28
- Configure Control-C Checking (BREAK) 790
- Configure Device (MODE) 887
- Configure Fixed Disk (FDISK) 858–62
- Configure Internal Disk Buffers (BUFFERS) 791
- Configure Internal Stacks (STACKS) 805
- Configure Printer (MODE) 888–89
- Configure Serial Port (MODE) 892–93
- Configure System Disk for a Specific Country (SELECT) 925–29
- Console. *See* Keyboard; Screen
- Control-Break, exception handling 385, 386, 387, 389
- Control-Break (user defined). *See* Interrupt 1BH
- Control-C
 - configuring check 790
 - setting check 770
- Control-C exception handler 385, 386–89
 - customizing 387–89
 - processing Control-C 389
- Control-C Handler Address. *See* Interrupt 23H
- Controller Diagnostics. *See* Interrupt 13H
 - Function 14H
- CONTROL Panel (Windows) 507
- Control-Z in EDLIN commands 846
- Conventional memory 297–305, 907
 - block move from extended memory to 318–19
 - functions to support 299(table)
 - using functions in 300–305
- Convert .EXE File to Binary-Image File (EXE2BIN) 971–73
- Cooked versus raw mode 153–54
- Coprocessor Error exception. *See* Interrupt 10H
- Coprocessor Not Available exception. *See* Interrupt 07H
- Coprocessor Segment Overrun exception. *See* Interrupt 09H
- COPY command 806–9
 - ASSIGN and 741
 - batch files and 752
 - DISKCOPY and 822
 - escape sequences using 732
- Copy File or Device (COPY) 806–9
- Copy Files (XCOPY) 955–59
- Copy Floppy Disk (DISKCOPY) 822
- Copy Lines (EDLIN C) 835–36
- Country, configure disk for a specific 925–29
- COUNTRY command (CONFIG.SYS) 788, 793–94
 - BACKUP and 747
 - development of 36
 - MS-DOS version 3.3 1442–43
 - setting date 812
 - setting time 942
- CP/M operating system 8, 9–10, 56, 142
 - compatibility with 63
 - competition with MS-DOS 27–29
 - file management 30–31
- Create Directory. *See* Interrupt 21H Function 39H
- Create .EXE File (LINK) 987–98
- Create File with FCB. *See* Interrupt 21H Function 16H
- Create File with Handle. *See* Interrupt 21H
 - Function 3CH
- Create New File. *See* Interrupt 21H
 - Function 5BH
- Create New Program Segment Prefix. *See* Interrupt 21H Function 26H
- Create Symbol File for SYMDEB (MAPSYM) 1004–6
- Create Temporary File. *See* Interrupt 21H
 - Function 5AH
- CREF utility 967–70
- Critical error handler 390–98
 - customized 394–98
 - mechanics of 392–93
 - processing 393–94
 - in TSR programs 353–55
- Critical Error Handler Address. *See* Interrupt 24H
- CTERM.C terminal emulator program 230–46
 - functions 242–43(table)
 - prototype file CTERM.H 243–44(fig.)
- Ctrl-Break. *See* Control-Break
- Ctrl-C. *See* Control-C
- Ctrl-Z. *See* Control-Z in EDLIN commands

CTTY command 810
 Cursor movement, escape sequences to
 control 732-33
 Cylinder, disk 88

D

Data
 entering into memory 1029, 1091
 moving (copying) 1039, 1115
 sharing/exchange in Windows 537-38
 Data area, DEBUG initializing 582
 Data files, setting a search path for. *See* APPEND
 command
 DATE command 811-12
 Debugging in MS-DOS 541-642
 art of 546
 communications device driver 205-6
 hardware debugging aids 640-42
 inspection and observation 546-49
 instrumentation
 external 555-72
 internal 549-55
 software debugging monitors 573-640
 CodeView 573, 619-40 (*see also* CodeView
 utility)
 DEBUG 573, 574-86 (*see also* DEBUG
 utility)
 SYMDEB 573, 586-618 (*see also* SYMDEB
 utility)
 summary of example programs to illustrate
 541-45
 DEBUG utility 113, 573, 574-86, 1020-53
 A command 141, 577, 1021, 1024-25
 basic techniques 574-81
 breakpoints 578-79, 584-85
 C command 1021, 1026
 D command 1021, 1027-28
 E command 141, 1021, 1029-30
 establishing initial conditions 581-83
 F command 1021, 1031-32
 G command 577, 584-85, 1021, 1033-34
 H command 1021, 1035
 I command 1021, 1036
 L command 1021, 1037-38
 M command 577, 1021, 1039
 N command 1021, 1040-41, 1052
 O command 1021, 1042
 patching .COM programs with 146
 patching .EXE programs with 585-86, 141-42
 P command 580, 1021, 1043

DEBUG utility (*continued*)
 Q command 142, 1021, 1044
 R command 142, 576, 1021, 1045-47
 S command 1021, 1048-49
 T command 576, 1021, 1050
 U command 577, 1021, 1051
 using Write commands 585-86
 W command 141, 577, 585-86, 1021, 1052-53
 Define Command Search Path (PATH) 897-98
 Define Keyboard (KEYB:xx) 879-81
 Define System Prompt (PROMPT) 904-6
 DEL/ERASE command 813-14
 Delete File. *See* Interrupt 21H Function 13H; Interrupt
 21H Function 41H
 Delete File (DEL or ERASE) 813-14
 Delete Lines (EDLIN D) 837-38
 Desk-checking 547
 Development of MS-DOS 3-45
 before MS-DOS 3-15
 creating MS-DOS 15-19
 future of MS-DOS 45
 hardware and 27-28
 international market and 35-37
 software and 38
 versions 1.x 20-29
 versions 2.x 30-38
 versions 3.x 39-44
 DEVICE command (CONFIG.SYS) 149-50, 788,
 795-96
 MS-DOS version 3.3 1443-45
 Device driver(s) 52-53, 57
 Device driver(s), installable 180, 447-86. *See also*
 ANSI.SYS; Block device(s); Character-
 device input/output; RAMDRIVE.SYS;
 VDISK.SYS
 development of, in MS-DOS version 2.0
 32-33
 loading/initializing 74, 75(fig.)
 processing of a typical I/O request 468-69
 relationship to resident 448-50
 structure of 450-68
 device header 450-52
 interrupt routine 453-68
 strategy routine 452-53
 writing 469-86
 TEMPLATE example 471-78
 TINYDISK example 478-86
 Device driver, installable communications package
 180, 182-215
 memory-resident generic
 CDVUTL.C utility 209-15
 COMDVR.ASM device driver 182-206
 modem engine 206-9

- Device driver, installable communications package
 - (*continued*)
 - memory-resident generic (*continued*)
 - vs traditional method 181
 - traditional 215–46
 - exception-handler module 223–25
 - hardware ISR module 215–22
 - terminal emulator CTERM.C 230–46
 - video display module 225–30
- Device driver(s), resident 62
 - relationship to installable device drivers 448–50
- Device header 450–52
 - device attribute word in 452(table)
- DGROUP 718–21
- Dialog boxes (Windows) 504–5
- Dialog window commands (CodeView) 1163–65
- Digital Equipment Corporation (DEC) 28
- Digital Research, development of CP/M 9–10, 12, 28
- DIR.ASM program 288–90
- DIR command 815–17
- DIRDUMP.C program 291–92
- Direct Console I/O. *See* Interrupt 21H Function 06H
- Direct memory access. *See* DMA (direct memory access) controller
- Directory 101–3, 279–96. *See also* Subdirectory; Volume label(s)
 - alias 103, 282, 283
 - analyzing for errors 774
 - attribute field 282(fig.)
 - changing current 772
 - copying 955
 - current 281, 288
 - date/time fields 283(fig.)
 - displaying 815
 - displaying structure 944
 - format 281–83
 - functional support for 284–96
 - creating/deleting 287
 - examining/modifying 287
 - MS-DOS functions for accessing 284–86(table)
 - programming examples 288–92
 - searching 286
 - specifying current 288
 - wildcard characters 286–87
 - hexadecimal dump of 102(fig.)
 - initializing 865
 - joining to disk 877
 - making 885
 - removing 923
 - root (*see* Root directory)
 - structure 32, 54, 279(fig.), 280–81
- Directory (*continued*)
 - system calls for 1183
- Directory management commands 729
 - APPEND 739–40
 - CHDIR/CD 772–73
 - MKDIR/MD 885–86
 - PATH 897–98
 - RMDIR/RD 923–24
 - TREE 944–46
- Disable Breakpoints (SYMDEB BD) 1067–68
- Disable Source Display Mode (SYMDEB S-) 1128
- Disassemble (Unassemble) Program
 - DEBUG U 1051
 - SYMDEB U 1132–33
- Disk
 - checking status of 774
 - configuring for a specific country 925
 - configuring internal buffer 791
 - directories (*see* Directory)
 - displaying volume label 944–45
 - fixed (*see* Fixed disk)
 - floppy (*see* Floppy disk)
 - initialize 865
 - joining to directory 877
 - name (*see* Volume label[s])
 - recovering files from damaged 910
 - structure of 85–103
 - virtual 907, 948
 - writing file/sectors to 1052
- Disk cache, configure 791
- Disk Parameter Pointer. *See* Interrupt 1EH
- DISKCOMP command 818–21
 - ASSIGN and 741
 - JOIN and 877
- DISKCOPY command 822–25
 - ASSIGN and 741
 - JOIN and 877
- Disk management commands 729
 - ASSIGN 741–42
 - DISKCOMP 818–21
 - DISKCOPY 822–25
 - FORMAT 865–71
 - LABEL 882–84
 - SUBST 938–39
 - SYS 940–41
 - VERIFY 953
 - VOL 954
- Disk management system calls 1182
- Disk Reset 1213–14
- Disk Services. *See* Interrupt 13H
- Disk transfer area (DTA)
 - default 267–68

- Disk transfer area (*continued*)
 - getting address (*see* Interrupt 21H Function 2FH)
 - setting address (*see* Interrupt 21H Function 1AH)
 - TSR programs 353
 - Display 10-Byte Reals (SYMDEB DT) 1087-88
 - Display ASCII (SYMDEB DA) 1077-78
 - Display by Screenful (MORE) 896
 - Display Bytes (SYMDEB DB) 1079-80
 - Display Directory (DIR) 815-17
 - Display Directory Structure (TREE) 944-46
 - Display Disk Name (VOL) 954
 - Display Doublewords (SYMDEB DD) 1081-82
 - Display File (TYPE) 947
 - Display in Pages (EDLIN P) 844
 - Display Long Reals (SYMDEB DL) 1083-84
 - Display Memory
 - DEBUG D 1027-28
 - SYMDEB D 1075-76
 - Display Memory Areas 1075-76
 - Display or Modify Registers
 - DEBUG R 1045-47
 - SYMDEB R 1122-24
 - Display output 157-60. *See also* Character-device input/output; CON; Screen
 - of batch-file execution 758
 - CH2.ASM communications module 225-30
 - color capability of 733
 - controlling the screen 158-59
 - cursor movement control 732-33
 - debugging with CodeView 629-40
 - erasing 733
 - graphics attributes 734
 - in pages 844
 - programming examples 160
 - role of ROM BIOS in 159
 - by screenful 896
 - setting mode 890-91
 - width 733
 - wrap around 733
 - Display Short Reals (SYMDEB DS) 1085-86
 - Display Source Line (SYMDEB .) 1151
 - Display String. *See* Interrupt 21H Function 09H
 - Display Text (ECHO) 758
 - Display Version (VER) 952
 - Display window commands (CodeView) 1159-62
 - Display Words (SYMDEB DW) 1089-90
 - Divide by Zero exception. *See* Interrupt 00H
 - DIVZERO.ASM program 419, 420-24
 - DMA (direct memory access) controller 69
 - /DOSSEG switch, LINK use of 718-19
 - Double-Fault Exception. *See* Interrupt 08H
 - Doublewords
 - displaying 1081
 - entering 1097
 - Drive(s)
 - assigning aliases 741-42
 - substituting for subdirectory 938
 - DRIVER.SYS 826-28
 - DRIVPARM command (CONFIG.SYS) 788, 797-98
 - /DSALLOCATE switch, LINK use of 719-21
 - Dump. *See* Display Memory
 - Duplicate File Handle. *See* Interrupt 21H Function 45H
 - Dynamic Data Exchange (DDE) 538
- E**
- EBCDIC character set 1469-70
 - ECHO command (BATCH) 66, 753; 758-59
 - and PAUSE 766
 - Edit Line (EDLIN *linenumber*) 832-33
 - EDLIN commands 730, 829-52
 - A command 834
 - C command 835-36
 - D command 837-38
 - E command 839
 - escape character in 732
 - I command 840
 - L command 841
 - linenumber* command 832-33
 - M command 842-43
 - P command 844
 - Q command 845
 - R command 846-47
 - S command 848-49
 - T command 850-51
 - W command 852
 - Enable Breakpoints (SYMDEB BE) 1069-70
 - Enable Source and Machine Code Display Mode (SYMDEB S&) 1129
 - Enable Source Display Mode (SYMDEB S+) 1127
 - End Editing Session (EDLIN E) 839
 - ENGINE.ASM program 207-8
 - Enhanced Graphics Adapter (EGA) 157
 - MS-DOS version 3.3 code-page switching 1447
 - Enter 10-Byte Reals (SYMDEB ET) 1102-3
 - Enter ASCII String (SYMDEB EA) 1093-94
 - Enter Bytes (SYMDEB EB) 1095-96
 - Enter Comment (SYMDEB *) 1156
 - Enter Data
 - DEBUG E 1029-30
 - SYMDEB E 1091-92

- Enter Doublewords (SYMDEB ED) 1097
- Enter Long Reals (SYMDEB EL) 1098–99
- Enter Short Reals (SYMDEB ES) 1100–1101
- Enter Words (SYMDEB EW) 1104
- Environment(s)
 - in child program execution 326–27
 - MS-DOS operating 51–52, 65–66
- Environment variable, set 930
- Equipment Information. *See* Interrupt 11H
- ERASE. *See* DEL/ERASE command
- Error codes
 - device-driver 454(table)
 - extended, in MS-DOS version 3.3 1461–63
 - MS-DOS, MS OS/2 compatibility 495
- Error handling. *See also* Critical error handler;
Extended error information
 - file control block 269
 - file handle function 250–51
- Error messages 24–25
- Escape (Esc) characters 731
 - in CTERM.C terminal emulator 244–45
- Escape sequences, controlling screen display with 731–36
- Escape to Shell (SYMDEB !) 1154–55
- Evans, Eric 37, 39
- Examine Symbol Map (SYMDEB X) 1138–39
- Exception handler(s) 385–408
 - communications device driver 223–25
 - Control-C handler 386–89
 - critical error handler 390–98
 - extended error information 401–8
 - hardware-generated exception interrupts 398–400
 - overview of 385–86
- EXE2BIN utility 144, 971–73
- EXEC function 321–43. *See also* Interrupt 21H
 - Function 4BH
 - functioning of 322–23
 - loading external commands with 79
 - loading overlays with 336–41
 - loading and executing 336–37
 - making memory available 335–36
 - preparing parameters 336
 - program example 337–42
 - loading programs with 323–35
 - making memory available 323
 - parent and child program example 329–33
 - preparing parameters 323–26
 - running child programs 327–29
 - using COMMAND.COM with 328–29
 - loading shell program with 328
 - running SORT as a child process with 442–46
- EXECSORT.ASM program 442–46
- Execute Command on File Set (FOR) 760–61
- EXEMOD utility 974–76
- EXEPACK utility 977–79
- .EXE program files 23, 64, 107–42
 - compressing 977
 - vs* .COM programs 147–48
 - controlling the structure of
 - MASM GROUP directive 131–32
 - MASM SEGMENT directive 125–30
 - sample program 132–37
 - converting to binary memory-image and .COM files 971
 - creating with LINK 643–44(fig.) (*see also* Object Linker)
 - giving control to 108–15
 - preallocated memory 112–13
 - program segment prefix 108–11
 - registers 113–15
 - stacks 111–12
 - loading 124–25
 - memory allocated to 300, 302–3
 - memory diagram 137(fig.)
 - memory map report 136–37(fig.)
 - memory map segments (*see* Memory segments)
 - memory models and 137–40
 - modifying file header with EXEMOD 140–41, 974–76
 - patching with DEBUG 141–42, 585–86
 - structure of 119–24
 - file header 119–24
 - load module 124
 - terminating 115–19
 - RET instruction 118–19
 - Terminate Process function 119
 - Terminate Process with Return Code function 115–17
 - Terminate Program interrupt 117
 - terminating and staying resident 119
 - Warm Boot/Terminate vector 117–18
 - Windows construction of 518–20
- EXIT command 853
- Expanded memory 907–8, 305–16
 - checking for 307–9
 - manager 305–6
 - relationship to conventional memory 306(fig.)
 - using the manager 309–16
 - error codes 313–14(table)
 - program skeleton 314–15(fig.)
 - software interface to application programs provided by 310–12(table)
- Expanded Memory Specification (EMS) 305
- EXP.BAS programs 542
 - corrected code 554–55

EXP.BAS programs (*continued*)
 incorrect code 550–51
 EXTDEF External Names Definition object record
 651, 663–64
 Extended error information 401–8
 Function 59H and newer system calls 406–8
 Function 59H and older system calls 405–6
 MS-DOS version 3.3 1461–63
 MS-DOS versions 2.0 and 3.0 401–5
 TSR set/get functions 352
 Extended memory 316–19, 907
 block move descriptor table format 317(table)
 PC/AT ROM BIOS Interrupt 15H functions
 316–17, 316–17(tables)
 program transferring data from, to conventional
 memory 318–19
 External disk drive, configurable driver for 826

F

Family API 489–90
 FASTOPEN command 1433–34
 FCBS command (CONFIG.SYS) 44, 788, 799–800
 FC command 785, 854–57
 FDISK command 92, 858–62
 MS-DOS version 3.3 1437
 File allocation table (FAT) 54, 97–101
 analyze for errors 774, 775
 assembly-language routine to access 12-bit and
 16-bit 100(figs.)
 development of 8, 13, 23
 initialize 865
 relationship to file data area 98, 99(fig.)
 space allocation 98(fig.)
 File(s) and file/record management 247–78. *See also*
 Batch file(s); .COM program files; .EXE
 program files
 attribute getting/setting 261–62
 backing up 745–51
 changing name 912
 changing read-only/archive attributes 743
 closing
 with FCBs 271
 with handles 255–56
 comparing 785–87, 854–57
 copying 806, 955
 creating
 with FCBs 269
 with handles 251–53
 date/time getting and setting 262
 date/time stamping of 25

File(s) and file/record management (*continued*)
 delete/erase command and 813
 deleting
 with FCBs 276–77
 with handles 260–61
 displaying 947
 duplicating/redirecting handles 262–63
 error handling
 with FCBs 269
 with handles 250–51
 file control block (*see* File control blocks)
 finding size of, and testing for existence 277
 getting/setting file attributes 261–62
 getting/setting file date and time 262
 handles (*see* File handles)
 hidden 774, 940–41
 historical perspective 247–48
 loading 1037, 1113
 MS-DOS version 3.3 changes 1433–35, 1448–51
 names (*see* Filenames)
 opening existing
 with FCBs 270–71
 with handles 253–55
 positioning the read/write pointer 258–59
 reading and writing
 with FCBs 271–75
 with handles 256–58
 recovering 910
 renaming
 with FCBs 275–76
 with handles 260
 restoring backup 918
 setting maximum open 799–800, 801–2
 system calls for 1182–83
 transferring system 940
 transferring with EDLINT 850
 updating 914
 writing file or sectors 1052, 1136
 File control blocks (FCBs) 22, 32, 38, 44, 247, 263–77
 closing files 271
 compatibility issues 494
 creating files 269
 DEBUG initializing 582–83
 default, in executing child programs 327
 deleting files 276–77
 error handling and 269
 extended 266–67
 finding file size and testing for existence 277
 opening files 270–71
 parsing filenames 268–69
 program segment prefixes and 267–68
 reading/writing files 271–75
 renaming files 275–76

- File control blocks (*continued*)
 - setting maximum open files using 799–800
 - structure of 264–67
 - extended 1475(table), 1476(fig.)
 - normal 1473(fig.), 1474–75(table)
- File data area 103
 - relationship to FAT 98, 99
- File handles 32, 38, 56, 801–2, 247–63
 - closing a file 255–56
 - creating a file 251–53
 - deleting a file 260–61
 - duplicating and redirecting handles 262–63
 - error handling 250–51
 - getting/setting date and time 262
 - getting/setting file attributes 261–62
 - opening an existing file 253–55
 - positioning the read/write pointer 258–59
 - reading and writing with 256–58
 - renaming a file 260
- File header 119–24
 - modify with EXEMOD 974–76
 - segmented (new) .EXE format 1487–97
- File management commands 728
 - ATTRIB 743–44
 - BACKUP 745–51
 - COMP 785–87
 - COPY 806–9
 - DEL/ERASE 813–14
 - EDLIN 829–52
 - FC 854–57
 - RECOVER 910–11
 - RENAME/REN 912–13
 - REPLACE 914–17
 - RESTORE 918–22
 - TYPE 947
 - XCOPY 955–59
- File management system, MS-DOS
 - networking and 44
 - versions 2.x 30–32
- File menu (CodeView) 1159
- Filenames 101
 - common extensions for 1485–86
 - compatibility issues 492–93
 - parameters 1040, 1116
 - parsing 268–69
- FILES command (CONFIG.SYS) 250, 789, 801–2
- File set, execute command or program on a 760
- File sharing support, installing 933
- File system
 - block device layout of 93–103
 - boot sector 94–96
 - file allocation table 97–101
 - file area 103
- File system (*continued*)
 - block device layout (*continued*)
 - root directory 101–3
 - MS-DOS kernel 54–55
- Fill Memory
 - DEBUG F 1031–32
 - SYMDEB F 1105–6
- Filter(s) 429–46
 - building 431–41
 - how filters work 430–31
 - system support for 429–30
 - used as child process 441–46
- Filter commands 729, 863, 896, 935
- Find Character String (FIND) 863–64
- FIND command 863–64
- FIND.C program 439–41
- Find First File. *See* Interrupt 21H Function 11H; Interrupt 21H Function 4EH
- Find Next File. *See* Interrupt 21H Function 12H; Interrupt 21H Function 4FH
- Fixed disk
 - configuring 858–62
 - interleaving 90(fig.)
 - layout of 86–87
 - partitions 90–92, 858
 - sectors 88–89
- FIXUPP Fixup object record 651, 682–93
 - examples 686–93
 - fixup field 684–86
 - FRAME fixup methods 683
 - location 686
 - TARGET fixup methods 684
 - thread field 682–84
- Flags
 - display with DEBUG 1045–47
 - maintained by DEBUG 1023
 - maintained by SYMDEB 1060
- Floating-point numbers
 - display
 - 10-byte 1087–88
 - long (64-bit) 1083–84
 - short (32-bit) 1085–86
 - enter
 - 10-byte 1102–03
 - long (64-bit) 1098–99
 - short (32-bit) 1100–1101
- Floppy disk
 - comparing 818–21
 - copying 822–26
 - layout of 86–87
 - sectors 88–89
- Flow control 168, 204

- Flush Buffer, Read Keyboard. *See* Interrupt 21H Function 0CH
- Flux reversal 86
- Force Duplicate File Handle. *See* Interrupt 21H Function 46H
- FOR command (BATCH) 66, 753, 760–61
- Foreground program 900
- Format and Verify Track on Logical Drive. *See* Interrupt 21H Function 44H Subfunction 0DH
- FORMAT command 44, 865–71
- ASSIGN and 741
 - directory format 281–83
 - DISKCOPY and 822
 - FDISK and 858
 - JOIN and 877–78
- Format Disk Tracks. *See* Interrupt 13H Function 05H
- FORTRAN (language) 8, 14
- FORTRAN Compiler, Microsoft
- memory models using 137–40
 - utilities with 974, 977, 980, 987, 999
- Free Memory Block. *See* Interrupt 21H Function 49H
- Frequency modulation (FM) recording 86
- Function calls. *See* System calls
- G**
- Gates, Bill 8(fig.), 16(fig.)
- in the development of early BASIC 3–8, 11
 - in the development of MS-DOS 14–15, 20
- General Protection exception. *See* Interrupt 0DH
- Generate Cross-Reference Listing (CREF) 967–70
- Generic I/O Control for Block Devices. *See* Interrupt 21H Function 44H Subfunction 0DH
- Generic I/O Control for Handles. *See* Interrupt 21H Function 44H Subfunction 0CH
- Get and Set Time. *See* Interrupt 1AH
- Get Assign-List Entry. *See* Interrupt 21H Function 5FH Subfunction 02H
- Get Current Country. *See* Interrupt 21H Function 38H
- Get Current Directory. *See* Interrupt 21H Function 47H
- Get Current Disk. *See* Interrupt 21H Function 19H
- Get Current Drive Parameters. *See* Interrupt 13H Function 08H
- Get Current Video Mode. *See* Interrupt 10H Function 0FH
- Get Date. *See* Interrupt 21H Function 2AH
- Get Default Drive Data. *See* Interrupt 21H Function 1BH
- Get Device Data. *See* Interrupt 21H Function 44H Subfunction 00H
- Get Disk Free Space. *See* Interrupt 21H Function 36H
- Get Disk Status. *See* Interrupt 13H Function 01H
- Get Disk Type. *See* Interrupt 21H Function 15H
- Get Drive Data. *See* Interrupt 21H Function 1CH
- Get DTA Address. *See* Interrupt 21H Function 2FH
- Get Extended Country Information. *See* Interrupt 21H Function 65H
- Get Extended Error Information. *See* Interrupt 21H Function 59H
- Get File Size. *See* Interrupt 21H Function 23H
- Get Interrupt Vector. *See* Interrupt 21H Function 35H
- Get Lead Byte Table. *See* Interrupt 21H Function 63H
- Get Logical Drive Map. *See* Interrupt 21H Function 44H Subfunction 0EH
- Get Machine Name. *See* Interrupt 21H Function 5EH Subfunction 00H
- Get MS-DOS Version Number. *See* Interrupt 21H Function 30H
- Get Peripheral Equipment List. *See* Interrupt 11H
- Get Port Status. *See* Interrupt 14H Function 03H
- Get Printer Setup. *See* Interrupt 21H Function 5EH Subfunction 03H
- Get Printer Status. *See* Interrupt 17H Function 02H
- Get Program Segment Prefix Address. *See* Interrupt 21H Function 51H; Interrupt 21H Function 62H
- Get Return Code of the Child Process. *See* Interrupt 21H Function 4DH
- Get/Set Allocation Strategy. *See* Interrupt 21H Function 58H
- Get/Set Control-C Check Flag. *See* Interrupt 21H Function 33H
- Get/Set Date/Time of File. *See* Interrupt 21H Function 57H
- Get/Set File Attributes. *See* Interrupt 21H Function 43H
- Get Shift Status. *See* Interrupt 16H Function 02H
- Get Time. *See* Interrupt 21H Function 2CH
- Get/Set Time/Date. *See* Interrupt 1AH
- Get Usable Memory Size (KB). *See* Interrupt 12H
- Get Verify Flag. *See* Interrupt 21H Function 54H
- Gilbert, Paul 5–6
- Global descriptor table (GDT) 317
- Go
- DEBUG G 584–85, 1033–34
 - SYMDEB G 1107–8
- GOTO command (BATCH) 67, 753, 762–63
- GRAFTABL command 872–73
- MS-DOS version 3.3 1445
- Graphics
- loading character set 872–73
 - loading screen-dump program 874–76
 - screen-display attributes 734
- Graphics Character Table. *See* Interrupt 1FH

GRAPHICS command 874–76
 Graphics Device Interface (GDI), Windows 529–37
 bit-block transfers 535–36
 device context 530
 device-context attributes 531
 device-independent programming 530–31
 drawing functions 533
 mapping modes 531–32
 metafiles 536–37
 raster operations for pens 534–35
 text and fonts 536
 Greenberg, Bob 8(fig.)
 GROUP directive (MASM), controlling .EXE
 programs with 131–32
 sample .EXE program using 132–37
 GRPDEF Group Definition object record 651, 680–81

H

Handle-type function calls, for accessing character
 devices 150, 152–53, 155, 158, 161, 163
 Hangeul characters 37
 Hard disk. *See* Fixed disk
 Hardware
 breakpoints 640, 641, 642
 for communications 170–80
 compatibility issues, with MS OS/2 489–92
 BIOS 491
 CPU speed 491
 family API 489–90
 linear *vs* segmented memory 490–91
 program timing 491
 protected mode 489
 debugging aids 640–42
 developers of, and MS-DOS 27–29, 35–37
 MS-DOS requirements for
 memory 58
 microprocessor 57–58
 peripheral devices 59
 ROM BIOS 59–60
 Hardware instrumentation 555–56
 Hardware interrupts 398–400, 409–27
 categories 411–12
 characteristics of maskable interrupts 412–13
 handling maskable interrupts 413–19
 IBM interrupt usage 410(table)
 Intel reserved exception 398(table),
 409–10(table)
 programming for 419–27
 sample replacement handler 419–24
 supplementary handlers 424–26

Hardware IRQ0 (timer tick). *See* Interrupt 08H
 Hardware IRQ1 (keyboard). *See* Interrupt 09H
 Hardware IRQ2 (reserved). *See* Interrupt 0AH
 Hardware IRQ3 (COM2). *See* Interrupt 0BH
 Hardware IRQ4 (COM1). *See* Interrupt 0CH
 Hardware IRQ5 (fixed disk). *See* Interrupt 0DH
 Hardware IRQ6 (floppy disk). *See* Interrupt 0EH
 Hardware IRQ7 (printer). *See* Interrupt 0FH
 Heads, read/write 86, 88
 HELLO.ASM program 357–59
 as typical object module 651–54
 Help menu (CodeView) 1162
 Help or Evaluate Expression (SYMDEB ?) 1152–53
 Hercules Graphics Card 157
 Hewlett Packard HP150 computer 34
 Hexadecimal arithmetic 1035, 1109
 binary-to-hexadecimal file conversion
 utility 1503–5
 Hexadecimal bytes
 displaying contents of memory as 1079–80
 entering into memory 1095–96
 Hexadecimal object file format 1499–1505
 .HEX files, and DEBUG 585–86, 1020, 1052
 /HIGH switch, LINK use of 719–21
 Hooks, MS-DOS 53
 Hot-key sequence 348, 382
 Huge memory model 139

I

IBMBIO.COM 20, 33, 52, 448, 774, 940
 IBM Corporation computers
 interrupt usage 410(table)
 PC (Personal Computer) 19(fig.), 20, 21(fig.),
 26, 34(fig.)
 PC/AT computer 39–43, 417–18
 PCjr computer 35, 36, 37
 PC/XT computer 30, 34(fig.)
 Personal System/2, MS-DOS version 3.3
 1448
 role in the development of MS-DOS 14–15, 26
 IBMDOS.COM 20, 447, 774, 940
 loading 52
 IBM extended character set 1465–67
 IBM Professional Debug Utility 641
 Idle Interrupt. *See* Interrupt 28H
 IF command (BATCH) 67, 753, 764–65
 with GOTO 762
 Include Comment Line (REM) 768
 InDOS flag 355–56
 Inference rule, and MAKE utility 1001
 Information management system calls, list 1183

- Initialization. *See* Interrupt 14H Function 00H
- Initialize Disk (FORMAT) 865–71
- Initialize Hard-Disk Parameter Table. *See* Interrupt 13H Function 09H
- Initialize Port Parameters. *See* Interrupt 14H Function 00H
- Initialize Printer. *See* Interrupt 17H Function 01H
- Initial SP value field (.EXE file header) 122
 - modifying 140
- Input from Port
 - DEBUG I 1036
 - SYMDEB I 1110
- Input/output (I/O). *See also* Character-device
 - input/output
 - input port 1036, 1110
 - output port 1042, 1118
 - redirection 67–68
 - redirection and filters 429–30
 - SYMDEB redirection 1143–49
- INS8250 Universal Asynchronous Receiver Transmitter (UART) 171–72
 - architecture 172–79
 - bit rate divisor table 175(table)
 - control circuits 173, 174–77
 - interrupt enable register constants 177(table)
 - interrupt identification and causes 178(table)
 - line control register bit values 175–76(table)
 - line status register bit values 177(table)
 - modem control register bit values 176(table)
 - port offset from base address 174(table)
 - programming interface 173–74
 - receiver 172
 - status circuits 173, 177–79
 - transmitter 172–73
 - programming 179–80
- Insert Lines (EDLIN I) 840
- Inspection-and-observation debugging 547–49
- Install Device Driver (DEVICE) 795–96
- Install File-Sharing Support (SHARE) 933–34
- Instruction sets
 - 8086/8088 1479–80
 - 80286 1480–82
 - 80386 1482–84
- Instrumentation debugging
 - external 555–72
 - internal 549–55
- INT24. ASM critical error handling program 394, 395–98
- Intel 4004 chip 5(fig.)
- Intel 8008 chip 5(fig.)
- Intel 8080 chip 5(fig.), 10
- Intel 8086 chip 11(fig.), 12, 58
 - compatibility issues 1507–8
- Intel 8086 chip (*continued*)
 - exception interrupts 398(table), 409–10(table)
 - instruction set 1479–80
 - interrupt priorities 411
- Intel 8088 chip 58
 - compatibility issues 1507–8
 - instruction set 1479–80
- Intel 8259A Programmable Interrupt Controller (PIC) 349, 411, 414(fig.), 415, 416(fig.). *See also* Maskable interrupts
- Intel 80186 chip 58
- Intel 80188 chip 58
- Intel 80286 chip 42(fig.), 58
 - compatibility issues 489–92
 - instruction set 1481–82
- Intel 80386 chip 42(fig.), 58
 - compatibility issues 489
 - instruction set 1483–84
- Interleaving, disk 89–90
- Internal disk buffers, configure 791–92
- Internal stacks
 - configuring 805
 - at entry to a critical error exception handler 391(fig.)
 - in .EXE programs 111–12
 - performing stack trace 1111–12
 - in TSR programs 353, 354–55(fig.)
- Internationalization
 - MS-DOS and 32–33, 35–37
 - MS-DOS version 2.25 1415–16
 - new national language support, MS-DOS version 3.3 1438–48, 1451–55
 - support 793
 - Windows 538
- Interrupt(s)
 - configure internal stacks for 805
 - daisy-chaining handlers 557
 - hardware (*see* Hardware interrupts)
 - manual 640, 641
 - TSR processing of hardware 349
- Interrupt 00H, Divide by Zero 398, 399, 409
 - demonstration handler 419–24
- Interrupt 01H, Single Step 398, 399, 409
- Interrupt 02H, Nonmaskable Interrupt (NMI) 398, 399, 409, 411
- Interrupt 03H, Breakpoint Trap 400, 409
- Interrupt 04H, Overflow Trap 398, 400, 409
- Interrupt 05H
 - IBM, Print Screen 410
 - Intel, BOUND Range Exceeded 398, 400, 409
- Interrupt 06H
 - IBM, Unused 410
 - Intel, Invalid Opcode 398, 400, 409

- Interrupt 07H
 - IBM, Unused 410
 - Intel, Coprocessor Not Available 398, 409
- Interrupt 08H
 - IBM, Hardware IRQ0/ (Time Tick) 382, 383, 410, 425-26
 - Intel, Double-Fault Exception 398, 409
- Interrupt 09H
 - IBM, Hardware IRQ1 (Keyboard) 348, 382, 410
 - Intel, Coprocessor Segment Overrun 398, 409
- Interrupt 0AH
 - IBM, Hardware IRQ2 (Reserved) 410
 - Intel, Invalid Task State Segment (TSS) 398, 409
- Interrupt 0BH
 - IBM, Hardware IRQ3 (COM2) 410
 - Intel, Segment Not Present 398, 409
- Interrupt 0CH
 - IBM, Hardware IRQ4 (COM1) 410
 - Intel, Stack Exception 398, 409
- Interrupt 0DH
 - IBM, Hardware IRQ5 (Fixed Disk) 410
 - Intel, General Protection Exception 398, 409
- Interrupt 0EH
 - IBM, Hardware IRQ6 (Floppy Disk) 410
 - Intel, Page Fault 398, 409
- Interrupt 0FH
 - IBM, Hardware IRQ7 (Printer) 410
 - Intel, Reserved 398, 410
- Interrupt 10H
 - IBM, PC ROM BIOS video driver 159, 410, 872, 1513-18
 - Function 00H, Set Video Mode 1513
 - Function 01H, Set Cursor Size and Shape 1514
 - Function 02H, Set Cursor Position 1514
 - Function 03H, Read Cursor Position, Size, and Shape 1514
 - Function 04H, Read Light-Pen Position 1514-15
 - Function 05H, Select Active Page 1515
 - Function 06H, Scroll Window Up 1515
 - Function 07H, Scroll Window Down 1515
 - Function 08H, Read Character and Attribute at Cursor 1515-16
 - Function 09H, Write Character and Attribute 1516
 - Function 0AH, Write Character Only 1516
 - Function 0BH, Select Color Palette 1516
 - Function 0CH, Write Pixel Dot 1517
 - Function 0DH, Read Pixel Dot 1517
 - Function 0EH, Write Character as TTY 1517
 - Function 0FH, Get Current Video Mode 1517
 - Function 13H, Write Character String 1518
 - Intel, Coprocessor Error 398, 410
- Interrupt 11H, Get Peripheral Equipment List 1518
- Interrupt 12H, Get Usable Memory Size (KB) 1519
- Interrupt 13H, Disk Services 1519-23
 - Function 00H, Reset Disk System 1519
 - Function 01H, Get Disk Status 1519-20
 - Function 02H, Read Disk Sectors 1520
 - Function 03H, Write Disk Sectors 1520
 - Function 04H, Verify Disk Sectors 1520
 - Function 05H, Format Disk Tracks 1520
 - Function 08H, Get Current Drive Parameters 1520-21
 - Function 09H, Initialize Hard-Disk Parameter Table 1521
 - Function 0AH, Read Long 1521
 - Function 0BH, Write Long 1521
 - Function 0CH, Seek to Head 1521
 - Function 0DH, Alternate Disk Reset 1522
 - Function 10H, Test for Drive Ready 1522
 - Function 11H, Recalibrate Drive 1522
 - Function 14H, Controller Diagnostic 1522
 - Function 15H, Get Disk Type 1522-23
 - Function 16H, Check for Change of Floppy Disk Status 1523
 - Function 17H, Set Disk Type 1523
- Interrupt 14H, Serial Port Services 161, 1523-25
 - debugging and 556-57
 - Function 00H, Initialize Port Parameters 222, 1523-24
 - Function 01H, Send One Character 1524
 - Function 02H, Receive One Character 1524
 - Function 03H, Get Port Status 1524-25
- Interrupt 15H, Miscellaneous System Services 1525-26
 - access to extended memory functions 316-17(table)
 - block move descriptor table format 317(table)
 - Function 02H, Read Data from Cassette 1525-26
 - Function 03H, Write Data to Cassette 1526
 - Function 87H, Move Extended Memory Block 316-17
 - Function 88H, Obtain Size of Extended Memory 316(table)
- Interrupt 16H, Keyboard Services 1526-27
 - Function 00H, Read Next Character 1526
 - Function 01H, Report If Character Ready 1527
 - Function 02H, Get Shift Status 1527
- Interrupt 17H, Printer Services 1527-28
 - Function 00H, Send Byte to Printer 1527
 - Function 01H, Initialize Printer 1528
 - Function 02H, Get Printer Status 1528
- Interrupt 18H, Transfer Control to ROM-BASIC 1528
- Interrupt 19H, Reboot Computer (Warm Start) 1528
- Interrupt 1AH, Get/Set Time/Date 1528-30
 - Function 00H, Read Current Clock Count 1528-29

Interrupt 1AH (*continued*)

- Function 01H, Set Current Clock Count 1529
- Function 02H, Read Real-Time Clock 1529
- Function 03H, Set Real-Time Clock 1529
- Function 04H, Read Date from Real-Time Clock 1529–30
- Function 05H, Set Date in Real-Time Clock 1530
- Function 06H, Set Alarm 1530
- Function 07H, Reset Alarm (Turn Alarm Off) 1530

Interrupt 1BH, Control-Break (user defined) 387–89, 410

Interrupt 1CH, Timer Tick (user defined) 410

Interrupt 1DH, Video Parameter Pointer 410

Interrupt 1EH, Disk Parameter Pointer 410

Interrupt 1FH, Graphics Character Table 872–73

Interrupt 20H, Terminate Program 63, 108, 1185–86
terminating .EXE programs 117, 118

Interrupt 21H, MS-DOS system calls 63, 110, 1050

for accessing directories 284–86(table)

compatibility, with MS OS/2 493–94

error information 401, 402

for file and record management 248(table)

Function 00H, Terminate Process 1187–88

Function 01H, Character Input with Echo 154, 1189–90

Function 02H, Character Output 158, 1191–92

Function 03H, Auxiliary Input 161, 169, 1193–94

Function 04H, Auxiliary Output 161, 1195–96

Function 05H, Print Character 163, 1197–98

Function 06H, Direct Console I/O 154, 158, 1199–1200

Function 07H, Unfiltered Character Input
Without Echo 154, 1201–2

Function 08H, Character Input Without Echo
154, 169, 1203–4

Function 09H, Display String 158, 1205–6

Function 0AH, Buffered Keyboard Input 154, 155, 1207–8

Function 0BH, Check Keyboard Status 154, 155, 169, 1209–10

Function 0CH, Flush Buffer, Read Keyboard 154, 155, 1211–12

Function 0DH, Disk Reset 1213–14

Function 0EH, Select Disk 1215–16

Function 0FH, Open File with FCB 270, 1217–19

Function 10H, Close File with FCB 271, 1220–21

Function 11H, Find First File 277, 286, 287, 1222–24

Function 12H, Find Next File 286, 287, 1225–26

Function 13H, Delete File 276–77, 1227–28

Function 14H, Sequential Read 272, 1229–30

Function 15H, Sequential Write 272, 1231–32

Interrupt 21H (*continued*)

Function 16H, Create File with FCB 156, 269, 1233–34

Function 17H, Rename File 275, 287, 1235–36

Function 19H, Get Current Disk 1237

Function 1AH, Set DTA Address 268, 353, 1238–39

Function 1BH, Get Default Drive Data 1240–41

Function 1CH, Get Drive Data 1242–44

Function 21H, Random Read 272, 1245–46

Function 22H, Random Write 273, 1247–48

Function 23H, Get File Size 277, 1249–50

Function 24H, Set Relative Record 1251–52

Function 25H, Set Interrupt Vector 352, 419, 1253–54

Function 26H, Create New Program Segment
Prefix 1255–56

Function 27H, Random Block Read 273, 1257–59

Function 28H, Random Block Write 273–75, 1260–62

Function 29H, Parse Filename 268, 1263–65

Function 2AH, Get Date 1266–67

Function 2BH, Set Date 1268–69

Function 2CH, Get Time 1270–71

Function 2DH, Set Time 1272–73

Function 2EH, Set/Reset Verify Flag 1274–75

Function 2FH, Get DTA Address 268, 353, 1276

Function 30H, Get MS-DOS Version Number
1277–78

Function 31H, Terminate and Stay Resident 351, 381, 1279–80 (*see also* Terminate-and-stay-resident utilities)

Function 33H, Get/Set Control-C Check Flag
1281–82

Function 34H, Return Address of InDOS Flag
355–56, 1283

Function 35H, Get Interrupt Vector 307, 315, 352, 419, 1284

Function 36H, Get Disk Free Space 1285–86

Function 38H, Get/Set Current Country 793, 1451

Get Current Country 1287–89

Set Current Country 1290

Function 39H, Create Directory 287, 1291–92

Function 3AH, Remove Directory 287, 1293–94

Function 3BH, Change Current Directory 281, 288, 1295–96

Function 3CH, Create File with Handle 251, 287, 1297–99

Function 3DH, Open File with Handle 155, 158, 161, 163, 253, 282, 307, 315, 1300–1303

Function 3EH, Close File 255, 307, 1304–5

Function 3FH, Read File or Device 154, 155, 161, 256, 431, 1306–7

Interrupt 21H (continued)

- Function 40H, Write File or Device 158, 161, 163, 256, 431, 1308-9
- Function 41H, Delete File 260, 287, 1310-11
- Function 42H, Move File Pointer 258, 1312-14
- Function 43H, Get/Set File Attributes 261-62, 287, 1315-16
- Function 44H, IOCTL 164-66, 203, 315, 1317-18
 - extended MS-DOS version 3.3 1455-58
 - Subfunction 00H, Get Device Data 164, 165, 307, 1319-21
 - Subfunction 01H, Set Device Data 164, 165, 1322-23
 - Subfunction 02H, Receive Control Data from Character Device 164-65, 1324-25
 - Subfunction 03H, Send Control Data to Character Device 165, 1324-25
 - Subfunction 04H, Receive Control Data from Block Device 1326-28
 - Subfunction 05H, Send Control Data to Block Device 1326-28
 - Subfunction 06H, Check Input Status 155, 165, 1329-30
 - Subfunction 07H, Check Output Status 165, 1329-30
 - Subfunction 08H, Check If Block Device Is Removable 1331-32
 - Subfunction 09H, Check If Block Device Is Remote 1333-34
 - Subfunction 0AH, Check If Handle Is Remote 165, 1335-36
 - Subfunction 0BH, Change Sharing Retry Count 1337-38
 - Subfunction 0CH, Generic I/O Control for Handles 165, 1339-40, 1455-58
 - Subfunction 0DH, Generic I/O Control for Block Devices 1341-42
 - Subfunction 0DH, minor code 40H, Set Device Parameters 1343-46
 - Subfunction 0DH, minor code 41H, Write Track on Logical Drive 1350-51
 - Subfunction 0DH, minor code 42H, Format and Verify Track on Logical Drive 1352-53
 - Subfunction 0DH, minor code 60H, Get Device Parameters 1347-49
 - Subfunction 0DH, minor code 61H, Read Track on Logical Drive 1350-51
 - Subfunction 0DH, minor code 62H, Verify Track on Logical Drive 1352-53
 - Subfunction 0EH, Get Logical Drive Map 1354-55

Interrupt 21H (continued)

- Function 44H, IOCTL (continued)
 - Subfunction 0FH, Set Logical Drive Map 1354-55
- Function 45H, Duplicate File Handle 67, 262, 1356-57
- Function 46H, Force Duplicate File Handle 67, 263, 1358-59
- Function 47H, Get Current Directory 288, 1360-61
- Function 48H, Allocate Memory Block 299, 303, 352, 1362-63
- Function 49H, Free Memory Block 299, 303, 352, 1364-65
- Function 4AH, Resize Memory Block 299, 323, 1366-67
- Function 4BH, Load and Execute Program (EXEC) 64, 718, 1368-74. (see also EXEC function)
- Function 4CH, Terminate Process with Return Code 115-17, 144, 1375-76
- Function 4DH, Get Return Code of Child Process 328, 1377-78
- Function 4EH, Find First File 285, 286, 287, 288-90, 1379-81
- Function 4FH, Find Next File 285, 286, 287, 288-90, 1382-84
- Function 50H, Set Program Segment Prefix Address 352, 383
- Function 51H, Get Program Segment Prefix Address 352, 383
- Function 54H, Get Verify Flag 1385
- Function 56H, Rename File 260, 287, 1386-87
- Function 57H, Get/Set Date/Time of File 262, 265, 287, 1388-90
- Function 58H, Get/Set Allocation Strategy 1391-92
- Function 59H, Get Extended Error Information 269, 327, 383-84, 1393-96
 - and newer system calls 406-8
 - and older system calls 405-6
- Function 5AH, Create Temporary File 251, 252, 1397-98
- Function 5BH, Create New File 251, 252, 1399-1400
- Function 5CH, Lock/Unlock File Region 1401-3
- Function 5DH, Set Extended Error Information 352
- Function 5EH, Network Machine Name/Printer Setup
 - Subfunction 00H, Get Machine Name 1404
 - Subfunction 02H, Set Printer Setup 1405-6
 - Subfunction 03H, Get Printer Setup 1405-6

- Interrupt 21H (*continued*)
- Function 5FH, Get/Make Assign-List Entry
 - Subfunction 02H, Get Assign-List Entry 1407-8
 - Subfunction 03H, Make Assign-List Entry 1409-10
 - Subfunction 04H, Cancel Assign-List Entry 1411-12
 - Function 62H, Get Program Segment Prefix Address 1413-14
 - Function 63H, Get Lead Byte Table 1415-16
 - Function 65H, Get Extended Country Information 1451-54
 - Function 66H, Select Code Page 1454-55
 - Function 67H, Set Handle Count 1448-50
 - Function 68H, Commit File 1448, 1450-51 for terminate-and-stay-resident programs 350-53
- Interrupt 22H, Terminate Routine Address 63, 110, 1417
- Interrupt 23H, Control-C Handler Address 63, 110, 386-89, 1418
- Interrupt 24H, Critical Error Handler Address 63, 110, 354, 390-98, 1419-21
- MS-DOS versions 2.0 and later 402-3
- Interrupt 25H, Absolute Disk Read 63, 1422-23
- Interrupt 26H, Absolute Disk Write 63, 1424-25
- Interrupt 27H, Terminate and Stay Resident 63, 266, 351, 1426-27. *See also* Terminate-and-stay-resident utilities
- Interrupt 28H, Idle Interrupt 63, 266, 353
- Interrupt 2FH, Multiplex Interrupt 63, 356-57, 381, 1428-29
- Interrupt 30H 63
- Interrupt 60H 565, 600
- Interrupt 67H 306, 307, 309, 315
- Interrupt enable register constants, INS8250 UART chip 177(table)
- Interrupt identification and causes, INS8250 UART chip 178(table)
- Interrupt request lines (IRQ) 414, 416-19
- 16-level designs 417-19
 - cascade effect 417, 418(fig.)
 - eight-level designs 417(table)
- Interrupt routine (*Intr*), device driver 453-68
- Build BIOS Parameter Block function 459-60
 - command-code functions 454-55
 - Device Open/Close functions 464-65
 - Flush Input/Output Buffer functions 463-64
 - Generic IOCTL function 466
 - Get/Set Logical Device functions 467-68
 - Init (Initialization) function 455-57
 - Input/Output Status functions 463
 - IOCTL Read/Write functions 464
- Interrupt routine (*continued*)
- Media Check function 457-59
 - Nondestructive Read function 462
 - Output Until Busy function 466
 - Read, Write, and Write with Verify functions 461-62
 - Removable Media function 465-66
- Interrupt service routine (ISR) 180, 203-4, 412
- in COMDVR.ASM 196-98, 203-4
 - hardware module 215-22
- Interrupt vector functions, in TSR programs 352
- Interrupt vector table 58
- in conventional memory 297-98
 - initializing 69, 70(fig.)
- Invalid Opcode exception. *See* Interrupt 06H
- Invalid Task State Segment (TSS) exception. *See* Interrupt 0AH
- IOCTL. *See* Interrupt 21H Function 44H
- IO.SYS 33, 448, 774, 940
- BIOS and 61-62
 - loading 52, 72(fig.)
 - modules 73
- ISO Open System Interconnect 42
- ISR. *See* Interrupt service routine
- J**
- JOIN command 877-78
- ASSIGN and 741
 - BACKUP and 747
 - CHKDSK and 775
 - DISKCOMP and 818
 - DISKCOPY and 822
 - FORMAT and 866
 - MKDIR/MD and 885
- Join Disk to Directory (JOIN) 877-78
- Jump to Label (GOTO) 762-63
- K**
- Kanji characters 37(fig.)
- Kernel. *See* MS-DOS kernel
- KEYB command 1440-41
- Keyboard 154-57
- ANSI.SYS key and extended key codes 1471-72
 - character input functions 154(table)
 - defining 879, 1440-41
 - redefining to a specific string 734-36
 - sample input programs 156-57
 - TSR input (*see* Hot-key sequence)

Keyboard (KEYB) 1440–41
 Keyboard Services. *See* Interrupt 16H
 KEYBxx command 879–81
 Key commands (CodeView) 1163
 Kildall, Gary 10

L

Label(s)
 displaying volume 954
 jumping to batch-file line following specified
 label 762–63
 modify volume 882
 LABEL command 882–84
 ASSIGN and 741
 Lane, Jim 8(fig.)
 Language menu (CodeView) 1161–62
 Large memory model 139
 LASTDRIVE command (CONFIG.SYS) 789, 803
 LC.ASM lowercase filter program 437–39
 LEDATA Logical Enumerated Data object record 651,
 694–95
 Letwin, Gordon 8(fig.)
 Lewis, Andrea 8(fig.)
 Library Manager. *See* LIB utility
 LIB utility 701–2, 980–86
 LIDATA Logical Iterated Data object record 651,
 696–97
 Lifeboat Associates 12, 27
 Line control register bit values 175(table)
 Line Editor (EDLIN) 829–31
 Line number, defined 1058
 Line Status Register bit values 177(table)
 LINK. *See* Object Linker
 LINNUM Line Number object record 651, 672–73
 List Breakpoints (SYMDEB BL) 1071
 List Lines (EDLIN L) 841
 LNames List of Names object record 651, 674–75
 Load and Execute Program. *See* EXEC function;
 Interrupt 21H Function 4BH
 Loader, operating system 52, 72
 Load File or Sectors
 DEBUG L 1037–38
 SYMDEB L 1113–14
 Load Graphics Character Set (GRAFTABL) 872–73
 Load Graphics Screen-Dump Program (GRAPHICS)
 874–76
 Loading MS-DOS 68–83
 COMMAND.COM shell 76–83
 ROM BIOS, POST and bootstrapping 68–72
 system initialization 73–76
 Lock/Unlock File Region 1401–3

Loop or Subroutine, Proceed Through 1043
 LPT1 (first parallel printer port) 151, 163
 LPT2 (second parallel printer port) 151, 163
 LPT3 (third parallel printer port) 151, 163

M

McDonald, Marc 8 (fig.), 9
 Machine Code Display Mode, Enable 1129
 Machine language
 assembling 1024, 1063
 disassembling programs in 1051, 1132
 Macro(s), in MAKE utility 1000–1001
 Macro Assembler, Microsoft. *See* Microsoft Macro
 Assembler
 Maintain Programs (MAKE) 999–1003
 Make Assign-List Entry 1409–10
 Make Directory (MKDIR or MD) 885–86
 MAKE utility 999–1003
 Map files, processed to create symbol files 1004
 MAPSYM utility 593, 1004–6
 MARK condition 172
 Maskable interrupts 412–19
 characteristics of 412–13
 general interrupt sequence 413(fig.)
 handling 413–19
 8259A Programmable Interrupt Controller
 (PIC) 415, 416(fig.)
 IRQ levels 416–19
 MASM. *See* Microsoft Macro Assembler
 MAXALLOC field 121, 124, 322
 .EXE memory 300–301
 modifying 140
 MCOPY program 956–57
 MD command. *See* MKDIR/MD command
 M-DOS, development of 8–9, 12, 15–19
 Medium memory model 138
 Memory 297–319
 allocated to .COM and .EXE programs 142,
 300–305
 comparing areas of 1026, 1074
 conventional (*see* Conventional memory)
 displaying 1027, 1075–90
 entering data into 1029, 1091–1104
 expanded (*see* Expanded memory)
 extended (*see* Extended memory)
 filling 1031, 1105
 linear *vs* segmented 490–91
 making available with EXEC 323, 336–37
 management
 with MS-DOS kernel 53–54
 with Windows 510–11

- Memory (*continued*)
- moving area contents 1039
 - MS-DOS requirements 58
 - preallocated, in .EXE programs 112–13
 - searching 1048, 1125
 - segments (*see* Memory segments)
 - system calls for 1184
 - transient use of, by COMMAND.COM 24
 - TSR RAM management 351–52
 - virtual disk in 907
- Memory arena 298
- Memory-image files, converting .EXE files to 971
- Memory models, for .EXE programs 137–40
- MEMORY parameter 128
- Memory segments
- absolute segments 647
 - alignment of 647, 708–9
 - classes of 707–8
 - concatenated segments 647–48
 - creating values 490–91
 - DGROUP 718–21
 - fixups 648, 649(fig.)
 - frames 646
 - groups for unified addressing 714
 - groups of segments 648–49, 709
 - vs* linear memory 490
 - logical segments 646
 - order and combinations 707–9
 - overlays 715–18
 - relocatable segments 646–47
 - TSR programs 713–14
 - uninitialized data 714–15
- Memory Size. *See* Interrupt 12H
- MEMO.TXT program 252
- Messaging system, Windows 522–29
- Metafiles (Windows) 536–37
- Micro Instrumentation Telemetry Systems (MITS) 4, 7(fig.)
- Microprocessor, MS-DOS requirements for 57–58. *See also specific chips*
- Microsoft Corporation
- 8086 chip technology and 11–13
 - BASIC development 3–8, 14
 - competition with CP/M 9–10, 27–29
 - M-DOS development 8–9, 15–19
 - MS-DOS (*see* Development of MS-DOS; MS-DOS operating system; MS-DOS versions 1.x through version 3.3)
 - OS/2 (*see* MS OS/2)
 - personnel in 1978 8(fig.)
- Microsoft Macro Assembler (MASM)
- description 1007–11
 - messages 1012–19
 - sample program structuring with SEGMENT and GROUP 132–36
- Microsoft Macro Assembler (*continued*)
- using GROUP to control .EXE programs 131–32
 - using SEGMENT to control .EXE programs 125–37
 - utilities with 967, 974, 977, 980, 987, 1004, 1054, 1157
- Microsoft Networks 43–44, 933. *See also* Networking
- Microsoft Object Linker (LINK). *See* Object Linker
- Microsoft Windows. *See* Windows
- MINALLOC field 121, 124
- .EXE memory 300
 - modifying 140
- Miscellaneous System Services. *See* Interrupt 15H
- Mitsubishi Corporation 35
- MKDIR/MD command 885–86
- Mode(s), real *vs* protected operating 58, 316
- MODE command 887
- AUTOEXEC.BAT and 755, 887
 - code-page options 1446–47
 - display 890–91
 - MS-DOS version 3.3 1438, 1446–47
 - printer 888–89
 - redirect printing 894–95
 - serial port 892–93
- Modem 170–71
- Modem Control Register bit values 176(table)
- Modem engine 168, 206–9
- code 207–8
 - implementing with MS-DOS functions 168–70
- Modem Status Register bit values 178(table)
- MODEND Module End object record 651, 661–62
- Modified frequency modulation (MFM) 86
- Modify .EXE File Header (EXEMOD) 974–76
- Modify Volume Label (LABEL) 882–84
- MODULE_A program 132–34
- MODULE_B program 134–35
- MODULE_C program 135–36
- Monochrome Display Adapter (MDA) 157
- MORE command 896
- Move (Copy) Data
- DEBUG M 1039
 - SYMDEB M 1115
- Move Extended Memory Block. *See* Interrupt 15H
- Function 87H
- Move File Pointer. *See* Interrupt 21H Function 42H
- Move Lines (EDLIN M) 842–43
- MS-DOS Executive (Windows) 505–6(fig.)
- MS-DOS kernel 53–55, 62–63, 447. *See also*
- MSDOS.SYS
 - file system 54–55
 - initializing 73, 74
 - memory management 53–54
 - peripheral support 54
 - process control 53

- MS-DOS operating system 51–60. *See also* BIOS; COMMAND.COM; MS-DOS kernel
- basic character devices 151–64
 - basic requirements for 57–60
 - compatibility with OS/2 489–97
 - hardware issues 489–92
 - operating-system issues 492–97
 - development of (*see* Development of MS-DOS)
 - displaying version 952
 - loading 68–83
 - major elements of 61–68
 - system components 52–57
 - system initialization (*see* SYSINIT)
 - three operating system types 51(table)
 - user interface 55 (*see also* COMMAND.COM; SHELL comand)
 - versions 55–57. *See also names of individual versions, e.g.,* MS-DOS versions 1.x
- MSDOS.SYS 62, 447, 774, 940. *See also* MS-DOS kernel
- loading 52, 72(fig.)
 - moving to begin initialization 73, 74(fig.)
- MS-DOS system calls. *See* System calls, MS-DOS
- MS-DOS versions 1.x
- development of 20–29
- MS-DOS versions 2.x
- development of 30–38
 - internal stack use in TSR programs 353, 354–55
- MS-DOS version 3.0
- development of 39–44
 - extended error information 401–8
 - internal stack use in TSR programs 343, 354–55
- MS-DOS version 3.1
- development of 43–44
 - extended error information 401–8
- MS-DOS version 3.2
- development of 44
 - extended error information 401–8
- MS-DOS version 3.3 1433–59
- critical error handling 390
 - new national language support 1438–48
 - programming considerations 1448–58
 - extension of IOCTL 1455–58
 - file management 1448–51
 - internationalization support 1451–55
 - MS-DOS partitions extension 1458
 - user considerations 1433–48
 - batch-file processing 1434–35
 - enhanced commands 1436–38
 - FASTOPEN command 1433–34
 - PC-DOS commands 1435–36
- MS OS/2 operating system, programming for
- compatibility 489–97
 - hardware 489–92
 - operating-system issues 492–97
- Multi-Color Graphics Array (MCGA) 157
- Multiplex Interrupt. *See* Interrupt 2FH
- Multitasking 53
- compatibility issues in 496–97
 - Windows 529
- MYFILE.DAT program 257–58, 274–75
- ## N
- Name File or Command-Tail Parameters
- DEBUG N 1040–41, 1052
 - SYMDEB N 1116–17
- National language support, MS-DOS version 3.3 1438–48. *See also* COUNTRY command
- code pages and code-page switching 1438–39
 - for EGA-only systems 1447
 - for PS/2 and printer 1448
 - modified support commands 1442–47
 - new support commands 1440–42
 - system files 1439
- National Language Support Function (NLSFUNC) command, MS-DOS 1441–42
- Network Adapter card, IBM 42, 43
- Networking
- installing file-sharing support 933
 - MS-DOS versions 3.x 35, 39–44
- Network Machine Name/Printer Setup. *See* Interrupt 21H Function 5EH
- New Executable file header format 1487–97
- code and data segment 1495–97
 - entry table 1493–94
 - imported names table 1493
 - module reference table 1493
 - nonresident names tables 1494–95
 - vs* old 1487
 - resident names table 1492–93
 - resource table 1491–92
 - segment table 1490
- Nishi, Kay 14–15
- NLSFUNC command 1441–42
- Nonmaskable interrupt (NMI) 399, 411, 640. *See also* Interrupt 02H
- NOTEPAD display (Windows) 501–4(fig.)
- NUL device 59, 151
- and CTTY 810

O

- OBJDUMP.C program 1509–12
- Object files 701–2
 - hexadecimal files format 1499–1505
- Object Linker (LINK) 701–21, 757, 981, 993–98, 1004
 - building a .EXE file header 712(table)
 - combine parameters 127–28
 - converting .EXE files produced by, with EXE2BIN 971–73
 - creating .EXE files 620–21
 - creating map files with 1004
 - description of 988–92
 - environmental variables in 931
 - functions of 703
 - LINK intervals 709–12
 - messages 993–98
 - object files, object libraries, and LIB 701–2
 - object module order 703–6
 - operating in .EXE program 111, 113
 - organizing memory with 713–21
 - return codes 992–93
 - segment order/combinations 707–9
- Object module(s) 643–700
 - contents of 645–46
 - dump utility 1509–12
 - linking (*see* Object Linker)
 - object record formats 655–56
 - object records listed 657–700
 - order of 703–6
 - structure of 650–55
 - object record order 651
 - references between records 654–55
 - terminology 646–49
 - translation of assembly programs into relocatable (*see* Microsoft Macro Assembler)
 - types of 650, 651(fig.)
 - typical 651–54
 - use of 643–44
- Object module library file 701–2
 - creating/modifying 980–86
- Object records
 - formats 655–56
 - listed 657–700
 - order 651
 - references between 654–55
 - types 650, 651(fig.)
- Obtain Size of Extended Memory. *See* Interrupt 15H Function 88H
- OFFSET operator (MASM), using on labels in grouped segments 131–32
- Open File with FCB. *See* Interrupt 21H Function 0FH
- Open File with Handle. *See* Interrupt 21H Function 3DH
- Open-loop servomechanism 89
- Open Symbol Map (SYMDEB XO) 1140
- Operating system
 - compatibility issues, MS-DOS and MS OS/2 492–97
 - error codes 495
 - filenames 492–93
 - MS-DOS function calls 493–94
 - multitasking concerns 496–97
 - seeks 495
 - in conventional memory 298
 - three types of 51(table), 52
 - transfer 940
- Operating-system loader 52, 72
- Options menu (CodeView) 1161
- O'Rear, Bob 8(fig.), 15–19
- OS/2 operating system. *See* MS OS/2 operating system
- Output to Port
 - DEBUG O 1042
 - SYMDEB O 1118
- Overflow Trap exception. *See* Interrupt 04H
- OVERLAY.ASM program 342
- Overlays, program 122–23
 - EXEC function and 321, 322–23, 335–43
 - example program 337–42
 - loading and executing 336–37
 - making memory available 335–36
 - preparing parameters 336–37
 - LINK memory organization using 715–18

P

- PAGE alignment 126–27
- Page Fault exception. *See* Interrupt 0EH
- Panners, Nancy 34
- PARA alignment 126
- Parallel port, input/output 163
- PARENT.ASM program 330–34
- Parent program, use of EXEC by 321
 - sample program 330–36
- Parity parameters 892
- Parse Filename. *See* Interrupt 21H Function 29H
- Partition(s)
 - block device 90–92, 858
 - extended, in MS-DOS version 3.3 1458
- Partition table 91, 92
- Pascal (language) 14

- Pascal Compiler, Microsoft, utilities with 974, 977, 980, 987, 1157
- Paterson, Tim 6, 12–13, 16
- PATH command 739, 897–98
 - AUTOEXEC.BAT and 65, 755
 - COMMAND.COM and 65, 783
 - SET and 930, 931
- PATH variable 930
- PAUSE command (BATCH) 67, 753, 766–67
- PC-DOS xix, 27, 55–57, 725
 - basic character devices 151–64
 - commands from, included in MS-DOS version 3.3 1435–36
 - commands only in 725, 785, 925, 948
 - loading 52
 - memory requirements 58
 - versions 55–57
- PC Probe hardware debugging aid 641
- PC ROM BIOS function calls 1513–30. *See also*
 - Interrupt 10H *through* 1AH
- Perform Conditional Execution (IF) 764–65
- Perform Hexadecimal Arithmetic
 - DEBUG H 1035
 - SYMDEB H 1109
- Perform Stack Trace (SYMDEB K) 1111–12
- Peripheral devices supported by MS-DOS 59
- Peripheral support, with MS-DOS kernel 54
- Periscope hardware debugging aid 641
- Peters, Chris 33–34, 39
- PIFEDIT (Windows) 507
- Pipes 53
 - I/O redirection through 67
- POST (power-on self test), and loading MS-DOS 68–72
- Print Character. *See* Interrupt 21H Function 05H
- PRINT command 33, 899–903
 - ASSIGN and 741
- Printer. *See also* PRN
 - configuring 888
 - input/output 163–64
 - redirecting output 894–95
- Printer Services. *See* Interrupt 17H
- Print Screen. *See* Interrupt 05H
- Print Spooler (PRINT) 899–903
 - development in MS-DOS 33
- PRN (printer output) 22, 59, 62, 151, 163–64. *See also*
 - LPT1; LPT2; LPT3
 - CTTY and 810
 - filters and 429
 - opening 76
- Proceed Through Loop or Subroutine
 - DEBUG P 1043
 - SYMDEB P 1119–20
- Process control, with MS-DOS kernel 53
- Process management system calls 1183
- Program(s). *See also* .COM program files; .EXE program files
 - assembling machine instructions for 1024
 - crash protection for 640
 - debugger 1020–23
 - disassembling 1051
 - go execute 1033, 1107
 - loading (*see* EXEC function)
 - overlays (*see* Overlays, program)
 - timing of 491
 - trace execution of 1050, 1130–31
- Program Debugger (DEBUG) 1020–23. *See also*
 - Debugging in MS-DOS; DEBUG utility
- Program Information File (PIF) 500
- Programmable Interrupt Controller. *See* Intel 8259A Programmable Interrupt Controller (PIC); Maskable interrupts
- Program segment(s)
 - controlling .EXE programs with MASM GROUP 131–32
 - controlling .EXE programs with MASM SEGMENT 125–30
 - size reduction of 130
- Program segment prefix (PSP) 1020
 - .EXE programs 108–11
 - file control block functions and 267–68
 - get/set address functions in TSR programs 352
 - inserting filenames/switches into simulated 1040
 - structure 1477
 - warm boot/terminate vector 117–18
- PROMPT command 904–6
 - AUTOEXEC.BAT and 65, 755
 - COMMAND.COM and 65, 783
 - escape sequences in 732
 - SET and 931
- Protected mode
 - compatibility issues 489
 - vs* real mode 58, 316
- PROTOD.ASM character filter program 431–33
- PROTOD.C character filter program 433
- PROTOL.ASM line filter program 434–35
- PROTOL.C line filter program 436
- p-System operating system 26
- PUBDEF Public Names Definition object record 651, 669–71
- PUBLIC* parameter 127

Q

QDOS operating system 12, 27
 QuickBASIC programs 550–55, 567–69, 569–72,
 1503–5
 Quit DEBUG (DEBUG Q) 1044
 Quit EDLIN (EDLIN Q) 845
 Quit SYMDEB (SYMDEB Q) 1121

R

RAMdisk 86
 RAMDRIVE.SYS 907–9
 Random Block Read. *See* Interrupt 21H Function 27H
 Random Block Write. *See* Interrupt 21H Function 28H
 Random Read. *See* Interrupt 21H Function 21H
 Random Write. *See* Interrupt 21H Function 22H
 Range, defined 1058
 Raster operation codes (Windows) 534, 535–36
 Raw versus cooked mode 153–54
 RD command. *See* RMDIR/RD command
 Read Character and Attribute at Cursor. *See* Interrupt
 10H Function 08H
 Read Current Clock Count. *See* Interrupt 1AH
 Function 00H
 Read Cursor Position, Size, and Shape. *See* Interrupt
 10H Function 03H
 Read Data from Cassette. *See* Interrupt 15H
 Function 02H
 Read Date from Real-Time Clock. *See* Interrupt 1AH
 Function 04H
 Read Disk Sectors. *See* Interrupt 13H Function 02H
 Read File or Device. *See* Interrupt 21H Function 3FH
 Read Light-Pen Position. *See* Interrupt 10H
 Function 04H
 Read Long. *See* Interrupt 13H Function 0AH
 Read Next Character. *See* Interrupt 16H Function 00H
 Read Pixel Dot. *See* Interrupt 10H Function 0DH
 Read Real-Time Clock. *See* Interrupt 1AH
 Function 02H
 Read Track on Logical Drive. *See* Interrupt 21H
 Function 44H Subfunction 0DH
 Read/write multiple sectors 24
 Real mode 58, 316
 Reboot Computer (Warm Start). *See* Interrupt 19H
 Recalibrate Drive. *See* Interrupt 13H Function 11H
 Receive Control Data from Block Device. *See*
 Interrupt 21H Function 44H
 Subfunction 04H
 Receive Control Data from Character Device. *See*
 Interrupt 21H Function 44H
 Subfunction 02H
 Receive One Character. *See* Interrupt 14H
 Function 02H
 RECOVER command 910–11
 Recover Files (RECOVER) 910–11
 Redirectable I/O, and filter operation 429–30
 Redirect Printing (MODE) 894–95
 Redirect SYMDEB Input (SYMDEB <) 1143–44
 Redirect SYMDEB Input and Output
 (SYMDEB =) 1146
 Redirect SYMDEB Output (SYMDEB >) 1145
 Redirect Target Program Input (SYMDEB !) 1147
 Redirect Target Program Input and Output
 (SYMDEB ~) 1149
 Redirect Target Program Output (Symdeb !) 1148
 Registers
 AX-extended error code, MS-DOS version 3.3
 1461–62
 BH-error class, MS-DOS version 3.3 1462
 BL-suggested action, MS-DOS version 3.3 1463
 child program execution 328-
 CH-locus, MS-DOS version 3.3 1463
 critical error handling 394–98
 DEBUG initialization 582
 displaying or modifying 1045, 1122
 .EXE program settings 113–15
 expanded memory 310–12
 extended error information 401–2, 404–5
 extended memory 316–19
 INS8250 UART chip 171–80
 maintained by DEBUG 1022
 maintained by SYMDEB 1060–61
 overlay execution 337
 PC 1045
 Relocation pointer table, in .EXE file headers 123
 REM command (BATCH) 67, 753, 768
 Remove Directory. *See* Interrupt 21H Function 3AH
 Remove Directory (RMDIR or RD) 923–24
 Rename File (RENAME or REN). *See* Interrupt 21H
 Function 17H; Interrupt 21H
 Function 56H
 RENAME/REN command 912–13
 REPLACE command 914–17
 Replace Text (EDLIN R) 846–47
 Report If Character Ready. *See* Interrupt 16H
 Function 01H
 Request header, device driver 452–53(fig.)
 device open/close 464(fig.)
 flush input/output status 463(fig.)
 generic IOCTL 466–67(fig.)
 get/set logical device 467–68(fig.)
 initialization 456(fig.)
 input/output status 463(fig.)
 IOCTL Read, Write, Write with Verify 461(fig.)
 media check 458(fig.)

Request header (*continued*)
 nondestructive read 462
 removable media 464(fig.), 464–66
 status word 454(table)

Reset Alarm (Turn Alarm Off). *See* Interrupt 1AH
 Function 07H

Reset Disk System. *See* Interrupt 13H Function 00H

Resize Memory Block. *See* Interrupt 21H
 Function 4AH

Restart System. *See* Interrupt 19H

Restore Backup Files (RESTORE) 918–22

RESTORE command 918–22
 ASSIGN and 741
 BACKUP and 745, 918
 JOIN and 877

RET instruction, terminating .EXE programs
 with 118–19

Return Address of INDOS Flag. *See* Interrupt 21H
 Function 34H

Reynolds, Aaron, in development of MS-DOS 30, 34,
 35, 39, 43

RMDIR/RD command 923–24

ROM BASIC. *See* Interrupt 18H

ROM BIOS 20, 59–60
 loading MS-DOS and 68–72
 location in memory 69(fig.)
 role in display I/O 159
 role in keyboard I/O 156
 system calls 1513–30 (*see also* Interrupts 10H
through 1AH)
 tables 69, 70(fig.)
 TSR interrupt processing 349

ROM monitor operating system 51

ROOT.ASM program 338–42

Root directory 101–3

RS232C signals 170, 171(table)

Run length limited (RLL) encoding 87

Run menu (CodeView) 1160

S

SAMPLE.C program (Windows) 512–17
 display 512(fig.)
 .EXE file construction 518–20
 header 516(fig.)
 make file 517(fig.)
 message processing 527–29
 module-definition file 516–17(fig.)
 program initialization 520–21
 resource script 516
 source code 513–15

Sams, Jack 14

Screen. *See also* Display output
 ANSI.SYS escape sequences to control 731–38
 clearing 781
 controlling 158–59
 graphics mode (*see* Graphics)
 screen output debugging with CodeView
 629–40
 swap 1055, 1150

Scroll Window Down. *See* Interrupt 10H
 Function 07H

Scroll Window Up. *See* Interrupt 10H Function 06H

Search for Text (EDLIN S) 848–49

Search Memory
 DEBUG S 1048–49
 SYMDEB S 1125–26

Search menu (CodeView) 1160

Search path
 defining command 897
 setting with APPEND 739

Seattle Computer Products, and 86-DOS 12–13, 15

Sector, disk 88–89
 loading 1037, 1113
 writing 1052, 1136

Seeks, compatibility issues 495

Seek to Head. *See* Interrupt 13H Function 0CH

SEGDEF Segment Definition object record 651,
 676–79

Segment. *See* Memory segments; Program
 segment(s); Program segment prefix
 (PSP); SEGMENT directive

SEGMENT directive (MASM), to structure .EXE
 programs 125–30
align type parameter 125–27
class type parameter 128–30
combine type parameter 127–28
 ordering segments to shrink .EXE files 130
 sample .EXE program using 132–37

Segment Not Present exception. *See* Interrupt 0BH

Select Active Page. *See* Interrupt 10H Function 05H

Select Code Page function 1454–55

Select Color Palette. *See* Interrupt 10H Function 0BH

SELECT command 925–29
 MS-DOS version 3.3 1435–36

Select Disk. *See* Interrupt 21H Function 0EH

Send Byte to Printer. *See* Interrupt 17H Function 00H

Send Control Data to Block Device. *See* Interrupt 21H
 Function 44H Subfunction 05H

Send Control Data to Character Device. *See* Interrupt
 21H Function 44H Subfunction 03H

Send One Character. *See* Interrupt 14H Function 01H

Sequential Read. *See* Interrupt 21H Function 14H

Sequential Write. *See* Interrupt 21H Function 15H

Serial communications monitoring 556–57
 debugging program 587–600
 demonstration program 557–72

- Serial communications ports 161–62
 - configuring 892–93
 - hardware 171–80
 - programming examples 162
- Serial Port Services. *See* Interrupt 14H
- Servomechanism, open *vs* closed loop 89
- Set Alarm 1530
- Set Block-Device Parameters (DRIVPARM) 797–98
- Set Breakpoints (SYMDEB BP) 1072–73
- SET command 930–32
 - AUTOEXEC.BAT and 65, 755
 - COMMAND.COM and 65, 66, 783
- Set Control-C Check (BREAK) 770–71
- Set Country Code (COUNTRY) 793–94
- Set Current Clock Count. *See* Interrupt 1AH
 - Function 01H
- Set Current Country. *See* Interrupt 21H Function 38H
- Set Cursor Position. *See* Interrupt 10H Function 02H
- Set Cursor Size and Shape. *See* Interrupt 10H
 - Function 01H
- Set Data-File Search Path (APPEND) 739–40
- Set Date (DATE) 811–12, 1268–69
- Set Date in Real-Time Clock. *See* Interrupt 1AH
 - Function 05H
- Set Device Data. *See* Interrupt 21H Function 44H
 - Subfunction 01H
- Set Device Parameters. *See* Interrupt 21H Function
 - 44H Subfunction 0DH
- Set Disk Type. *See* Interrupt 13H Function 17H
- Set Display Mode (MODE) 890–91
- Set DTA Address. *See* Interrupt 21H Function 1AH
- Set Environment Variable (SET) 930–32
- Set Extended Error Information. *See* Interrupt 21H
 - Function 5DH
- Set Handle Count Function 1449–50
- Set Highest Logical Drive (LASTDRIVE) 803
- Set Interrupt Vector. *See* Interrupt 21H Function 25H
- Set Logical Drive Map. *See* Interrupt 21H Function
 - 44H Subfunction 0FH
- Set Maximum Open Files
 - using file control blocks (FCBs) 799–800
 - using handles (FILES) 801–2
- set_mdm()* parameter coding 222(table)
- Set Printer Setup. *See* Interrupt 21H Function 5EH
 - Subfunction 02H
- Set Program Segment Prefix Address. *See* Interrupt
 - 21H Function 50H
- Set Real-Time Clock. *See* Interrupt 1AH Function 03H
- Set Relative Record. *See* Interrupt 21H Function 24H
- Set/Reset Verify Flag. *See* Interrupt 21H
 - Function 2EH
- Set Symbol Value (SYMDEB Z) 1141–42
- Set System Time (TIME) 942–43
- Set Time. *See* Interrupt 21H Function 2DH
- SETUP program 942
- Set Verify Flag (VERIFY) 953
- Set Video Mode. *See* Interrupt 10H Function 00H
- SHARE command 799, 933–34
- Shell 55, 63–68, 76–83. *See also* COMMAND.COM
 - custom 79–83
 - escape to 1154–55
- SHELL.ASM program 81–83
- SHELL command (CONFIG.SYS) 789, 804
 - COMMAND.COM and 65–66
 - replacing COMMAND.COM with a custom shell
 - 79–83
 - SET and 930, 931
- SHIFT command (BATCH) 67, 753, 754, 769
 - with GOTO 762
- Shift Replaceable Parameters (SHIFT) 769
- Single Step exception. *See* Interrupt 01H
- Small memory model 138
- SNAP.ASM program 359–84
 - activating the application 382–83
 - block structure of 381(fig.)
 - code 360–80
 - detecting a hot key 382
 - executing 383–84
 - installing 381–82
- Softcard 11
- SofTech Microsystems 26
- Software. *See also* Application programs; Operating
 - system; Program(s)
 - in the development of MS-DOS 38
 - instrumentation debugging 555–72
 - three layers of 447–48
- Software Bus 86 operating system 27
- Software Development Kit (Windows) 511–12
- SORT command 935–37
- SORT.EXE program 442–46
- Source code
 - displaying mode
 - disabling 1128
 - enabling 1127, 1129
 - displaying source line 1151
 - viewing 1134–35
- SPACE signal 172
- Special characters 879–81
 - Kanji and Hangeul 37
- Specify Command Processor (SHELL) 804
- SPOOLER (Windows) 507
- Stack(s). *See* Internal stacks
- Stack exception. *See* Interrupt 0CH
- STACK parameter 127–28
- STACKS command (CONFIG.SYS) 805
- Stand-alone Disk BASIC 3, 8, 12
- Stop bits 892
- Storage devices 85–103. *See also* Block device(s)

- Storage devices (*continued*)
 - block device layout 86–90
 - file system layout 93–103
 - partition layout 90–92
- Strategy routine (*Strat*), in device drivers 452–53
- Subdirectory 282
 - copying 955
 - substituting drive for 938
- Subroutine, proceed through 1043
- SUBST command 938–39
 - ASSIGN and 741
 - BACKUP and 747
 - CHKDSK and 775
 - DISKCOMP and 818
 - DISKCOPY and 822
 - FORMAT and 866
 - JOIN and 877
 - LABEL and 882
 - MKDIR/MD and 885
 - RMDIR/RD and 923
- Substitute Drive for Subdirectory (SUBST) 938–39
- Suspend Batch-File Execution (PAUSE) 766–67
- Swap Screen (SYMDEB \) 1055, 1150
- Symbol
 - defined 1057
 - set value 1141–42
- Symbol file, for use with with SYMDEB 1004–6
- Symbolic Debugger (SYMDEB). 1054–62 *See also*
 - Debugging in MS-DOS; SYMDEB utility
- Symbol map
 - examining 1138–39
 - opening 1140
- SYMDEB utility 573, 586–618, 115, 1054–62
 - A command 1063–64
 - BC command 1065–66
 - BD command 1067–68
 - BE command 1069–70
 - binary operators 1059
 - BL command 597–98, 608, 1071
 - BP command 597, 608, 1072–73
 - C command 1074
 - commands and actions 1056–57(table)
 - creating symbol file for 1004
 - D command 1075–76
 - DA command 1077–78
 - DB command 1079–80
 - DD command 595, 599, 1081–82
 - debugging C programs with 600–618
 - debugging TSRs with 587–600
 - description 1054–61
 - DL command 1083–84
 - DS command 1085–86
 - DT command 1087–88
 - DW command 1089–90
 - SYMDEB utility (*continued*)
 - E command 1091–92
 - EA command 1093–94
 - EB command 1095–96
 - ED command 1097
 - EL command 1098–99
 - ES command 1100–1101
 - ET command 1102–3
 - EW command 1104
 - examples 1061–62
 - F command 1105–6
 - G command 595, 1107–8
 - H command 1109
 - I command 1110
 - K command 1111–12
 - L command 1113–14
 - MAPSYM and 1004–5
 - M command 1115
 - N command 614, 1116–17, 1136
 - O command 1118
 - P command 1119–20
 - Q command 595, 1121
 - R command 593, 596, 606, 1122–24
 - registers and flags 1060
 - S command 1125–26
 - S+ command 1127
 - S– command 1128
 - S& command 1129
 - T command 594, 598, 1130–31
 - U command 1132–33
 - unary operators 1059
 - V command 1134–35
 - W command 1136–37
 - X command 594, 596, 598–99, 606, 607, 613, 614, 1138–39
 - XO command 598, 612, 1140
 - Z command 598, 612, 1141–42
 - < command 1143–44
 - > command 1145
 - = command 1146
 - { command 1147
 - } command 1148
 - ~ command 1149
 - \ command 1150
 - . command 1151
 - ? command 1152–53
 - ! command 1154–55
 - * command 1156
 - SYS command 940–41
 - ASSIGN and 741
 - SYSINIT 61, 73–76
 - System batch-file interpreter (BATC) 752–69
 - System calls, MS-DOS 1177–84. *See also* Interrupts
 - 20H through 2FH

- System calls (*continued*)
 arranged by functional group 1181–84
 format 1178–81
 PC ROM BIOS 1513–30
 version differences 1177–78
- System configuration and control commands 728
 BREAK 770–71
 COMMAND 782–84
 DATE 811–12
 EXIT 853
 PROMPT 904–6
 SELECT 925–29
 SET 930–32
 SHARE 933–34
 TIME 942–43
 VER 952
- System Configuration File (CONFIG.SYS) 788–89
- System configuration file directives 729–30, 788–89
 BREAK 790
 BUFFERS 791–92
 COUNTRY 793–94
 DEVICE 795–96
 DRIVPARM 797–98
 FCBS 799–800
 FILES 801–2
 LASTDRIVE 803
 SHELL 804
 STACKS 805
- System Startup Batch File (AUTOEXEC.BAT) 755–57
- T**
- Tandy 2000 computer 34
 Tape drive storage 103
 Template, editing buffer 832
 TEMPLATE.ASM character-device driver 471–78
 TERMINAL dialog box (Windows) 505(fig.)
 Terminal emulator CTERM.C 230–46
 Terminate and Stay Resident. *See* Interrupt 21H Function 31H; Interrupt 27H
 Terminate-and-stay-resident utilities 347–84. *See also* Interrupt 21H Function 31H; Interrupt 27H
 APPEND command 739–40
 building instrumentation software for debugging with 556–72
 determining MS-DOS status 353–56
 multiplex interrupt 356–57
 organization in memory 348(fig.)
 programming examples 357–81
 HELLO.ASM 357–59
 SNAP.ASM 359–81
 segment order for 713–14
- Terminate-and-stay-resident utilities (*continued*)
 structure of 275–349
 system calls for 350–53
 using SYMDEB to debug 587–600
- Terminate Command Processor (EXIT) 853
 Terminate Process. *See* Interrupt 21H Function 00H
 Terminate Process with Return Code. *See* Interrupt 21H Function 4CH
 Terminate Program. *See* Interrupt 20H
 Terminate Routine Address. *See* Interrupt 22H
 TESTCOMM.ASM programs 544
 corrected code 580–81
 incorrect code 574–75
- Test for Drive Ready. *See* Interrupt 13H Function 10H
 Text and files (Windows) 536
 Text editor, escape sequences in 732. *See also* EDLIN commands
- THEADR Translator Header object record 651, 657
 TIME command 942–43
- Timer
 setting date 811
 setting time 942
- Timer Tick (user defined). *See* Interrupt 1CH
 Time-slicing 900
 TINYDISK.ASM block-device driver 478–86
 Torode, John 10
- Trace Program Execution
 DEBUG T 1050
 SYMDEB T 1130–31
- Tracks, disk 87, 88(fig.)
 Traf-O-Data machine 5–6
 Transfer Another File (EDLIN T) 850–51
 Transfer Control to ROM-BASIC. *See* Interrupt 18H
 Transfer System Files (SYS) 940–41
 Transient program area (TPA) 79
 in conventional memory 298–99
- TREE command 944–46
 TSR. *See* Terminate-and-stay-resident utilities
 TYPDEF Type Definition object record 651, 665–68
 TYPE command 947
 escape sequences using 732
- U**
- UART. *See* INS8250 Universal Asynchronous Receiver Transmitter (UART)
- Ulloa, Mani 34, 37
 Unary operators, SYMDEB 1059
 Unfiltered Character Input Without Echo. *See* Interrupt 21H Function 07H
- UNIX operating system 68
 directories 284
 file management 30

Update Files (REPLACE) 914-17
 UPPERCAS.C programs 545
 correct code 629(fig.)
 correction of 620-29
 incorrect 620(fig.)

V

VDISK.SYS 948-51
 VER command 952
 VERIFY command 953
 Verify Disk Sectors. *See* Interrupt 13H Function 04H
 Verify flag, set 953
 Verify Track on Logical Drive. *See* Interrupt 21H
 Function 44H Subfunction 0DH
 Version, display 952
 Victor Corporation 35
 Video. *See* Character-device input/output; Display
 output; Screen
 Video Graphics Array (VGA) 157
 Video Parameter Pointer. *See* Interrupt 1DH
 Video Services. *See* Interrupt 10H
 View menu (CodeView) 1160
 View Source Code (SYMDEB V) 1134-35
 Virtual Disk (RAMDRIVE.SYS) 907-9
 Virtual Disk (VDISK.SYS) 948-51
 VOL command 954
 Volume label(s) 103, 283-84
 displaying 954
 modifying 882
 program example for updating 292-96

W

Wallace, Bob 8(fig.)
 Warm boot 68
 Warm Boot/Terminate vector 117-18
 Watch menu (CodeView) 1161
 Watchpoints 619
 Wildcard(s)
 COPY 806
 DEL/ERASE 813
 DIR 816
 directory searches 286-87
 REPLACE 914
 RESTORE 918
 Window-Oriented Debugger (CodeView). 1157-73
 See also CodeView utility; Debugging
 in MS-DOS

Windows 499-538
 application and utility programs in 506-7
 data sharing/data exchange
 Clipboard 537-38
 dynamic data exchange 538
 display 500-505
 dialog boxes 504-5
 parts of the window 501-4
 graphics device interface 529-37
 internationalization 538
 memory management 510-11
 MS-DOS Executive 505, 506(fig.)
 multitasking 529
 new executable header 1487-97
 program categories 499-500
 structure of 507-10
 libraries and programs 509-10
 modules 507-9
 structure of a program 511-29
 message processing 525-26
 message processing example 527-29
 messages 524-25
 messaging system 522-24
 program components 512-17
 program construction 518-20
 program initialization 520-21
 software development kit 511-12
 Wood, Marla 8(fig.)
 Wood, Steve 8(fig.)
 Word(s), 16-bit 172, 222
 displaying 1089-90
 entering 1104
 WORD alignment 126
 Wrap around, screen display 733
 Write Character and Attribute. *See* Interrupt 10H
 Function 09H
 Write Character as TTY. *See* Interrupt 10H
 Function 0EH
 Write Character Only. *See* Interrupt 10H
 Function 0AH
 Write Character String. *See* Interrupt 10H
 Function 13H
 Write Data to Cassette. *See* Interrupt 15H
 Function 03H
 Write Disk Sectors. *See* Interrupt 13H Function 03H
 Write File or Device. *See* Interrupt 21H
 Function 40H
 Write File or Sectors
 DEBUG W 586-87, 1052-53
 SYMDEB W 1136-37
 Write Lines to Disk (EDLIN W) 852
 Write Long. *See* Interrupt 13H Function 0BH
 Write Pixel Dot. *See* Interrupt 10H Function 0CH
 Write Track on Logical Drive. *See* Interrupt 21H
 Function 44H Subfunction 0DH

X

XCOPY command 955-59
ATTRIB and 743
DISKCOPY and 822
XENIX operating system 30, 31, 68
directories 284
XON/XOFF 168

Z

Zbikowski, Mark, in the development of MS-DOS 30,
34, 35, 37, 39, 43
Z-DOS operating system 27

Commands and System Calls

This index lists only primary command and system call entries. Please use the Subject Index for related entries.

SYMBOLS

@ (BATCH) 1434

A

ANSI.SYS 731-38
 APPEND 739-40, 1436-37
 ASSIGN 741-42
 ATTRIB 743-44, 1437
 AUTOEXEC.BAT (BATCH) 755-57

B

BACKUP 745-51, 1437
 BATCH 752-69, 1434-35
 BREAK 770-71
 BREAK (CONFIG.SYS) 790
 BUFFERS (CONFIG.SYS) 791-92

C

CALL (BATCH) 1434-35
 CD 772-73
 CHCP 1440
 CHDIR 772-73
 CHKDSK 774-80
 CLS 781
 CodeView utility 1157-73
 COMMAND 782-84
 COMP 785-87, 1435
 CONFIG.SYS 788-805
 COPY 806-9
 COUNTRY (CONFIG.SYS) 793-94, 1442-43
 CREF utility 967-70
 CTTY 810

D

DATE 811-12
 DEBUG, general 1020-23
 DEBUG utility 1020-53
 A command 1024-25
 C command 1026
 D command 1027-28
 E command 1029-30
 F command 1031-32
 G command 1033-34
 H command 1035
 I command 1036
 L command 1037-38
 M command 1039
 N command 1040-41
 O command 1042
 P command 1043
 Q command 1044
 R command 1045-47
 S command 1048-49
 T command 1050
 U command 1051
 W command 1052-53
 DELETE 813-14
 DEVICE (CONFIG.SYS) 795-96, 1443-45
 DIR 815-17
 DISKCOMP 818-21
 DISKCOPY 822-25
 DRIVER.SYS 826-28
 DRIVPARM (CONFIG.SYS) 797-98

E

ECHO (BATCH) 758-59
 EDLIN, general 829-31
 EDLIN line editor 829-52
 A command 834
 C command 835-36
 D command 837-38
 E command 839

EDLIN line editor (*continued*)

I command 840
 L command 841
linenumber command 832-33
 M command 842-43
 P command 844
 Q command 845
 R command 846-47
 S command 848-49
 T command 850-51
 W command 852
 ERASE 813-14
 EXE2BIN utility 971-73
 EXEMOD utility 974-76
 EXEPACK utility 977-79
 EXIT 853

F

FASTOPEN 1433-34
 FC 854-57
 FCBS (CONFIG.SYS) 799-800
 FDISK 858-62, 1437
 FILES (CONFIG.SYS) 801-2
 FIND 863-64
 FOR (BATCH) 760-61
 FORMAT 865-71

G

GOTO (BATCH) 762-63
 GRAFTABL 872-73, 1445
 GRAPHICS 874-76

I

IF (BATCH) 764-65
 Interrupt 10H, Video Services 1513-18
 Function 00H, Set Video Mode 1513
 Function 01H, Set Cursor Size and Shape 1514
 Function 02H, Set Cursor Position 1514
 Function 03H, Read Cursor Position, Size, and Shape 1514
 Function 04H, Read Light-Pen Position 1514-15
 Function 05H, Select Active Page 1515
 Function 06H, Scroll Window Up 1515
 Function 07H, Scroll Window Down 1515

Interrupt 10H (*continued*)

 Function 08H, Read Character and Attribute at Cursor 1515-16
 Function 09H, Write Character and Attribute 1516
 Function 0AH, Write Character Only 1516
 Function 0BH, Select Color Palette 1516
 Function 0CH, Write Pixel Dot 1517
 Function 0DH, Read Pixel Dot 1517
 Function 0EH, Write Character as TTY 1517
 Function 0FH, Get Current Video Mode 1517
 Function 13H, Write Character String 1518
 Interrupt 11H, Get Peripheral Equipment List 1518
 Interrupt 12H, Get Usable Memory Size (KB) 1519
 Interrupt 13H, Disk Services 1519-23
 Function 00H, Reset Disk System 1519
 Function 01H, Get Disk Status 1519-20
 Function 02H, Read Disk Sectors 1520
 Function 03H, Write Disk Sectors 1520
 Function 04H, Verify Disk Sectors 1520
 Function 05H, Format Disk Tracks 1520
 Function 08H, Get Current Drive Parameters 1520-21
 Function 09H, Initialize Hard-Disk Parameter Table 1521
 Function 0AH, Read Long 1521
 Function 0BH, Write Long 1521
 Function 0CH, Seek to Head 1521
 Function 0DH, Alternate Disk Reset 1522
 Function 10H, Test for Drive Ready 1522
 Function 11H, Recalibrate Drive 1522
 Function 14H, Controller Diagnostic 1522
 Function 15H, Get Disk Type 1522-23
 Function 16H, Check for Change of Floppy-Disk Status 1523
 Function 17H, Set Disk Type 1523
 Interrupt 14H, Serial Port Services 1523-25
 Function 00H, Initialize Port Parameters 1523-24
 Function 01H, Send One Character 1524
 Function 02H, Receive One Character 1524
 Function 03H, Get Port Status 1524-25
 Interrupt 15H, Miscellaneous System Services 1525-26
 Function 00H, Turn On Cassette Motor 1525
 Function 01H, Turn Off Cassette Motor 1525
 Function 02H, Read Data from Cassette 1525-26
 Function 03H, Write Data to Cassette 1526
 Interrupt 16H, Keyboard Services 1526-27
 Function 00H, Read Next Character 1526
 Function 01H, Report If Character Ready 1527
 Function 02H, Get Shift Status 1527
 Interrupt 17H, Printer Services 1527-28

Interrupt 17H (*continued*)

- Function 00H, Send Byte to Printer 1527
- Function 01H, Initialize Printer 1528
- Function 02H, Get Printer Status 1528
- Interrupt 18H, Transfer Control to ROM-BASIC 1528
- Interrupt 19H, Reboot Computer (Warm Start) 1528
- Interrupt 1AH, Get and Set Time 1528–30
 - Function 00H, Read Current Clock Count 1528–29
 - Function 01H, Set Current Clock Count 1529
 - Function 02H, Read Real-Time Clock 1529
 - Function 03H, Set Real-Time Clock 1529
 - Function 04H, Read Date from Real-Time Clock 1529–30
 - Function 05H, Set Date in Real-Time Clock 1530
 - Function 06H, Set Alarm 1530
 - Function 07H, Reset Alarm (Turn Alarm Off) 1530
- Interrupt 20H, Terminate Program 1185–86
- Interrupt 21H, MS-DOS function calls 1187–1416
 - Function 00H, Terminate Process 1187–88
 - Function 01H, Character Input with Echo 1189–90
 - Function 02H, Character Output 1191–92
 - Function 03H, Auxiliary Input 1193–94
 - Function 04H, Auxiliary Output 1195–96
 - Function 05H, Print Character 1197–98
 - Function 06H, Direct Console I/O 1199–1200
 - Function 07H, Unfiltered Character Input Without Echo 1201–2
 - Function 08H, Character Input Without Echo 1203–4
 - Function 09H, Display String 1205–6
 - Function 0AH, Buffered Keyboard Input 1207–8
 - Function 0BH, Check Keyboard Status 1209–10
 - Function 0CH, Flush Buffer, Read Keyboard 1211–12
 - Function 0DH, Disk Reset 1213–14
 - Function 0EH, Select Disk 1215–16
 - Function 0FH, Open File with FCB 1217–19
 - Function 10H, Close File with FCB 1220–21
 - Function 11H, Find First File 1222–24
 - Function 12H, Find Next File 1225–26
 - Function 13H, Delete File 1227–28
 - Function 14H, Sequential Read 1229–30
 - Function 15H, Sequential Write 1231–32
 - Function 16H, Create File with FCB 1233–34
 - Function 17H, Rename File 1235–36
 - Function 19H, Get Current Disk 1237
 - Function 1AH, Set DTA Address 1238–39
 - Function 1BH, Get Default Drive Data 1240–41
 - Function 1CH, Get Drive Data 1242–44
 - Function 21H, Random Read 1245–46
 - Function 22H, Random Write 1247–48

Interrupt 21H (*continued*)

- Function 23H, Get File Size 1249–50
- Function 24H, Set Relative Record 1251–52
- Function 25H, Set Interrupt Vector 1253–54
- Function 26H, Create New Program Segment Prefix 1255–56
- Function 27H, Random Block Read 1257–59
- Function 28H, Random Block Write 1260–62
- Function 29H, Parse Filename 1263–65
- Function 2AH, Get Date 1266–67
- Function 2BH, Set Date 1268–69
- Function 2CH, Get Time 1270–71
- Function 2DH, Set Time 1272–73
- Function 2EH, Set/Reset Verify Flag 1274–75
- Function 2FH, Get DTA Address 1276
- Function 30H, Get MS-DOS Version Number 1277–78
- Function 31H, Terminate and Stay Resident 1279–80
- Function 33H, Get/Set Control-C Check Flag 1281–82
- Function 34H, Return Address of InDOS Flag 1283
- Function 35H, Get Interrupt Vector 1284
- Function 36H, Get Disk Free Space 1285–86
- Function 38H, Get/Set Current Country 1287–90
 - Get Current Country 1287–89
 - Set Current Country 1290
- Function 39H, Create Directory 1291–92
- Function 3AH, Remove Directory 1293–94
- Function 3BH, Change Current Directory 1295–96
- Function 3CH, Create File with Handle 1297–99
- Function 3DH, Open File with Handle 1300–1303
- Function 3EH, Close File 1304–5
- Function 3FH, Read File or Device 1306–7
- Function 40H, Write File or Device 1308–9
- Function 41H, Delete File 1310–11
- Function 42H, Move File Pointer 1312–14
- Function 43H, Get/Set File Attributes 1315–16
- Function 44H, IOCTL 1317–18
 - Subfunction 00H, Get Device Data 1319–21
 - Subfunction 01H, Set Device Data 1322–23
 - Subfunction 02H, Receive Control Data from Character Device 1324–25
 - Subfunction 03H, Send Control Data to Character Device 1324–25
 - Subfunction 04H, Receive Control Data from Block Device 1326–28
 - Subfunction 05H, Send Control Data to Block Device 1326–28
 - Subfunction 06H, Check Input Status 1329–30

Interrupt 21H (*continued*)

- Function 44H (*continued*)
 - Subfunction 07H, Check Output Status 1329–30
 - Subfunction 08H, Check If Block Device Is Removable 1331–32
 - Subfunction 09H, Check If Block Device Is Remote 1333–34
 - Subfunction 0AH, Check If Handle Is Remote 1335–36
 - Subfunction 0BH, Change Sharing Retry Count 1337–38
 - Subfunction 0CH, Generic I/O Control for Handles 1339–40, 1455–58
 - Subfunction 0DH, Generic I/O Control for Block Devices 1341–42
 - Subfunction 0DH, minor code 40H, Set Device Parameters 1343–46
 - Subfunction 0DH, minor code 41H, Write Track on Logical Drive 1350–51
 - Subfunction 0DH, minor code 42H, Format and Verify Track on Logical Drive 1352–53
 - Subfunction 0DH, minor code 60H, Get Device Parameters 1347–49
 - Subfunction 0DH, minor code 61H, Read Track on Logical Drive 1350–51
 - Subfunction 0DH, minor code 62H, Verify Track on Logical Drive 1352–53
 - Subfunction 0EH, Get Logical Drive Map 1354–55
 - Subfunction 0FH, Set Logical Drive Map 1354–55
- Function 45H, Duplicate File Handle 1356–57
- Function 46H, Force Duplicate File Handle 1358–59
- Function 47H, Get Current Directory 1360–61
- Function 48H, Allocate Memory Block 1362–63
- Function 49H, Free Memory Block 1364–65
- Function 4AH, Resize Memory Block 1366–67
- Function 4BH, Load and Execute Program (EXEC) 1368–74
- Function 4CH, Terminate Process with Return Code 1375–76
- Function 4DH, Get Return Code of Child Process 1377–78
- Function 4EH, Find First File 1379–81
- Function 4FH, Find Next File 1382–84
- Function 54H, Get Verify Flag 1385
- Function 56H, Rename File 1386–87
- Function 57H, Get/Set Date/Time of File 1388–90
- Function 58H, Get/Set Allocation Strategy 1391–92

Interrupt 21H (*continued*)

- Function 59H, Get Extended Error Information 1393–96
- Function 5AH, Create Temporary File 1397–98
- Function 5BH, Create New File 1399–1400
- Function 5CH, Lock/Unlock File Region 1401–3
- Function 5EH, Network Machine Name/Printer Setup 1404–6
 - Subfunction 00H, Get Machine Name 1404
 - Subfunction 02H, Set Printer Setup 1405–6
 - Subfunction 03H, Get Printer Setup 1405–6
- Function 5FH, Get/Make Assign-List Entry 1407–12
 - Subfunction 02H, Get Assign-List Entry 1407–8
 - Subfunction 03H, Make Assign-List Entry 1409–10
 - Subfunction 04H, Cancel Assign-List Entry 1411–12
- Function 62H, Get Program Segment Prefix Address 1413–14
- Function 63H, Get Lead Byte Table 1415–16
- Function 65H, Get Extended Country Information 1451–54
- Function 66H, Select Code Page 1454–55
- Function 67H, Set Handle Count 1449–50
- Function 68H, Commit File Function 1450–51
- Interrupt 22H, Terminate Routine Address 1417
- Interrupt 23H, Control-C Handler Address 1418
- Interrupt 24H, Critical Error Handler Address 1419–21
- Interrupt 25H, Absolute Disk Read 1422–23
- Interrupt 26H, Absolute Disk Write 1424–25
- Interrupt 27H, Terminate and Stay Resident 1426–27
- Interrupt 2FH, Multiplex Interrupt 1428–29

J, K, L

- JOIN 877–78
- KEYB 1440–41
- KEYBxx 879–81
- LABEL 882–84
- LASTDRIVE (CONFIG.SYS) 803
- LIB utility 980–86
- LINK utility 987–98

M

- MAKE utility 999–1003
- MAPSYM utility 1004–6

MASM utility 1007-19
 MD 885-86
 MKDIR 885-86
 MODE 887-95, 1446-47
 MORE 896

N, P

NLSFUNC 1441-42
 PATH 897-98
 PAUSE (BATCH) 766-67
 PRINT 899-903
 Programming Utilities (Introduction) 963-65
 PROMPT 904-6

R

RAMDRIVE.SYS 907-9
 RD 923-24
 RECOVER 910-11
 REM (BATCH) 768
 REN 912-13
 RENAME 912-13
 REPLACE 914-17
 RESTORE 918-22
 RMDIR 923-24

S

SELECT 925-29, 1435-36
 SET 930-32
 SHARE 933-34
 SHELL (CONFIG.SYS) 804
 SHIFT (BATCH) 769
 SORT 935-37
 STACKS (CONFIG.SYS) 805
 SUBST 938-39
 SYMDEB, general 1054-62
 SYMDEB utility 1054-1156
 A command 1063-64
 BC command 1065-66
 BD command 1067-68
 BE command 1069-70
 BL command 1071
 BP command 1072-73
 C command 1074
 D command 1075-76

SYMDEB utility (*continued*)

 DA command 1077-78
 DB command 1079-80
 DD command 1081-82
 DL command 1083-84
 DS command 1085-86
 DT command 1087-88
 DW command 1089-90
 E command 1091-92
 EA command 1093-94
 EB command 1095-96
 ED command 1097
 EL command 1098-99
 ES command 1100-1101
 ET command 1102-3
 EW command 1104
 F command 1105-6
 G command 1107-8
 H command 1109
 I command 1110
 K command 1111-12
 L command 1113-14
 M command 1115
 N command 1116-17
 O command 1118
 P command 1119-20
 Q command 1121
 R command 1122-24
 S command 1125-26
 S+ command 1127
 S- command 1128
 S& command 1129
 T command 1130-31
 U command 1132-33
 V command 1134-35
 W command 1136-37
 X command 1138-39
 XO command 1140
 Z command 1141-42
 < command 1143-44
 > command 1145
 = command 1146
 { command 1147
 } command 1148
 ~ command 1149
 \ command 1150
 . command 1151
 ? command 1152-53
 ! command 1154-55
 * command 1156
 SYS 940-41
 System Calls (Introduction) 1177-84
 format of entries 1178-81

System Calls (*continued*)
by functional group 1181-84
version differences 1177-78

T, U

TIME 942-43
TREE 944-46
TYPE 947
User Commands (Introduction) 725-30
by functional group 728-30
key to entries 726-27

V, X

VDISK.SYS 948-51
VER 952
VERIFY 953
VOL 954
XCOPY 955-59

Book Design by The NBBJ Group, Seattle, Washington

Cover Design by Greg Hickman

Principal Typography by Carol L. Luke

The manuscript for this book was prepared and submitted to Microsoft Press in electronic form. Text files were processed and formatted using Microsoft Word.

Text composition by Microsoft Press in Garamond with display in Garamond Bold using the Magna composition system and the Linotronic 300 laser imagesetter.

Praise for The MS-DOS® Encyclopedia:

“A superb, nearly inexhaustible reference work.... Anyone serious about programming for MS-DOS will not want to be without [THE MS-DOS ENCYCLOPEDIA].”

Online Today

“The ultimate authority.”

Reference & Research Book News

“A splendid volume.”

Dr. Dobb's Journal of Software Tools

“For those with any technical involvement in the PC industry, this is the one and the only volume worth reading.” *PC WEEK*

“If you like the idea of a one-stop DOS reference book, then this book is for you.” *PC Magazine*

“There's no doubting that this is a superb reference work on MS-DOS.”

EXE magazine

Here, from Microsoft Press, is the ultimate resource for writing, maintaining, and upgrading well-behaved, efficient, reliable, and robust MS-DOS programs. Covering all MS-DOS releases through version 3.2, with a special section on version 3.3, this encyclopedia is *the* standard reference for the working community of MS-DOS programmers and for anyone making strategic decisions about MS-DOS implementation. Included are version-specific technical data and descriptions for:

- More than 100 system calls—each accompanied by C-callable assembly-language routines and programmer's notes
- More than 90 user commands—the most comprehensive version-specific analysis ever assembled
- Key MS-DOS programming utilities and debuggers

THE MS-DOS ENCYCLOPEDIA has hundreds of hands-on examples and thousands of lines of great sample code plus in-depth articles on debugging, writing filters, installable device drivers, TSRs, Windows, memory management, the future of MS-DOS, and much more. There are also more than a dozen appendixes, an index to commands and system calls, and a subject index. THE MS-DOS ENCYCLOPEDIA was researched and written by a team of MS-DOS experts—many involved in the creation and development of MS-DOS—so you know it's accurate and authoritative.

U.S.A. **\$69.95**
U.K. **£48.95**
Austral. **\$104.95**
(recommended)

