DOS INTERRUPTS

Contents

The Legal Words

Please redistribute the following files as a group:

INTERRUP.LST	this file
INTERRUP.SUM	a one-line-per-function summary
INTERRUP.PRI	a brief introduction to interrupts
INTERRUP.1ST	the read-me file, containing credits

Release 88.8 Last change 10/29/88

If you know of any information which is not in this list, or which is incorrect, please let me know!

Ralf Brown

ARPA: ralf@cs.cmu.edu \
UUCP: {ucbvax,harvard}!cs.cmu.edu!ralf > preferred
BIT: ralf%cs.cmu.edu@cmuccvma /
FIDO: Ralf Brown 1:129/31

I reply to all submissions and inquiries, but some of my replies bounce because of bad return paths. If you don't get a response from me within a week, send it again with a better return path.

Key to system abbreviations (unless otherwise indicated, a function is available on all systems)

PC	IBM PC
XT	IBM PC XT
PORT	IBM PC Portable (uses same BIOS as XT)
Jr	IBM PCjr
AT	IBM PC AT
XT2	IBM PC XT 2
XT286	IBM PC XT/286
CONV	IBM Convertible
PS	IBM PS/2, any model
PS30	IBM PS/2 Model 30 and below
PS50+	IBM PS/2 Model 50,60,70,80
CGA	Color Graphics Adapter
EGA	Enhanced Graphics Adapter
VGA	Video Graphics Array
MCGA	Multi-Color Graphics Array
TopView	TopView/DESQview/TaskView/other TopView-compatible environments

To keep the lawyers happy:

The use of -> instead of = signifies that the indicated register or register pair contains a pointer to the specified item

The Interrupts

INT 00 - internal - DIVIDE ERROR

Automatically called at end of DIV or IDIV operation that results in error or overflow. Normally set by DOS to display an error message and abort the program.

INT 01 - internal - SINGLE-STEP

Generated at end of each machine instruction if TF bit in FLAGS is set. This is what makes the T command of DEBUG work for single-stepping. It is not generated after MOV to segment register or POP of segment register (unless you have a very early 8088 with a microcode bug).

INT 02 - hardware - NMI (NON-MASKABLE INTERRUPT)

Generated by NMI signal in hardware. This signal has various uses:

Parity error: all except Jr and CONV
Coprocessor interrupt: all except Jr and CONV
Keyboard interrupt: Jr, CONV
I/O channel check: CONV, PS50+
Disk-controller power-on request: CONV
System suspend: CONV
Real-time clock: CONV
System watch-dog timer, time-out interrupt: PS50+
DMA timer time-out interrupt: PS50+

INT 03 - ONE-BYTE INTERRUPT

Generated by opcode CCh. Generally used to set breakpoints for debuggers. Also used by Turbo Pascal versions 1,2,3 when {$U+} specified

INT 04 - internal - OVERFLOW

Generated by INTO instruction if OF flag is set. If flag is not set, INTO is effectively a NOP. Used to trap any arithmetic errors before the erroneous results propagate further through the computation.

INT 05 - PRINT-SCREEN KEY

Automatically called by keyboard scan when print-screen key is pressed. Normally executes routine to print the screen, but may call any routine that can safely be executed from inside the keyboard scanner. Status and result byte for default handler is at address 0050:0000.

INT 05 - internal - BOUND CHECK FAILED (80186/80286)

Generated by BOUND instruction when the value to be tested is less than the indicated lower bound or greater than the indicated upper bound.

INT 06 - internal - UNDEFINED OPCODE (80286)

INT 07 - internal - NO MATH UNIT AVAILABLE (80286)

INT 08 - IRQ0 - TIMER INTERRUPT

Generated 18.2 times per second, this interrupt is used to keep the time-of-day clock updated.

INT 08 - internal - DOUBLE FAULT (80286 protected mode)

Called when multiple exceptions occur on one instruction, or an exception occurs in an exception handler. If an exception occurs in the double fault handler, the CPU goes into SHUTDOWN mode (which circuitry in the PC/ATconverts to a reset).

INT 09 - IRQ1 - KEYBOARD INTERRUPT

Generated when data is received from the keyboard. This is normally a scan code, but may also be an ACK or NAK of a command on AT-class keyboards.

INT 09 - internal - MATH UNIT PROTECTION FAULT (80286 protected mode)

INT 0A - IRQ2 - EGA VERTICAL RETRACE

Notes: on the Tandy 1000, this interrupt is used by the hard disk the TOPS and PCnet adapters use this interrupt request line by default

INT 0A - internal - INVALID TASK STATE SEGMENT (80286 protected-mode)

INT 0B - IRQ3 - COM2 INTERRUPT

Note: the TOPS and PCnet adapters use this interrupt request line as an alternate
Note: on PS/2's, COM2 through COM8 share this interrupt on many PC's, COM4 shares this interrupt

INT 0B - internal - NOT PRESENT (80286 protected-mode)

Generated when loading a segment register if the segment descriptor indicates that the segment is not currently in memory. May be used to implement virtual memory.

INT 0C - IRQ4 - COM1 INTERRUPT

Note: on many PC's, COM3 shares this interrupt

INT 0C - internal - STACK FAULT (80286 protected-mode)

Generated on stack overflow/underflow. Note that the 80286 will shut down in real mode if SP=1 before a push.

INT 0D - IRQ5 - FIXED DISK (PC), LPT2 (AT/PS)

Note: the Tandy 1000 uses this line for the 60Hhz RAM refresh

INT 0D - internal - GENERAL PROTECTION VIOLATION (80286)

Called in real mode when an instruction attempts to access a word operand located at offset FFFFh

INT 0E - IRQ6 - DISKETTE INTERRUPT

Generated by floppy disk controller on completion of an operation

INT 0E - internal - PAGE FAULT (80386 native mode)

INT 0F - IRQ7 - PRINTER INTERRUPT

Generated by the LPT1 printer adapter when printer becomes ready.
Note: most printer adapters do not reliably generate this interrupt.

INT 10 - internal - COPROCESSOR ERROR (80286/80386)

Generated by the CPU when the -ERROR pin is asserted by the coprocessor. AT's and clones usually wire the coprocessor to use IRQ13, but not all get it right.

INT 10 - AH = 00h VIDEO - SET VIDEO MODE

AL = mode (graphics mode if graphics resolution listed)

	text resol	pixel box	graphic resoltn	color	disp page	scrn addr	system
00h =	40x25	8x8		B&W	8	B800	CGA
=	40x25	8x14		B&W	8	B800	ATI VIP
01h =	40x25	8x8		16	8	B800	CGA
=	40x25	8x14		16	8	B800	ATI VIP
02h =	80x25	8x8		B&W	4	B800	CGA
=	80x25	8x8		B&W	8	B800	EGA/MCGA/VGA
=	80x25	8x14		B&W	8	B800	ATI VIP
03h =	80x25	8x8		16	4	B800	CGA
=	80x25	8x8		16	8	B800	EGA/MCGA/VGA
04h =	40x25	8x8	320x200	4	1	B800	CGA
05h =	40x25	8x8	320x200	4 gray	1	B800	CGA
06h =	80x25	8x8	640x200	B&W	1	B800	CGA
07h =	80x25	9x14		mono	1	B000	MDA/Hercules
=	80x25				8		EGA/VGA
=	80x25	9x14		mono		B000	ATI VIP
08h =	20x25	8x8	160x200	16		B800	PCjr/Tandy 1000
09h =	40x25	8x8	320x200	16		B800	PCjr/Tandy 1000
0Ah	80x25	8x8	640x200	4		B800	PCjr/Tandy 1000
0Bh = reserved (used internally by EGA BIOS)							
0Ch = reserved (used internally by EGA BIOS)							
0Dh =	40x25	8x8	320x200	16	8	A000	EGA/VGA
0Eh =	80x25	8x8	640x200	16	4	A000	EGA/VGA
0Fh =	80x25	8x14	640x350	mono	2	A000	EGA/VGA
10h =	80x25	8x14	640x350	4or16	2	A000	EGA/VGA
11h =	80x30	8x16	640x480	mono		A000	VGA/MCGA/ATI EGA/AT VIP
12h =	80x30	8x16	640x480	16		A000	VGA/ATI VIP
=	80x30	8x16	640x480	16/64		A000	ATI EGA Wonder
13h =	40x25	8x8	320x200	256		A000	VGA/MCGA/ATI VIP
14h =	80x25	8x8	640x200				Lava Chrome II EGA
15h =	80x25	8x14	640x350				Lava Chrome II EGA
16h =	80x25	8x14	640x350				Lava Chrome II EGA
17h =	80x34	8x14	640x480				Lava Chrome II EGA
18h =	132x44	8x8		mono			Tseng Labs EVA
=	80x34	8x14	640x480				Lava Chrome II EGA
19h =	132x25	8x14		mono			Tseng Labs EVA
1Ah =	132x28	8x13		mono			Tseng Labs EVA
22h =	132x44	8x8					Tseng, Ahead
23h =	132x25	6x14					Tseng Labs EVA
=	132x25	8x14					Ahead Systems EGA2001
=	132x25	8x8		16		B800	ATI EGA Wonder/ATI VIP
24h =	132x28	6x13					Tseng Labs EVA
25h =	80x60	8x8	640x480				Tseng Labs EVA
=	640x480			16			VEGA VGA
26h =	80x60	8x8					Tseng Labs EVA
=	80x60	8x8 640x480					Ahead Systems EGA2001
27h =	720x512			16			VEGA VGA
=	132x25 8x8			mono		B000	ATI EGA Wonder/ATI VIP
28h =	???x???						VEGA VGA
29h =	800x600			16			VEGA VGA
2Dh =	640x350			256			VEGA VGA
2Eh =	640x480			256			VEGA VGA
2Fh =	720x512			256			VEGA VGA
30h =	800x600			256			VEGA VGA
=	???x???					B800	AT&T 6300

33h =	132x44	8x8		16		B800	ATI EGA Wonder/ATI VIP
36h =	960x720			16			VEGA VGA
37h =	1024x768			16			VEGAVGA
=	132x44	8x8		mono		B000	ATI EGA Wonder/ATI VIP
40h =	80x25	8x16	640x400	2	1	B800	AT&T 6300, Compaq Portable
=	80x43						VEGA VGA
41h =			640x200	16	1		AT&T 6300
=	132x25						VEGA VGA
42h =	80x25	8x16	640x400	16			AT&T 6300
=	132x43						VEGA VGA
43h = unsupported 640x200 of 640x400 viewport							AT&T 6300
=	80x60						VEGA VGA
44h = disable VDC and DEB output							AT&T 6300
=	100x60						VEGA VGA
48h =	80x50	8x8	640x400		2	B800	AT&T 6300
49h =	80x30	8x16	640x480				Lava Chrome II EGA
4Dh =	120x25						VEGA VGA
4Eh =	120x43						VEGA VGA
4Fh =	132x25						VEGA VGA
50h =	132x25	9x14		mono			Ahead Systems EGA2001
=	80x30	8x16	640x480	16			Paradise EGA-480
=	80x43			mono			VEGA VGA
=			640x480	mono???			Taxan 565 EGA
=	80x34						Lava Chrome II EGA
51h =	80x30	8x16					Paradise EGA-480
=	132x25			mono			VEGA VGA
=	80x34	8x14	640x480	16			ATI EGA Wonder
=	80x30						Lava Chrome II EGA
52h =	132x44	9x8		mono			Ahead Systems EGA2001
=	132x43			mono			VEGA VGA
=	94x29	8x14	752x410	16			ATI EGA Wonder
=	80x60						Lava Chrome II EGA
53h =	100x40	8x14	800x560	16			ATI EGA Wonder/ATI VIP
=	132x43						Lava Chrome II EGA
54h =	132x43	8x8					Paradise EGA-480
=	132x43	7x9	16/256k			B800	Paradise VGA
=	132x43	8x9	16/256k			B800	Paradise VGA on multisync
=	132x43						Taxan 565 EGA
=	100x42	8x14	800x600	16		A000	ATI EGA Wonder
=	132x25						Lava Chrome II EGA
55h =	132x25	8x14					Paradise EGA-480
=	132x25	7x16	16/256k			B800	Paradise VGA
=	132x25	8x16	16/256k			B800	Paradise VGA on multisync
=	132x25						Taxan 565 EGA
=	80x66	8x8		16/256k		A000	ATI VIP
=	94x29	8x14	752x410				Lava Chrome II EGA
56h =	132x43	8x8	3???	2		B000	NSI Smart EGA+
=	132x43	7x9		4		B000	Paradise VGA
=	132x43	8x9		4		B000	Paradisk VGA on multisync
=	132x43			mono			Taxan 565 EGA
57h =	132x25	8x14	3???	4		B000	NSI Smart EGA+
=	132x25	7x16		4		B000	Paradise VGA
=	132x25	8x16		4		B000	Paradise VGA on multisync
=	132x25			mono			Taxan 565 EGA
58h =	100x75	8x8	800x600	16/256k		A000	Paradise VGA
=	80x33	8x14		16		B800	ATI EGA Wonder/ATI VIP
59h =	100x75	8x8	800x600	2		A000	Paradise VGA
=	80x66	8x8	16/256k			A000	ATI VIP
5Eh =			640x400	256			Paradise VGA,VEGA VGA
5Fh =			640x480	256			Paradise VGA
60h =	80x???		???x400				Corona/Cordata BIOS v4.10+
=			752x410				VEGA VGA
61h =			???x400				Corona/Cordata BIOS v4.10+
=			720x540				VEGA VGA
62h =			800x600				VEGA VGA
70h = extended mode set (see below)			Everex Micro Enhancer EGA				
71h =	100x35	8x16	800x600	16of64		A000	NSI Smart EGA+
74h =			640x400	2		B800	Toshiba 3100 AT&T mode
7Eh = special mode set (see below)							Paradise VGA
7Fh = special function set							Paradise VGA

(see below)					
82h =	80x25		B&W		AT&T VDC overlay mode *
83h =	80x25				AT&T VDC overlay mode *
86h =		640x200	B&W		AT&T VDC overlay mode *
C0h =		640x400 2/prog pallet			AT&T VDC overlay mode *
C4h = disable output					AT&T VDC overlay mode *
D0h =		640x400	2	B800	DEC VAXmate AT&T mode
??? =		640x225			Z-100
??? =		640x400			Z-100

* for AT&T VDC overlay modes, BL contains the DEB mode, which may be 06h, 40h, or 44h
Note: IBM standard modes do not clear the screen if the high bit of AL is set

INT 10 - AX = 0070h VIDEO - Everex Micro Enhancer EGA - EXTENDED MODE SET

BL = mode (graphics mode if graphics resolution listed)

	text resol	graphic resol	monitor
00h =		640x480	multisync'ing
01h =		752x410	multisync'ing
02h = reserved			
03h =	80x34		multisync'ing
04h =	80x60		multisync'ing
05h =	94x29		multisync'ing
06h =	94x51		multisync'ing
07h = reserved			
08h = reserved			
09h =	80x44		EGA
0Ah =	132x25		EGA
0Bh =	132x44		EGA
0Ch =	132x25		CGA
0Dh =	80x44		mono
0Eh =	132x25		mono
0Fh =	132x44		mono

INT 10 - AX = 007Eh VIDEO - Paradise VGA - SET SPECIAL MODE

BX = The horizontal dimension of the mode desired
CX = The vertical dimension of the mode desired
(both BX/CX in pixels for graphics modes, rows for alpha modes)
DX = The number of colors of the mode desired (use 0 for monochrome modes)
Return: BH = 7Eh if successful.

INT 10 - AX = 007Fh VIDEO - Paradise VGA - EXTENDED FUNCTIONS

BH = 00h set VGA operation
BH = 01h set non-VGA operation
Color modes (0,1,2,3,4,5,6) will set non-VGA CGA operation.
Monochrome mode 7 will set non-VGA MDA/Hercules operation.
BH = 02h query mode status
Return: BL = 0 if operating in VGA mode, 1 if non-VGA mode.
CH = Total video RAM size in 64k byte units.
CL = Video RAM used by the current mode.
BH = 03h lock current mode
Allows current mode (VGA or non-VGA) to survive re-boot.
BH = 0Ah,0Bh,0Ch,0Dh,0Eh,0Fh WRITE PARADISE REGISTERS 0,1,2,3,4,5
(port 03CEh indices A,B,C,D,E,F)
BL = Value to set in the paradise register.
BH = 1Ah,1Bh,1Ch,1Dh,1Eh,1Fh READ PARADISE REGISTERS 0,1,2,3,4,5
(port 03CEh indices A,B,C,D,E,F)
Return: BL = Value of the paradise register.
BH = 7Fh if successful.

INT 10 - AH = 01h VIDEO - SET CURSOR CHARACTERISTICS

CH bits 0-4 = start line for cursor in character cell
bits 5-6 = blink attribute (00=normal, 01=invisible, 10=slow, 11=fast)
CL bits 0-4 = end line for cursor in character cell
Note: buggy on EGA systems--BIOS remaps cursor shape in 43 line modes, but returns unmapped cursor shape

INT 10 - AH = 02h VIDEO - SET CURSOR POSITION

DH,DL = row, column (0,0 = upper left)
BH = page number
0 in graphics modes
0-3 in modes 2&3
0-7 in modes 0&1

INT 10 - AH = 03h VIDEO - READ CURSOR POSITION

BH = page number
0 in graphics modes
0-3 in modes 2&3
0-7 in modes 0&1
Return: DH,DL = row,column
CH = cursor start line
CL = cursor end line

INT 10 - AH = 04h VIDEO - READ LIGHT PEN POSITION (all but PS)

Return: AH = 0: light pen switch not activated
AH = 1: light pen values in registers
DH,DL = row,column of current position
CH = raster line (0-199) (EGA) old graphics modes
CX = (EGA) raster line (0-nnn) new graphics modes
BX = pixel column (0-319 or 0-639)

INT 10 - AH = 05h VIDEO - SELECT DISPLAY PAGE

AL =
0-7: new page value for modes 0 & 1
0-3: new page value for modes 2 & 3
80h: read CRT/CPU page registers [PCjr only]
81h: set CPU page register to value in BL [PCjr only]
82h: set CRT page register to value in BH [PCjr only]
83h: set both display registers to values in BH, BL [PCjr only]
{Corona/Cordata BIOS v4.10+}
00h: set address of graphics bitmap buffer (video modes 60h,61h)
BX = segment of buffer
0Fh: get address of graphics bitmap buffer (video modes 60h,61h)
Return: BH = CRT page register (if AL >= 80h)
BL = CPU page register (if AL >= 80h)
DX = segment of graphics bitmap buffer (video modes 60h,61h; AL=0Fh)

INT 10 - AH = 06h VIDEO - SCROLL PAGE UP

AL = number of lines to scroll window (0 = blank whole window)
BH = attributes to be used on blanked lines
CH,CL = row,column of upper left corner of window to scroll
DH,DL = row,column of lower right corner of window

INT 10 - AH = 07h VIDEO - SCROLL PAGE DOWN

AL = number of lines to scroll window (0 = blank whole window)
BH = attributes to be used on blanked lines
CH,CL = row,column of upper left corner of window to scroll
DH,DL = row,column of lower right corner of window

INT 10 - AH = 08h VIDEO - READ ATTRIBUTES/CHARACTER AT CURSOR POSITION

BH = display page
Return: AL = character
AH = attribute of character (alpha modes)

INT 10 - AH = 09h VIDEO - WRITE ATTRIBUTES/CHARACTERS AT CURSOR POS

AL = character
BH = display page
BL = attributes of char (alpha modes) or color (graphics modes)
if bit 7 == 1 in graphics mode, character is xor'ed onto screen
CX = number of times to write character
Note: all characters are displayed, including CR, LF, and BS

INT 10 - AH = 0Ah VIDEO - WRITE CHARACTERS ONLY AT CURSOR POS

AL = character
BH = display page - alpha mode
BL = color of character (graphics mode, PCjr only)
if bit 7 == 1 in graphics mode, character is xor'ed onto screen
CX = number of times to write character
(EGA) in graphics modes, replication count in CX works correctly only if all character written are contains on the same row
Note: all characters are displayed, including CR, LF, and BS

INT 10 - AH = 0Bh VIDEO - SET COLOR PALETTE

BH = 0
BL = border color (0-15) (text modes) border color and background color (graphics modes) (EGA)
BL = border color (0-15) and high-intensity background color (16-31??? maybe should be high nybble?)
BH = 1
BL = palette (0-3)

INT 10 - AH = 0Ch VIDEO - WRITE DOT ON SCREEN

AL = color of dot (0/1 in mode 6, 0-3 in modes 4 and 5)
if bit 7 set, new color will be XORed with current pixel
BH = display page (ignored if mode only supports one page)
CX = column
DX = row
Note: only valid in graphics modes

INT 10 - AH = 0Dh VIDEO - READ DOT ON SCREEN

BH = display page (ignored if mode only supports one page)
CX = column
DX = row
Return: AL = color read
Note: only valid in graphics modes

INT 10 - AH = 0Eh VIDEO - WRITE CHARACTER AND ADVANCE CURSOR (TTY WRITE)

AL = character
BH = display page (alpha modes)

BL = foreground color (graphics modes)
Note: characters 07h (BEL), 08h (BS), 0Ah (LF), and 0Dh (CR) are interpreted and do the expected things

INT 10 - AH = 0Fh VIDEO - GET CURRENT VIDEO MODE

Return: AH = number of columns on screen
AL = current video mode (see INT 10h/AH=00h)
BH = current active display page
Note: if mode was set with bit 7 set ("no blanking"), the returned mode will also have bit 7 set

INT 10 - AH = 10h VIDEO - SET PALETTE REGISTERS (Jr, PS, TANDY 1000, EGA, VGA)

AL = subfunction
00h set palette register
BL = palette register to set
BH = color value to store
(on MCGA, only BX = 0712h is supported)
01h set border color register
BH = color value to store
02h set all palette registers and overscan
ES:DX -> 17-byte list
bytes 0-15 = values for palette regs. 0-15 byte 16 = value for border color register

INT 10 - AX = 1003h VIDEO - TOGGLE INTENSITY/BLINKING BIT (Jr, PS, TANDY 1000, EGA, VGA)

BL = 00h enable intensity
= 01h enable blink

INT 10 - AH = 10h VIDEO - GET PALETTE REGISTERS (VGA)

AL = subfunction
07h read individual palette register
BL = palette register number
Return: BH = palette register value
08h read overscan (border color) register
Return: BH = value
09h read all palette registers and overscan register
ES:DX = buffer address (17 bytes)

INT 10 - AH = 10h VIDEO - GET/SET DAC REGISTERS (EGA, VGA/MCGA)

AL = subfunction
10h set individual DAC register
BX = register number
CH = new value for green (0-63)
CL = new value for blue (0-63)
DH = new value for red (0-63)
12h set block of DAC registers
BX = starting color register
CX = number of registers to set
ES:DX = Table of 3*CX bytes where each 3 byte
group represents one byte each of red, green and blue (0-63)
13h select video DAC color page (VGA only)
BL = 00h Select paging mode
BH = 00h Select 4 blocks of 64
BH = 01h Select 16 blocks of 16
BL = 01h Select Page
BH = page number (00h to 03h) or (00h to 0Fh)
(not valid in mode 13h)
15h read individual DAC register
BL = palette register number
Return: DH = red value
CH = green value
CL = blue value
17h read block of DAC registers
BX = starting palette register
CX = number of palette registers to read
ES:DX = buffer (3 * CX bytes in size)
Return: CX number of red, green and blue triples in buffer
18h *UNDOCUMENTED* set PEL mask
BL = new PEL value
19h *UNDOCUMENTED* read PEL mask
BL = value read
1Ah read video DAC color-page state (VGA only)
Return: BL = paging mode
0 four pages of 64
1 sixteen pages of 16
BH = current page
1Bh perform gray-scale summing
BX = starting palette register
CX = number of registers to convert

INT 10 - AH = 11h VIDEO - TEXT-MODE CHARACTER GENERATOR FUNCTIONS (PS, EGA, VGA)

The following functions will cause a mode set, completely resetting
the video environment, but without clearing the video buffer
AL = 00h, 10h: load user-specified patterns
ES:BP -> user table
CX = count of patterns to store

DX = character offset into map 2 block
BL = block to load in map 2
BH = number of bytes per character pattern
AL = 01h, 11h: load ROM monochrome patterns (8 by 14)
BL = block to load
AL = 02h, 12h: load ROM 8 by 8 double-dot patterns
BL = block to load
AL = 03h: set block specifier
BL = block specifier
(EGA/MCGA) bits 0,1 = block selected by chars with attribute bit 3 = 0
bits 2,3 = block selected by chars with attribute bit 3 = 1
(VGA) bits 0,1,4 = block selected by attribute bit 3 = 0
bits 2,3,5 = block selected by attribute bit 3 = 1
AL = 04h, 14h: load ROM 8x16 character set (VGA)
The routines called with AL=1xh are designed to be called only immediately after a mode set and are similar to the routines called with AL=0xh, except that:
Page 0 must be active.
Bytes/character is recalculated.
Max character rows is recalculated.
CRT buffer length is recalculated.
CRTC registers are reprogrammed as follows:
R09 = bytes/char-1 ; max scan line (mode 7 only)
R0A = bytes/char-2 ; cursor start
R0B = 0 ; cursor end
R12 = ((rows+1)*(bytes/char))-1 ; vertical display end
R14 = bytes/char ; underline loc
(*** BUG: should be 1 less ***)

INT 10 - AH = 11h VIDEO - GRAPHICS-MODE CHARACTER GENERATOR FUNCTIONS (PS, EGA, VGA)

AL = 20h: set user 8 by 8 graphics characters (INT 1Fh)
ES:BP -> user table
AL = 21h: set user graphics characters
ES:BP -> user table
CX = bytes per character
BL = row specifier
0: user set
DL = number of rows
1: 14 rows
2: 25 rows
3: 43 rows
AL = 22h: ROM 8 by 14 set
BL = row specifier
AL = 23h: ROM 8 by 8 double dot
BL = row specifier
AL = 24h: load 8x16 graphics characters (VGA/MCGA)
BL = row specifier
Note: these functions are meant to be called only after a mode set

INT 10 - AX = 1103h VIDEO - GET FONT INFORMATION (EGA, MCGA, VGA)

BH = pointer specifier
0: INT 1Fh pointer
1: INT 44h pointer
2: ROM 8 by 14 character font pointer
3: ROM 8 by 8 double dot font pointer
4: ROM 8 by 8 DD font (top half)
5: ROM alpha alternate (9 by 14) pointer
Return: ES:BP = specified pointer
CX = bytes/character
DL = character rows on screen

INT 10 - AH = 12h VIDEO - ALTERNATE FUNCTION SELECT (PS, EGA, VGA, MCGA)

BL = 10h: return EGA information
Return: BH = 0: color mode in effect (3Dx)
1: mono mode in effect (3Bx)
BL = 0: 64k bytes memory installed
1: 128k bytes memory installed
2: 192k bytes memory installed
3: 256k bytes memory installed
CH = feature bits
CL = switch settings
BL = 20h: select alternate print screen routine
BL = 30h: select vertical resolution for alphanumeric modes (VGA only)
AL = 00h 200 scan lines
01h 350 scan lines
02h 400 scan lines
Return: AL = 12h if function supported
BL = 31h: enable/disable default palette loading (VGA/MCGA)
AL = 00h enable default palette loading
01h disable default palette loading
Return: AL = 12h if function was supported

BL = 32h: enable/disable video addressing (VGA/MCGA)
AL = 00h enable video
01h disable video
Return: AL = 12h if function was supported
BL = 33h: enable/disable default gray-scale summing (VGA/MCGA)
AL = 00h enable gray scale summing
01h disable gray scale summing
Return: AL = 12h if function was supported
BL = 34h: enable/disable alphanumeric cursor emulation (VGA only)
AL = 00h enable cursor emulation
01h disable cursor emulation
Return: AL = 12h if function was supported
BL = 35h: PS/2 display-switch interface
AL = 00h initial adapter video off
01h initial planar video on
02h switch active video off
03h switch inactive video on
80h *UNDOCUMENTED* set system board video active flag
ES:DX = buffer (128 byte save area if AL = 0, 2 or 3)
Return: AL = 12h if function was supported
BL = 36h: video refresh control (VGA/PS)
AL = 00h enable refresh
01h disable refresh
Return: AL = 12h if function supported
BX = 5500h ??? (used by ATI and TAXAN)
BX = 5502h ??? (used by ATI and TAXAN)

INT 10 - AH = 13h VIDEO - WRITE STRING (AT,XT286,PS,EGA,VGA)

AL = mode
bit 0: set in order to move cursor after write
bit 1: set if string contains alternating characters and attributes
BL = attribute if AL bit 1 clear
BH = display page number
DH,DL = row,column of starting cursor position
CX = length of string
ES:BP -> start of string
Note: recognizes CR, LF, BS, and bell

INT 10 - AH = 14h VIDEO - LOAD LCD CHARACTER FONT (CONVERTIBLE)

AL = 0 load user-specified font
ES:DI -> character font
BH = number of bytes per character
BL = 0: load main font (block 0)
1: load alternate font (block 1)
CX = number of characters to store
DX = character offset into RAM font area
AL = 1 load system rom default font
BL = 0: load main font (block 0)
1: load alternate font (block 1)
AL = 2 set mapping of LCD high intensity attributes
BL = 0: ignore high intensity attribute
1: map high intensity to underscore
2: map high intensity to reverse video
3: map high intensity to selected alternate font

INT 10 - AH = 15h VIDEO - GET PHYSICAL DISPLAY PARAMETERS (CONVERTIBLE)

Return: AX = alternate display adapter type
0000h none
5140h LCD
5153h CGA
5151h mono
ES:DI -> parameter table
word 0: monitor model number
1: vertical pixels per meter
2: horizontal pixels per meter
3: total vertical pixels
4: total horizontal pixels
5: horizontal pixel separation in micrometers
6: vertical pixel separation in micrometers

INT 10 - AH = 1Ah VIDEO - DISPLAY COMBINATION (PS,VGA/MCGA)

AL = 00h read display combination code
Return: BL = active display code
BH = alternate display code
01h set display combination code
BL = active display code
BH = alternate display code
Return: AL = 1Ah if function was supported

Display combination codes:
00h no display
01h monochrome adapter w/ monochrome display
02h CGA w/ color display

03h reserved
04h EGA w/ color display
05h EGA w/ monochrome display
06h PGA w/ color display
07h VGA w/ monochrome analog display
08h VGA w/ color analog display
09h reserved
0Ah MCGA w/ digital color display
0Bh MCGA w/ monochrome analog display
0Ch MCGA w/ color analog display
FFh unknown display type

INT 10 - AH = 1Bh VIDEO - FUNCTIONALITY/STATE INFORMATION (PS,VGA/MCGA)

BX = implementation type
= 0000h return funtionality/state information
ES:DI -> 64 byte buffer
Return: AL = 1Bh if function supported
ES:DI buffer filled
00h address of static funtionality table
04h video mode in effect
05h number of columns
07h length of regen buffer in bytes
09h starting address of regen buffer
0Bh cursor position for page 0
0Dh cursor position for page 1
0Fh cursor position for page 2
11h cursor position for page 3
13h cursor position for page 4
15h cursor position for page 5
17h cursor position for page 6
19h cursor position for page 7
1Bh cursor type
1Dh active display page
1Eh CRTC port address
20h current setting of register (3?8)
21h current setting of register (3?9)
22h number of rows
23h bytes/character
25h DCC of active display
26h DCC of alternate display
27h number of colors supported in current mode
29h number of pages supported in current mode
2Ah number of scan lines active
(0,1,2,3) = (200,350,400,480)
2Bh primary character block
2Ch secondary character block
2Dh Miscellaneous flags
bit 0 all modes on all displays on
1 gray summing on
2 monochrome display attached
3 default palette loading disabled
4 cursor emulation enabled
5 0 = intensity; 1 = blinking
6 reserved
7 reserved
2Eh to 30h reserved
31h video memory available
00h = 64K, 01h = 128K, 02h = 192K, 03h = 256K
32h save pointer state flags
bit 0 512 character set active
1 dynamic save area present
2 alpha font override active
3 graphics font override active
4 palette override active
5 DCC override active
6 reserved
7 reserved
33h to 3Fh reserved

State Functionality Table format (16 bytes)
00h modes supported #1
bit 0 to bit 7 = 1 modes 0,1,2,3,4,5,6 supported
01h modes supported #2
bit 0 to bit 7 = 1 modes 8,9,A,B,C,D,E,F supported
02h modes supported #3
bit 0 to bit 3 = 1 modes 10,11,12,13 supported
bit 4 to bit 7 reserved
03h to 06h reserved
07h scan lines supported
bit 0 to bit 2 = 1 if scan lines 200,350,400 supported
08h total number of character blocks available in text modes
09h maximum number of active character blocks in text modes

0Ah miscellaneous function flags #1
bit 0 all modes on all displays function supported
1 gray summing function supported
2 character font loading function supported
3 default palette loading enable/disable supported
4 cursor emulation function supported
5 EGA palette present
6 color palette present
7 color paging function supported
0Bh miscellaneous function flags #2
bit 0 light pen supported
1 save/restore state function 1Ch supported
2 intensity blinking function supported
3 Display Combination Code supported
4-7 reserved
0Ch to 0Dh reserved
0Eh Save pointer function flags
bit 0 512 character set supported
1 dynamic save area supported
2 alpha font override supported
3 graphics font override supported
4 palette override supported
5 DCC extension supported
6 reserved
7 reserved
0Fh reserved

INT 10 - AH = 1Ch VIDEO - SAVE/RESTORE VIDEO STATE (PS50+,VGA)

CX = requested states
bit 0 video hardware
1 BIOS data areas
2 color registers and DAC state
3-15 reserved
AL = 0: return state buffer size
Return: BX = number of 64 byte blocks needed
1: save video state
ES:BX = buffer address
2: restore video state
ES:BX = buffer address of previously saved state
Return: AL = 1Ch if function supported

INT 10 - AH = 40h VIDEO - SET GRAPHICS MODE (Hercules GRAFIX)

INT 10 - AH = 41h VIDEO - SET TEXT MODE (Hercules GRAFIX)

INT 10 - AH = 42h VIDEO - CLEAR CURRENT PAGE (Hercules GRAFIX)

INT 10 - AH = 43h VIDEO - SELECT DRAWING PAGE (Hercules GRAFIX)

AL = page number (0,1)

INT 10 - AH = 44h VIDEO - SELECT DRAWING FUNCTION (Hercules GRAFIX)

AL = drawing function
00h clear pixels
01h set pixels
02h invert pixels

INT 10 - AH = 45h VIDEO - SELECT PAGE TO DISPLAY (Hercules GRAFIX)

AL = page number (0,1)

INT 10 - AH = 46h VIDEO - DRAW ONE PIXEL (Hercules GRAFIX)

DI = x (0-720)
BP = y (0-347)
Note: function 44h determines operation and function 43h which page to use

INT 10 - AH = 47h VIDEO - FIND PIXEL VALUE (Hercules GRAFIX)

DI = x (0-720)
BP = y (0-347)
Return: AL = 0 pixel clear
AL = 1 pixel set
Note: function 43h specifies which page is used

INT 10 - AH = 48h VIDEO - MOVE TO POINT (Hercules GRAFIX)

DI = x (0-720)
BP = y (0-347)

INT 10 - AH = 49h VIDEO - DRAW TO POINT (Hercules GRAFIX)

DI = x (0-720)
BP = y (0-347)
Note: function 48h or 49h specify first point, 44h operation and 43h page to use

INT 10 - AH = 4Ah VIDEO - BLOCK FILL (Hercules GRAFIX)

INT 10 - AH = 4Bh VIDEO - DISPLAY CHARACTER (Hercules GRAFIX)

AL = character to display
DI = x (0-720)
BP = y (0-347)
Note: Unlike the other BIOS character functions character position is specified in pixels rather than rows and columns.

INT 10 - AH = 4Ch VIDEO - DRAW ARC (Hercules GRAFIX)

INT 10 - AH = 4Dh VIDEO - DRAW CIRCLE (Hercules GRAFIX)

INT 10 - AH = 4Eh VIDEO - FILL AREA (Hercules GRAFIX)

INT 10 - AX = 6A00h Direct Graphics Interface Standard (DGIS) - INQUIRE AVAILABLE DEVICES

BX = 0
CX = 0
DX = buffer length (may be 0)
ES:DI = address of buffer
Return: BX = number of bytes stored in buffer
CX = bytes required for all descriptions (0 if no DGIS)
Note: buffer contains descriptions and addresses of DGIS-compatible display(s)
and printer(s)

INT 10 - AX = 6A01h DGIS - REDIRECT CHARACTER OUTPUT

CX = 0
ES:DI = address of device to send INT 10 output to
Return: CX = 0 output could not be redirected else INT 10h output now routed to requested display

INT 10 - AX = 6A02h DGIS - INQUIRE INT 10 OUTPUT DEVICE

ES:DI = 0:0
Return: ES:DI = 0:0 if current display is non-DGIS else address of the current DGIS INT 10 display

INT 10 - AX = 6F05h VIDEO - SET VIDEO MODE (VEGA EXTENDED EGA/VGA)

BL = mode (graphics mode if graphics resolution listed)

	graphic resltn	color	system
62h =	800x600	16	VEGA Extended EGA
65h =	1024x768	16	VEGA Extended EGA
66h =	640x400	256	VEGA Extended VGA
67h =	640x480	256	VEGA Extended VGA
68h =	720x540	256	VEGA Extended VGA
69h =	800x600	256	VEGA Extended VGA

INT 10 - AH = 70h VIDEO - GET VIDEO RAM ADDRESS (TANDY 1000)

Return: AX = segment address of the following
[BX] = offset address of green plane
[CX] = segment address of green plane
[DX] = segment address of red/blue plane
(red offset = 0, blue offset = 4000)

INT 10 - AH = 71h VIDEO - GET INCRAM ADDRESSES (TANDY 1000)

Return: AX = segment address of the following
[BX] = segment address of INCRAM
[CX] = offset address of INCRAM

INT 10 - AH = 72h VIDEO - SCROLL SCREEN RIGHT (TANDY 1000)

AL = number of columns blanked at left of window
0 = blank entire window
BH = attributes to be used on blank columns
CH,CL = row, column of upper left corner of window
DH,DL = row, column of lower right corner

INT 10 - AH = 73h VIDEO - SCROLL SCREEN LEFT (TANDY 1000)

AL = number of columns blanked at right of window
0 = blank entire window
BH = attributes to be used on blank columns
CH,CL = row, column of upper left corner of window
DH,DL = row, column of lower right corner

INT 10 - AH = 80h VIDEO (DESQview) - SET ??? HANDLER

DX = 4456h ('DV')
ES:DI -> FAR subroutine to be called on ???
Return: DS = segment of DESQview data structure for video buffer
Note: this function is probably meant for internal use only, due to the magic
value required in DX
the subroutine seems to be called when the DESQview menu is accessed;
on entry, AL = 3 or 4

INT 10 - AH = 81h VIDEO (DESQview) - GET ???

DX = 4456h ('DV')
Return: ES = segment of DESQview data structure for video buffer BYTE ES:[0] = current window number in DV 2.0x
Note: this function is probably meant for internal use only, due to the magic value required in DX

INT 10 - AH = 82h VIDEO (DESQview) - GET CURRENT WINDOW INFO

DX = 4456h ('DV')
Return: DS = segment in DESQview for data structure in DV 2.00,
BYTE DS:[0] = window number
WORD DS:[1] = segment of other data structure
WORD DS:[3] = segment of window's object handle
ES = segment of DESQview data structure for video buffer
AL = current window number
AH = ???
BL = direct screen writes
0 program does not do direct writes
1 program does direct writes, so shadow buffer not usable
BH = ???
CL = current video mode
CH = ???
Note: this function is probably meant for internal use only, due to the magic value required in DX

INT 10 - AH = BFh VIDEO - Compaq Portable Extensions

AL = subfunction
00h select external monitor
all registers preserved and the internal monitor is blanked
and the external monitor is now the active monitor
01h select internal monitor
all registers preserved and the external monitor is blanked
and the internal monitor is now the active monitor
02h set master mode of the currently active video controller
BH = 04h CGA
BH = 05h EGA
BH = 07h MDA
03h get environment
BX = 0000
Return: BH = active monitor
00h = External
01h = Internal
BL = master mode
00h = switchable VDU not present
04h = CGA
05h = EGA
07h = MDA
CH = 00h (reserved)
CL = switchable VDU mode supported
bit 0 = CGA supported
bits 1,2 = reserved (1)
bit 3 = MDA supported
bits 4-7 = reserved (1)
DH = internal monitor type
00h = none
01h = Dual-mode monitor
02h = 5153 RGB monitor
03h = Compaq Color monitor
04h = 640x400 flat panel
DL = external monitor type
00h = none
01h = Dual-mode monitor
02h = 5153 RGB monitor
03h = Compaq Color monitor
04h = 640x400 flat panel
04h set mode switch delay
BH = switch
00h = enable delay
01h = disable delay

INT 10 - AH = F0h Microsoft Mouse driver EGA support - READ ONE REGISTER

BL = register number
DX = group index
Pointer/data chips
00h CRT Controller (25 reg) 3B4h mono modes, 3D4h color modes
08h Sequencer (5 registers) 3C4h
10h Graphics Controller (9 registers) 3CEh
18h Attribute Controller (20 registers) 3C0h
Single registers
20h Miscellaneous Output register 3C2h
28h Feature Control register (3BAh mono modes, 3DAh color modes)
30h Graphics 1 Position register 3CCh
38h Graphics 2 Position register 3CAh
Return: BL = data

INT 10 - AH = F1h Microsoft Mouse driver EGA support - WRITE ONE REGISTER

DX = group index (see function F0h)
BL = register number
BH = value to write
Return: BL = data

INT 10 - AH = F2h Microsoft Mouse driver EGA support - READ REGISTER RANGE

CH = starting register number
CL = Number of registers (>1)
DX = group index
00h CRTC (3B4h mono modes, 3D4h color modes)
08h Sequencer 3C4h
10h Graphics Controller 3CEh
18h Attribute Controller 3C0h
ES:BX -> buffer, CL bytes

INT 10 - AH = F3h Microsoft Mouse driver EGA support - WRITE REGISTER RANGE

CH = starting register
CL = number of registers (>1)
DX = group index
00h CRTC (3B4h mono modes, 3D4h color modes)
08h Sequencer 3C4h
10h Graphics Controller 3CEh
18h Attribute Controller 3C0h
ES:BX -> buffer, CL bytes

INT 10 - AH = F4h Microsoft Mouse driver EGA support - READ REGISTER SET

CX = number of registers (>1)
ES:BX -> table of records in this format:
bytes 1-2 group index
Pointer/data chips
00h CRTC (3B4h mono modes, 3D4h color modes)
08h Sequencer 3C4h
10h Graphics Controller 3CEh
18h Attribute Controller 3C0h
Single registers
20h Miscellaneous Output register 3C2h
28h Feature Control register (3BAh mono modes, 3DAh color)
30h Graphics 1 Position register 3CCh
38h Graphics 2 Position register 3CAh
byte 3 register number (0 for single registers)
byte 4 register value

INT 10 - AH = F5h Microsoft Mouse driver EGA support - READ REGISTER SET

CX = number of registers (>1)
ES:BX -> table of records in this format:
bytes 1-2 port number
Pointer/data chips
00h CRTC (3B4h mono modes, 3D4h color modes)
08h Sequencer 3C4h
10h Graphics Controller 3CEh
18h Attribute Controller 3C0h
Single registers
20h Miscellaneous Output register 3C2h
28h Feature Control register (3BAh mono modes, 3DAh color)
30h Graphics 1 Position register 3CCh
38h Graphics 2 Position register 3CAh
byte 3 register number (0 for single registers)
byte 4 register value

INT 10 - AH = F6h Microsoft Mouse driver EGA support - REVERT TO DEFAULT REGISTERS

INT 10 - AH = F7h Microsoft Mouse driver EGA support - DEFINE DEFAULT REGISTER TABLE

DX = port number
Pointer/data chips
00h CRTC (3B4h mono modes, 3D4h color modes)
08h Sequencer 3C4h
10h Graphics Controller 3CEh
18h Attribute Controller 3C0h
Single registers
20h Miscellaneous Output register 3C2h
28h Feature Control register (3BAh mono modes, 3DAh color modes)
30h Graphics 1 Position register 3CCh
38h Graphics 2 Position register 3CAh
ES:BX address of table of one byte entries, one byte
to be written to each register

INT 10 - AH = FAh Microsoft Mouse driver EGA support - INTERROGATE DRIVER

BX = 0
Return: BX = 0 if mouse driver not present
ES:BX -> EGA Register Interface version number, if present:
byte 1 = major release number
byte 2 = minor release number

INT 10 - AH = FEh VIDEO (TopView) - GET VIDEO BUFFER

ES:DI = segment:offset of assumed video buffer
Return: ES:DI = segment:offset of actual video buffer

INT 10 - AH = FFh VIDEO (TopView) - UPDATE REAL SCREEN FROM VIDEO BUFFER

CX = number of sequential characters that have been modified
DI = offset of first character that has been modified
ES = segment of video buffer
Note: avoid CX=0

INT 11 - EQUIPMENT DETERMINATION

Return: AX = equipment flag bits
0 diskette installed
1 8087 present
2 mouse installed (PS2 only)
2,3 number of 16K banks of RAM on motherboard (PC only)
number of 64K banks of RAM on motherboard (XT only)
always = 11 on AT and above
4,5 initial video mode
01 = 40x25 color
10 = 80x25 color
11 = 80X25 IBM monochrome
6,7 number of diskette drives (only if bit 0 = 1)
00 = 1, 01 = 2, 10 = 3, 11 = 4
8 0 = dma present, 1 = no dma on system (PCjr???)
9-11 number of RS232 cards
12 game I/O attached
13 serial printer installed (PCjr)
internal modem installed (PC/Convertible)
14,15 number of printers

INT 12 - MEMORY SIZE

Return: AX = number of contiguous 1K blocks of memory

INT 13 - AH = 00h DISK - RESET DISK SYSTEM

DL = drive (if bit 7 is set both hard disks and floppy disks reset)

INT 13 - AH = 01h DISK - STATUS OF DISK SYSTEM

DL = drive (hard disk if bit 7 set)
Return: AL = status
00h = successful completion
01h = bad command
02h = address mark not found
03h = write attempted on write-protected disk
04h = sector not found
05h = reset failed (hard disk)
06h = diskette changed
07h = parameter act. failed (hard disk)
08h = DMA overrun (floppy disk)
09h = DMA across 64K boundary
0Ah = bad sector detected (hard disk)
0Bh = bad track detected (hard disk)
0Ch = unsupported track
0Dh = invalid number of sectors on format (hard disk)
0Eh = control data address mark detected (hard disk)
0Fh = DMA arbitration error (hard disk)
10h = bad CRC/ECC
11h = data ECC corrected (hard disk)
20h = controller failure
40h = seek failed
80h = time out
AAh = drive not ready (hard disk)
BBh = undefined error (hard disk)
CCh = write fault (hard disk)
E0h = status register error (hard disk)
FFh = sense operation failed (hard disk)

INT 13 - AH = 02h DISK - READ SECTORS INTO MEMORY

AL = number of sectors to read
CH = track (for hard disk, bits 8,9 in high bits of CL)
CL = sector
DH = head
DL = drive
ES:BX = address of buffer to fill
Return: CF set on error
AH = status (see AH=1 above)
AL = number of sectors read

INT 13 - AH = 03h DISK - WRITE SECTORS FROM MEMORY

AL = number of sectors to write
CH = track (if hard disk, bits 8,9 in high bits of CL)
CL = sector (if hard disk, high two bits are high bits of track #)
DH = head
DL = drive
ES:BX = address of buffer
Return: CF set on error

AH = status (see AH=1 above)
AL = number of sectors written

INT 13 - AH = 04h DISK - VERIFY SECTORS

AL = number of sectors to verify
CH = track (for hard disk, bits 8,9 in high bits of CL)
CL = sector
DH = head
DL = drive
Return: CF set on error
AH = status (see AH=1 above)
AL = number of sectors verified

INT 13 - AH = 05h FLOPPY - FORMAT TRACK

AL = number of sectors to create on this track
CH = track
CL = sector
DH = head
DL = drive
ES:BX -> array of 4-byte address fields
byte 1 = track
byte 2 = head
byte 3 = sector
byte 4 = bytes/sector 0=128, 1=256, 2=512, 3=1024
Return: CF set if error occurred
AH = status code (see AH=1 above)

INT 13 - AH = 05h FIXED DISK - FORMAT TRACK

AL = interleave value (XT only)
ES:BX = 512-byte format buffer
the first 2*(sectors/track) bytes contain F,N for each sector
F = 00 for good sector, 80h for bad sector
N = sector number
CH = cylinder number (bits 8,9 in high bits of CL)
CL = sector number
DH = head
DL = drive
Return: AH = status code (see AH=1 above)

INT 13 - AH = 06h FIXED DISK - FORMAT TRACK AND SET BAD SECTOR FLAGS (XT,PORT)

AL = interleave value
CH = cylinder number (bits 8,9 in high bits of CL)
CL = sector number
DH = head
DL = drive
Return: AH = status code (see AH=1 above)

INT 13 - AH = 07h FIXED DISK - FORMAT DRIVE STARTING AT GIVEN TRACK (XT,PORT)

AL = interleave value (XT only)
ES:BX = 512-byte format buffer, see AH=6 above
CH = cylinder number (bits 8,9 in high bits of CL)
CL = sector number
DH = head
DL = drive
Return: AH = status code (see AH=1 above)

INT 13 - AH = 08h DISK - GET CURRENT DRIVE PARAMETERS (XT,AT,XT286,CONV,PS)

DL = drive number
Return: CF set on error
AH = status code (see AH=1 above)
BL = drive type (see AH=17h below) (AT/PS2 floppies only)
DL = number of consecutive acknowledging drives
DH = maximum value for head number
CL = maximum value fo sector number
CH = maximum value for cylinder number
ES:DI = drive parameter table

INT 13 - AH = 09h FIXED DISK - INITIALIZE TWO FIXED DISK BASE TABLES (XT,AT,XT286,PS)

Return: CF set on error
AH = status code (see AH=1 above)
INT 41h points to table for drive 0
INT 46h points to table for drive 1

INT 13 - AH = 0Ah FIXED DISK - READ LONG (XT,AT,XT286,PS)

DL = drive ID
DH = head
CH = cylinder (bits 8,9 in high bits of CL)
CL = sector
ES:BX -> buffer to fill
Return: CF set on error
AH = status code (see AH=1 above)
AL = number of sectors read
Note: used for diagnostics only on PS/2 systems

INT 13 - AH = 0Bh FIXED DISK - WRITE LONG (XT,AT,XT286,PS)

DL = drive ID
DH = head
CH = cylinder (bits 8,9 in high bits of CL)
CL = sector
ES:BX -> buffer containing data
Return: CF set on error
AH = status code (see AH=1 above)
AL = number of sectors written
Note: used for diagnostics only on PS/2 systems

INT 13 - AH = 0Ch FIXED DISK - SEEK TO CYLINDER (XT,AT,XT286,PS)

DL = drive ID
DH = head
CH = cylinder (bits 8,9 in high bits of CL)
Return: CF set on error
AH = status code (see AH=1 above)

INT 13 - AH = 0Dh FIXED DISK - ALTERNATE DISK RESET (XT,AT,XT286,PS)

DL = drive ID
Return: CF set on error
AH = status code (see AH=1 above)

INT 13 - AH = 0Eh FIXED DISK - READ SECTOR BUFFER (XT,PS)

ES:BX -> buffer
Return: CF set on error
AH = status code (see AH=1 above)
Notes: transfers controller's sector buffer. No data is read from the drive
used for diagnostics only on PS/2 systems

INT 13 - AH = 0Fh FIXED DISK - WRITE SECTOR BUFFER (XT,PS)

ES:BX -> buffer
Return: CF set on error
AH = status code (see AH=1 above)
Notes: should be called before formatting to initialize the controller's
sector buffer.
used for diagnostics only on PS/2 systems

INT 13 - AH = 10h FIXED DISK - TEST FOR DRIVE READY (XT,AT,XT286,PS)

DL = drive ID
Return: CF set on error
AH = status code (see AH=1 above)

INT 13 - AH = 11h FIXED DISK - RECALIBRATE DRIVE (XT,AT,XT286,PS)

DL = drive ID
Return: CF set on error
AH = status code (see AH=1 above)

INT 13 - AH = 12h FIXED DISK - CONTROLER RAM DIAGNOSTIC (XT,PS)

Return: CF set on error
AH = status code (see AH=1 above)
Note: used for diagnostics only on PS/2 systems

INT 13 - AH = 13h FIXED DISK - DRIVE DIAGNOSTIC (XT,PS)

Return: CF set on error
AH = status code (see AH=1 above)
Note: used for diagnostics only on PS/2 systems

INT 13 - AH = 14h FIXED DISK - CONTROLLER DIAGNOSTICS (XT,AT,XT286,PS)

Return: CF set on error
AH = status code (see AH=1 above)
Note: used for diagnostics only on PS/2 systems

INT 13 - AH = 15h DISK - GET TYPE (AT,XT2,XT286,CONV,PS)

DL = drive ID
Return: CF set on error
AH = disk type
0 = disk not there
1 = floppy, no change detection present
2 = floppy with change detection
3 = fixed disk
CX:DX = number of 512-byte sectors

INT 13 - AH = 16h FLOPPY DISK - CHANGE OF DISK STATUS (AT,XT2,XT286,CONV,PS)

DL = drive to check
Return: AH = disk change status
0 = no disk change
6 = disk changed

INT 13 - AH = 17h DISK - SET TYPE (AT,XT2,XT286,CONV,PS)

AL = disk type
00h = no disk
01h = regular disk in regular drive

02h = regular disk in high-capacity drive
03h = high-capacity disk in high-capacity drive
04h = 720K disk in 720K drive
DL = drive ID

INT 13 - AH = 18h DISK - SET MEDIA TYPE FOR FORMAT (AT model 3x9,XT2,XT286,PS)

DL = drive number
CH = lower 8 bits of number of tracks
CL = sectors per track (bits 0-5)
top 2 bits of number of tracks (bits 6,7)
Return: AH = 00h requested combination supported
01h function not available
0Ch not supported or drive type unknown
80h there is no disk in the drive
ES:DI -> 11-byte parameter table

INT 13 - AH = 19h FIXED DISK - PARK HEADS (XT286,PS)

DL = drive
Return: CF set on error
AH = status (see AH=1 above)

INT 13 - AH = 1Ah ESDI FIXED DISK - FORMAT UNIT (PS)

AL = defect table count
CL = format modifiers
bit 0: ignore primary defect map
bit 1: ignore secondary defect map
bit 2: update secondary defect map
bit 3: perform surface analysis
bit 4: generate periodic interrupt
DL = drive
ES:BX -> defect table
Return: CF set on error
AH = status (see AH=1 above)
Note: if periodic interrupt selected, INT 15h/AH=0Fh is called after each cylinder is formatted

INT 14 - AH = 00h SERIAL I/O - INITIALIZE USART

AL = initializing parameters
7 - 6 - 5 4 - 3 2 1 - 0
-BAUD RATE- PARITY STOP WORD
BITS LENGTH
000 110 bd 00 none 0-1 10 - 7
001 150 bd 01 odd 1-2 11 - 8
010 300 bd 11 even
011 600 bd
100 1200 bd
101 2400 bd
110 4800 bd
111 9600 bd (4800 on PCjr)
DX = port number (0-3)
Return: AH = RS-232 status code bits
0: data ready
1: overrun error
2: parity error
3: framing error
4: break detected
5: transmission buffer register empty
6: transmission shift register empty
7: time out--if set, other bits invalid
AL = modem status bits
0: delta Clear-To-Send
1: delta Data-Set-Ready
2: trailing edge of ring detected
3: change in receive line signal detected
4: Clear-To-Send
5: Data-Set-Ready
6: ring detected
7: receive line signal detected

INT 14 - AH = 00h FOSSIL (Fido/Opus/Seadog Standard Interface Level) - INITIALIZE

AL = initializing parameters
7 - 6 - 5 4 - 3 2 1 - 0
-BAUD RATE- PARITY STOP WORD
BITS LENGTH
000 19200 bd 00 none 0-1 10 - 7
001 38400 bd 01 odd 1-2 11 - 8
010 300 bd 11 even
011 600 bd
100 1200 bd
101 2400 bd
110 4800 bd
111 9600 bd (4800 on PCjr)

DX = port number (0-3)
Return: AH = RS-232 status code bits
0: RDA - input data is available in buffer
1: OVRN - data has been lost
5: THRE - room is available in output buffer
6: TSRE - output buffer empty
AL = modem status bits
3: always 1
7: DCD - carrier detect

INT 14 - AH = 01h SERIAL I/O - TRANSMIT CHARACTER

AL = character
DX = port number (0-3)
Return: AX = port status (see AH = 00h above)

INT 14 - AH = 02h SERIAL I/O - RECEIVE CHARACTER

DX = port number (0-3)
Return: AL = character received
AH = RS-232 status code (see AH = 00h above)

INT 14 - AH = 02h FOSSIL - RECEIVE CHARACTER WITH WAIT

DX = port number (0-3)
Return: AL = character received
AH = 00h

INT 14 - AH = 03h SERIAL I/O - GET USART STATUS

DX = port number (0-3)
Return: AX = port status code (see AH = 00h above)

INT 14 - AH = 04h SERIAL I/O - EXTENDED INITIALIZE (CONVERTIBLE,PS)

AL = break status
0 if break 1 if no break
BH = parity
0 no parity 1 odd parity 2 even parity 3 stick parity odd 4 stick parity even
BL = number of stop bits
0: one stop bit 1: two stop bits (1.5 if 5 bit word length)
CH = word length
0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits
CL = baud rate
0: 110 1: 150 2: 300 3: 600 4: 1200 5: 2400 6: 4800 7: 9600 8: 19200
DX = port number
Return: AX = port status code (see AH = 00h above)

INT 14 - AH = 04h FOSSIL - INITIALIZE DRIVER

DX = port number
optionally BX=4F50h
ES:CX = address of byte to be set upon ^C
Return: AX = 1954h (if successful)
BL = maximum function number supported (excluding 7Eh and above)
BH = revision of FOSSIL supported
DTR is raised

INT 14 - AH = 05h SERIAL I/O - EXTENDED COMMUNICATION PORT CONTROL (CONVERTIBLE,PS)

AL = 0 read modem control register
Return: BL = modem control register (see below)
AH = status
AL = 1 write modem control register
BL = modem control register
bit 0: data terminal ready bit 1: request to send
bit 2: OUT1 bit 3: OUT2
bit 4: LOOP bits 5-7 reserved
Return: AX = status
DX = port number

INT 14 - AH = 05h FOSSIL - DEINITIALIZE DRIVER

DX = port number
Return: none
DTR is not affected

INT 14 - AH = 06h FOSSIL - RAISE/LOWER DTR

DX = port
AL = DTR state to be set
00h = lower 01h = raise

INT 14 - AH = 07h FOSSIL - RETURN TIMER TICK PARAMETERS

Return: AL = timer tick interrupt number
AH = ticks per second on interrupt number in AL
DX = approximate number of milliseconds per tick

INT 14 - AH = 08h FOSSIL - FLUSH OUTPUT BUFFER WAITING TILL ALL OUTPUT IS DONE

DX = port number

INT 14 - AH = 09h FOSSIL - PURGE OUTPUT BUFFER THROWING AWAY ALL PENDING OUTPUT

DX = port number

INT 14 - AH = 0Ah FOSSIL - PURGE INTPUT BUFFER THROWING AWAY ALL PENDING INPUT

DX = port number

INT 14 - AH = 0Bh FOSSIL - TRANSMIT NO WAIT

AL = character
DX = port number
Return: AX = 0000h character not accepted
= 0001h character accepted

INT 14 - AH = -Ch FOSSIL - NON-DESTRUCTIVE READ AHEAD

DX = port number
Return: AX = FFFFh character not available
AX = 00xxh character xx available

INT 14 - AH = 0Dh FOSSIL - KEYBOARD READ WITHOUT WAIT

Return: AX = FFFFh character not available
= xxyyh standard IBM-style scan code

INT 14 - AH = 0Eh FOSSIL - KEYBOARD READ WITH WAIT

Return: AX = xxyyh standard IBM-style scan code

INT 14 - AH = 0Fh FOSSIL - ENABLE/DISABLE FLOW CONTROL

AL = bit mask describing flow control requested
0: xon/xoff on transmit (watch for xoff while sending)
1: CTS/RTS (CTS on transmit/RTS on receive)
2: reserved
3: xon/xoff on receive (send xoff when buffer near full)
4-7: all 1
DX = port number

INT 14 - AH = 10h FOSSIL - EXTENDED ^C/^K CHECKING AND TRANSMIT ON/OFF

AL = bit mask
0: enable/disable ^C/^K checking
1: enable/disable the transmitter
DX = port number

INT 14 - AH = 11h FOSSIL - SET CURRENT CURSOR LOCATION

DH = row
DL = column
Note: this is the same as INT 10/AH=02h

INT 14 - AH = 12h FOSSIL - READ CURRENT CURSOR LOCATION

Return: DH = row
DL = column
Note: this is the same as INT 10/AH=03h

INT 14 - AH = 13h FOSSIL - SINGLE CHARACTER ANSI WRITE TO SCREEN

AL = character

INT 14 - AH = 14h FOSSIL - ENABLE OR DISABLE WATCHDOG PROCESSING

AL = 01h enable watchdog
00h disable watchdog
DX = port number

INT 14 - AH = 15h FOSSIL - WRITE CHARACTER TO SCREEN USING BIOS SUPPORT ROUTINES

AL = character

INT 14 - AH = 16h FOSSIL - INSERT/DELETE FUNCTION FROM TIMER TICK CHAIN

AL = function
00h = delete
01h = add
ES:DX -> routine to call
Return: AX = 0000h successful
0001h unsuccessful

INT 14 - AH = 17h FOSSIL - REBOOT SYSTEM

AL = method
00h = cold boot
01h = warm boot

INT 14 - AH = 18h FOSSIL - READ BLOCK

CX = maximum number of characters to transfer
DX = port number
ES:DI -> user buffer
Return: AX = number of characters transfered

INT 14 - AH = 19h FOSSIL - WRITE BLOCK

CX = maximum number of characters to transfer
DX = port number
ES:DI -> user buffer

Return: AX = number of characters transfered

INT 14 - AH = 1Ah FOSSIL - BREAK BEGIN OR END

AL = 00h stop sending 'break'
01h start sending 'break'
DX = port number

INT 14 - AH = 1Bh FOSSIL - RETURN INFORMATION ABOUT THE DRIVER

DX = port number
CX = size of user buffer
ES:DI -> user buffer
Return: AX = number of characters transferred
Structure =

WORD	size of structure in bytes
BYTE	FOSSIL spec driver conforms to
BYTE	revision level of this specific driver
DWORD	pointer to ASCII id string
WORD	size of the input buffer
WORD	number of bytes left in buffer
WORD	size of the output buffer
WORD	number of bytes left in buffer
BYTE	width of screen
BYTE	length of screen
BYTE	actual baud rate, computer to modem

INT 14 - AH = 7Eh FOSSIL - INSTALL AN EXTERNAL APPLICATION FUNCTION

AL = code assigned to external application
ES:DX -> entry point
Return: AX = 1954h
BL = code assigned to application (same as input AL)
DH = 00h failed
01h successful

INT 14 - AH = 7Fh FOSSIL - REMOVE AN EXTERNAL APPLICATION FUNCTION

AL = code assigned to external application
ES:DX -> entry point
Return: AX = 1954h
BL = code assigned to application (same as input AL)
DH = 00h failed
01h successful

INT 15 - AH = 00h CASSETTE - TURN ON MOTOR (PC,Jr)

Return: CF set on error, AH = 86h if no cassette present

INT 15 - AH = 01h CASSETTE - TURN OFF MOTOR (PC,Jr)

Return: CF set on error, AH = 86h if no cassette present

INT 15 - AH = 02h CASSETTE - READ DATA BLOCKS (PC,Jr)

CX = count of bytes
ES:BX -> data area
Return: CF set on error
AH = status
01h CRC error
02h bad tape signals
04h no data
80h invalid command
86h no cassette present
DX = count of bytes read
ES:BX = pointer past last byte read

INT 15 - AH = 03h CASSETTE - WRITE DATA BLOCKS (PC,Jr)

CX = count of bytes to write
ES:BX -> data area
Return: CF set on error
AH = status (see above)
ES:BX = pointer past last byte written
CX = 0

INT 15 - AH = 0Fh SYSTEM - FORMAT UNIT PERIODIC INTERRUPT (PS ESDI drives only)

AL = phase code
00h reserved
01h surface analysis
02h formatting
Return: CF clear if formatting should continue, set if it should terminate
Note: called during ESDI drive formatting after each cylinder is completed

INT 15 - AX=1000h TopView - "PAUSE" - GIVE UP CPU TIME

Return: after other processes run

INT 15 - AX=1001h TopView - "GETMEM" - ALLOCATE "SYSTEM" MEMORY

BX = number of bytes to allocate
Return: ES:DI -> block of memory

INT 15 - AX=1002h TopView - "PUTMEM" - DEALLOCATE "SYSTEM" MEMORY

ES:DI -> previously allocated block
Return: block freed

INT 15 - AX = 1003h TopView - "PRINTC" - DISPLAY CHARACTER/ATTRIBUTE ON SCREEN

BH = attribute
BL = character
DX = segment of object handle for window
Note: BX=0 does not display anything, it only positions the hardware cursor

INT 15 - AH = 10h TopView - UNIMPLEMENTED IN DV 2.0x

AL = 04h thru 12h
Return: pops up "Programming error" window in DV 2.0x

INT 15 - AX = 1013h TopView - "GETBIT" - DEFINE A 2ND-LEVEL INTERRUPT HANDLER

ES:DI -> FAR service routine
Return: BX = bit mask indicating which bit was allocated
0 if no more bits available

INT 15 - AX = 1014h TopView - "FREEBIT" - UNDEFINE A 2ND-LEVEL INTERRUPT HANDLER

BX = bit mask from INT 15/AX=1013h

INT 15 - AX = 1015h TopView - "SETBIT" - SCHEDULE ONE OR MORE 2ND-LEVEL INTERRUPTS

BX = bit mask for interrupts to post
Return: indicated routines will be called at next ???

INT 15 - AX = 1016h TopView - "ISOBJ" - VERIFY OBJECT HANDLE

ES:DI = possible object handle
Return: BX = -1 if ES:DI is a valid object handle
0 if ES:DI is not

INT 15 - AX = 1017h TopView - UNIMPLEMENTED IN DV 2.00

Return: pops up "Programming error" window in DV 2.00

INT 15 - AX = 1018h TopView - "LOCATE" - FIND WINDOW AT A GIVEN SCREEN LOCATION

BH = column
BL = row
ES = segment of object handle for ???
(0 = use default)
Return: ES = segment of object handle for window which is visible at the
indicated position

INT 15 - AX = 1019h TopView - "SOUND" - MAKE TONE

BX = frequency in Hertz
CX = duration in clock ticks (18.2 ticks/sec)
Return: immediately, tone continues to completion
Notes: if another tone is already playing, the new tone does not start until completion of the previous one. In DV 2.00, it is possible to enqueue about 32 tones before the process is blocked until a note completes. In DV 2.00, the lowest tone allowed is 20 Hz

INT 15 - AX = 101Ah TopView - "OSTACK" - SWITCH TO TASK'S INTERNAL STACK

Return: stack switched

INT 15 - AX = 101Bh TopView - "BEGINC" - BEGIN CRITICAL REGION

Return: task-switching temporarily disabled
Note: will not task-switch until END CRITICAL REGION (AX = 101Ch) called

INT 15 - AX = 101Ch TopView - "ENDC" - END CRITICAL REGION

Return: task-switching enabled

INT 15 - AX = 101Dh TopView - "STOP" - STOP TASK

ES = segment of object handle for task to be stopped
(== handle of main window for that task)
Return: indicated task will no longer get CPU time
Note: at least in DV 2.00, this function is ignored unless the indicated task is the current task.

INT 15 - AX = 101Eh TopView - "START" - START TASK

ES = segment of object handle for task to be started
(== handle of main window for that task)
Return: indicated task is started up again

INT 15 - AX = 101Fh TopView - "DISPEROR" - POP-UP ERROR WINDOW

BX = bit fields
bits 0-12: number of characters to display
bits 13,14: which mouse button may be pressed to remove window
00 = either 01 = left 10 = right 11 = either
bit 15: beep if 1
DS:DI -> text of message
CH = width of error window (0 = default)
CL = height of error window (0 = default)
DX = segment of object handle
Return: BX = status: 1 = left button, 2 = right, 27 = ESC pressed
Note: window remains on-screen until ESC or indicated mouse button is pressed

INT 15 - AX = 1020h TopView - UNIMPLEMENTED IN DV 2.0x

Return: pops up "Programming error" window in DV 2.0x

INT 15 - AX = 1021h TopView - "PGMINT" - INTERRUPT ANOTHER TASK

BX = segment of object handle for task to interrupt
DX:CX = address of FAR routine to jump to next time task is run
Return: nothing???
Note: the current ES, DS, SI, DI, and BP are passed to the FAR routine

INT 15 - AX = 1022h TopView - "GETVER" - GET VERSION

BX = 0
Return: BX nonzero, TopView or compatible loaded
(BL = major version, BH = minor version)
Notes: TaskView returns BX = 0001h, DESQview 2.0 returns BX = 0A01h

INT 15 - AX = 1023h TopView - "POSWIN" - POSITION WINDOW

BX = segment of object handle for parent window within which to
position the window (0 = full screen)
ES = segment of object handle for window to be positioned
DL = bit flags
bits 0,1: horizontal position
00 = current 01 = center 10 = left 11 = right
bits 2,3: vertical position
00 = current 01 = center 10 = top 11 = bottom
bit 4: don't redraw screen if set
bits 5-7 not used
CH = number of columns to offset from position specified by DL
CL = number of rows to offset from position specified by DL
Return: nothing

INT 15 - AX = 1024h TopView - "GETBUF" - GET VIRTUAL SCREEN INFO

BX = segment of object handle for window
(0 = use default)
Return: ES:DI = address of virtual screen
CX = size of virtual screen in bytes
DL = 0 ???
1 ???

INT 15 - AX = 1025h TopView - "USTACK" - SWITCH BACK TO USER'S STACK

Return: stack switched back
Note: call only after INT 15h/AX=101Ah

INT 15 - AH = 10h DESQview (TopView???) - UNIMPLEMENTED IN DV 2.0x

AL = 26h thru 2Ah
Return: pops up "Programming error" window in DV 2.0x

INT 15 - AX = 102Bh DESQview 2.0 (TopView???) - "POSTTASK" - AWAKEN TASK

BX = segment of object handle for task
Return: nothing

INT 15 - AX = 102Ch DESQview 2.0 (TopView???) - START NEW APPLICATION IN NEW PROCESS

ES:DI -> contents of .PIF/.DVP file
BX = size of .PIF/.DVP info
Return: BX = segment of object handle for new task
0 on error

INT 15 - AX = 102Dh DESQview 2.0 - KEYBOARD MOUSE CONTROL

BL = subfunction
00h determine whether using keyboard mouse
01h turn keyboard mouse on
02h turn keyboard mouse off
Return: if BL was 00h,
BL = 0 using real mouse
1 using keyboard mouse

INT 15 - AH = 11h TopView commands

AL = various
Note: in DESQview 2.0x, these function calls are identical to AH=DEh, so
see those below

INT 15 - AH = 12h TopView - SEND MESSAGE - "HANDLE" - RETURN OBJECT HANDLE

BH = 00h
BL = which handle to return
00h handle in DWORD on top of stack
01h current task's window handle
02h given task's mailbox handle (task's handle on stack)
03h current task's mailbox handle
04h given task's keyboard handle (task's handle on stack)
05h current task's keyboard object handle
06h given task's OBJECTQ handle (task's handle on stack)
07h current task's OBJECTQ handle
08h \
thru > return 0000:0000

10h /
Return: DWORD on top of stack is object handle

INT 15 - AH = 12h TopView - SEND MESSAGE - "NEW" - CREATE NEW OBJECT

BH = 01h
BL = object
00h handle is DWORD on top of stack
01h use task's window handle
02h given task's mailbox (task's handle on top of stack)
03h current task's mailbox
04h given task's keyboard (task's handle on top of stack)
05h current task's keyboard object
08h WINDOW class
09h MAILBOX class
0Ah KEYBOARD class
0Bh TIMER object (counts down 32-bit time in 10ms increments)
0Fh POINTER object
10h PANEL object
STACK: (if window object or WINDOW class)
DWORD address to jump to (no new task if high word == 0)
DWORD ??? (doesn't seem to be used)
DWORD bytes for task's private stack (-1 == default of 0100h)
DWORD bytes system memory allocation (0 == none, -1 == default)
DWORD window size, columns
DWORD window size, rows
DWORD length of window title
DWORD address of window title
Return: DWORD on top of stack is new object handle
Note: if a new task is created, it is started with
AX = BX = CX = SI = DI = BP = 0
DX = segment of parent's object handle
DS = ES = SS = segment of private stack (and new task's object handle)

INT 15 - AH = 12h TopView - SEND MESSAGE - "FREE" - FREE AN OBJECT

BH = 02h
BL = object
00h handle in DWORD on top of stack
window: close window and free
timer: free timer
panel: free panel object
pointer: free pointer
01h task's window handle - kills task, never returns
02h given task's mailbox (task's handle on top of stack)
03h current task's mailbox
04h given task's keyboard (task's handle on top of stack)
05h current task's keyboard object

INT 15 - AH = 12h TopView - SEND MESSAGE - "DIR" - GET PANEL FILE DIRECTORY

BX = 0300h
STACK: DWORD handle of panel object
Return: STACK: DWORD length of directory
DWORD address of directory

Format of panel file:
BYTE C0h C3h
BYTE number of panels in file
for each panel in file
8 BYTES blank-padded panel name
DWORD panel offset in file
WORD panel length
data for panels (each panel consists of one or more
window/query/manager streams)
first byte of each panel must be 1Bh, fifth byte must be E5h

INT 15 - AH = 12h TopView - SEND MESSAGE - "ADDR" - GET OBJECT HANDLE

BH = 03h
BL = object
00h handle in DWORD on top of stack
02h sender of last msg read from mailbox (task's handle on stack)
03h sender of last msg read from current task's mailbox
Return: DWORD on stack is handle

INT 15 - AH = 12h TopView - SEND MESSAGE - "READ" - WAIT FOR TIMER TO EXPIRE

BX = 0400h
STACK: DWORD timer's handle
Return: STACK: DWORD time in 1/100 sec since midnight when timer expires

INT 15 - AH = 12h TopView - SEND MESSAGE - "READ" - GET NEXT RECORD

BH = 04h
BL = object
00h handle is DWORD on top of stack
window: read next logical line
mailbox: wait for and get next message

pointer: wait for and get next message
01h read the next logical line from task's default window
02h get next message from mailbox (task's handle on top of stack)
03h get next message from current task's mailbox
04h get the next input from keyboard (handle on top of stack)
05h get the next input from task's default keyboard
06h wait for input from any object in OBJECTQ (handle on stack)
07h wait for input from any object in task's default OBJECTQ
Return: STACK: (if objectq) DWORD handle of object with input
(otherwise) DWORD number of bytes
DWORD address

INT 15 - AH = 12h TopView - SEND MESSAGE - "APPLY" - WRITE PANEL TO WINDOW

BX = 0400h
STACK: DWORD handle of panel object
DWORD window's handle or 0
DWORD length of panel name
DWORD pointer to panel name
Return: STACK: DWORD handle of created keyboard or 0
DWORD handle of window which was used
Notes: status of APPLY may be checked with STATUS message
panel MUST have the following format
first byte must be 1Bh (i.e. must start with a stream)
first opcode in stream must be E5h
single byte arg of opcode is interpreted thus:
bit 7 \ 11 means new window created
bit 6 / 01 means existing window used
bit 5 if set, create a new keyboard and put in field mode
bit 4 if set and bit 5 set, make new keyboard active

INT 15 - AH = 12h TopView - SEND MESSAGE - "WRITE" - WRITE TO OBJECT

BH = 05h
BL = object
00h handle is DWORD on top of stack
timer: start timer to end at a specified time
pointer: move pointer icon to specified position
02h send message by value/status=0 to mbox (task's handle on stack)
03h send message by value/status=0 to current task's mailbox
04h add input buffer to KEYBOARD queue (handle on top of stack)
05h add input buffer to task's default KEYBOARD queue
06h add an object to OBJECTQ (handle on top of stack)
07h add an object to task's default OBJECTQ
STACK: (if mailbox) DWORD length
DWORD address
(if keyboard) DWORD status (such as scan code)
DWORD length
DWORD address
(if objectq) DWORD handle of object to add
(if timer) DWORD 1/100ths seconds since midnight (actually
only accurate to 1/18 sec)
(if pointer) DWORD column relative to origin of window
DWORD row relative to origin of window

INT 15 - AH = 12h TopView - SEND MESSAGE - "WRITE" - WRITE STRING TO WINDOW

BH = 05h
BL = object
00h DWORD on top of stack is window handle
01h write string to task's default window
STACK: DWORD object handle if handle passed on stack
DWORD total length of string (high word == 0)
DWORD address of string to display
Note: service routine will pop stack
Return: indicated actions performed
a. non-control characters are displayed
b. CR/LF/BS/Tab cause the usual cursor movement
c. ESC starts a data structure with additional commands

Data Structure:

```
MAGIC  DB  1Bh
MODE   DB  ?   ; 00h, 01h, 10h, 14h-1Fh legal
LENGTH DW  ?   ; length of remainder in bytes
var-length fields follow, each an OPCODE followed by
    zero or more args
```

MODE 00h (set or display values) "WINDOW STREAM"
Opcodes:args
00h display 20h blanks with the default attribute
01h-1Fh display OPCODE blanks with the default attribute
20h display char with default attribute 20h times
BYTE char to repeat
21h-3Fh display char with default attribute OPCODE-20h times
BYTE char to repeat

40h display 20h blanks with specified attribute
BYTE attribute of blanks
41h-5Fh display OPCODE-40h blanks with specified attribute
BYTE attribute of blanks
60h display next 20h characters
20h BYTES characters to display
61h-7Fh display next OPCODE-60h characters
N BYTES characters to display
80h-87h display N blanks with default attribute
BYTE low 8 bits of 11-bit count (high 3 in low 3 bits of OPCODE)
[000h means 800h]
88h-8Fh display N copies of the character
BYTE low 8 bits of 11-bit count (high 3 in low 3 bits of OPCODE)
[000h means 800h]
BYTE character to repeat
90h-97h display N blanks with specified attribute
BYTE low 8 bits of 11-bit length (high 3 in low 3 bits of OPCODE)
[000h means 800h]
BYTE attribute
98h-9FH display string at logical cursor pos
BYTE low 8 bits of 11-bit length (high 3 in low 3 bits of OPCODE)
[000h means 800h]
N BYTES string to display
A0h set logical cursor row
BYTE row number (0 is top)
A1h set logical cursor column
BYTE column number (0 is leftmost)
A2h set top edge of scrolling region
BYTE row
A3h set left edge of scrolling region
BYTE column
A4h set row of physical window position
BYTE line
A5h set column of physical window position
BYTE column
A6h set height of physical window
BYTE #rows
A7h set width of physical window
BYTE #columns
A8h set viewport row
BYTE row
A9h set viewport column
BYTE column
AAh set virtual screen height
BYTE rows
ABh set virtual screen width
BYTE columns
ACh-AEh unused
AFh ???
BYTE ??? (ANDed with current value of something)
B0h move logical cursor down
BYTE #rows (signed, negative values move up)
B1h move logical cursor right
BYTE #cols (signed, negative values move left)
B2h shift top edge of scrolling region
BYTE #rows (signed)
B3h shift left edge of scrolling region
BYTE #cols (signed)
B4h shift window down
BYTE #lines (signed)
B5h shift window right
BYTE #columns (signed)
B6h expand physical window vertically
BYTE #lines (signed)
B7h expand physical window horizontally
BYTE #columns (signed)
B8h adjust viewport row
BYTE #rows (signed)
B9h adjust viewport column
BYTE #columns (signed)
BAh adjust virtual screen height
BYTE #rows to increase (signed)
BBh adjust virtual screen width
BYTE #cols to increase (signed)
BCh-BFh unused
C0h set logical cursor position
BYTE row number (0 is top border)
BYTE column number (0 is left border)
C1h set top left corner of scrolling region
BYTE row
BYTE column
C2h set window pos

BYTE upper left row (no top border if 0)
BYTE upper left column (no left border if 0)
C3h set current window size
BYTE #rows
BYTE #cols
C4h set upper left corner of viewport (portion of virtual screen displayed in window)
BYTE row
BYTE column
C5h set size of virtual screen
BYTE #rows
BYTE #cols
C6h unused
C7h unused
C8h set logical cursor relative to current position
BYTE number of rows to move down (signed)
BYTE number of columns to move right (signed)
C9h shift top left corner of scrolling region
BYTE #rows (signed)
BYTE #cols (signed)
CAh set window pos relative to current position
BYTE number of rows to shift down (signed)
BYTE number of columns to shift right (signed)
CBh set window size relative to current size
BYTE number of rows to expand (signed)
BYTE number of cols to expand (signed)
CCh shift viewport relative to current position
BYTE rows to shift (signed)
BYTE cols to shift (signed)
CDh resize virtual screen
BYTE #rows to expand (signed)
BYTE #cols to expand (signed)
CEh clear ???
CFh set ???
D0h turn on ??? (default)
D1h turn off ???
D2h turn on ???
D3h turn off ??? (default)
D4h window is visible
D5h window is hidden
D6h window has frame
D7h window unframed
D8h read characters from window (default)
D9h read attributes from window
DAh use logical attributes, which may be remapped
attributes
1 normal text
2 highlighted normal text
3 help text
4 highlighted help text
5 error message
6 highlighted error message
7 emphasized text
8 marked text
9-16 are reverse video versions of 1-8
DBh use physical attributes for characters
DCh enable special actions for control characters (default)
DDh disable special control char handling, all chars displayable by BIOS TTY call
DEh write both character and attribute (default)
DFh write character only, leave attribute untouched
E0h repeat following commands
BYTE number of times
E1h end of commands to repeat, start repeating them
E2h set color
BYTE color
E3h clear virtual screen
E4h redraw window
E5h select menu style
BYTE style
bits 5,4 = 01 use two-letter menu entries for remainder of this stream
E5h (panel file only)
BYTE modifier
bits 7,6 = 11 panel goes in new window
= 01 panel uses existing window
bit 5 = 1 create new keyboard in field mode
bit 4 = 1 make newly-created keyboard active
bits 3-0 unused ???
E6h create new window and perform rest of manipulations in new window
BYTE number of rows
BYTE number of columns

Return: DWORD object handle returned on stack at end
E7h unused
E8h scroll area up (top left corner defined by opcode C1h)
BYTE height
BYTE width
E9h scroll area down (top left corner defined by opcode C1h)
BYTE height
BYTE width
EAh scroll area left (top left corner defined by opcode C1h)
BYTE height
BYTE width
EBh scroll area right (top left corner defined by opcode C1h)
BYTE height
BYTE width
ECh set logical attributes for window contents
BYTE bit flags???
BYTE which attributes to set
bit 7 if set, copy single following byte to indicated attrs
bits 4-6 # of first attribute to change - 1
bits 0-3 # of consecutive attributes to change
N BYTES new attributes
EDh set logical attributes for window frame
BYTE bit flags???
BYTE which attributes to set
bit 7 if set, copy single following byte to indicated attrs
bits 4-6 # of first attribute to change - 1
bits 0-3 # of consecutive attributes to change
N BYTES new attributes
attributes
1 = top left corner
2 = top right corner
3 = bottom left corner
4 = bottom right corner
5 = top edge
6 = bottom edge
7 = left edge
8 = right edge
EEh set characters for window frame
BYTE bit flags???
BYTE which characters to set
bit 7 if set, copy single following byte to indicated chars
bits 4-6 # of first char to change - 1
bits 0-3 # of consecutive chars to change
N BYTES new chars (same relative position as attributes above)
EFh set window name
BYTE length of name
N BYTES name
F0h clear input field to blanks
BYTE field number
F1h fill input field with character
BYTE field number
BYTE char
F2h set color of input field
BYTE field number (1-N)
BYTE attribute
F3h set initial contents of input field
BYTE field number (1-N)
N BYTES enough chars to exactly fill field as defined by op FFh
F4h position cursor to specific input field
BYTE field number (1-N)
F5h change field table entry
BYTE field number
7-8 BYTEs field table entry (see FFh below)
F6h set field type
BYTE field number
BYTE type
F7h ???
N BYTES (one for each field???)
F8h scroll field up a line
BYTE field number
F9h scroll field down a line
BYTE field number
FAh scroll field left
BYTE field number
FBh scroll field right
BYTE field number
FCh set field table header
BYTE number of fields
BYTE screen behavior bits
bit 7 ???
bit 6 set if menu items may be selected via keyboard
bit 5 set if left mouse button may terminate entry

bit 4 set if right mouse button may terminate entry
bit 3 if set, menu fields return ' ' rather than 'Y' or 'N'
bit 2 ???
bits 0,1 = 00 no data returned on read of keyboard
01 data returned as array of chars containing all fields packed together, with menu fields represented by the character 'Y' if selected and 'N' if not selected
10 data returned as variable-length records for all fields
11 data returned as variable-length records for the fields which were modified
BYTE field in which cursor was when entry was terminated (updated by DESQview)
BYTE field in which mouse was when entry was terminated (updated by DESQview)
BYTE color of field currently pointed to during entry
BYTE color of input fields which have been selected
FDh reset modified bit for all fields
FEh reset selected and modified bits for all fields
FFh set up input fields
6 BYTES table header (see FCh above)
the field table entries, one for each field
BYTE start row \
BYTE start column \ if menu selection and start is to
BYTE end row / right or below end, select from kbd only
BYTE end column /
BYTE field type
bits 7,6 = 00 non-entry field
01 echos keystrokes input to make menu selection
10 fill-in field
11 menu selection
bit 5 ???
bit 4 ???
bit 3 ???
bit 2 ???
bit 1 set if field selected
bit 0 set if field modified
BYTE modifier
if type is fill-in, then bit flags to determine behavior
bit 7 if set, beep when field is full
bit 6 move to next field when current field is full
bit 5 if set, enter text from right end (for numbers)
bit 4 if set, force input to uppercase
bit 3 if set, clear old contents on first keystroke
bit 2 ???
bit 1 ???
bit 0 ???
if type is menu selection, first key to press to activate
00h if have to point-&-click or is an extended-ASCII keystroke (only if two-key menus enabled)
BYTE for menu item, color of field after cursor or mouse passes through it
BYTE second key for activating menu selection if field type is C0h (0 = only single key). This byte is present iff two-letter menu entries selected with opcode E5h, and in that case is present regardless of field type
Note: DESQview uses and updates the actual copy of the information which is contained in the stream. Thus this info must remain intact until after the data entry is complete.

MODE 01h "QUERY STREAM" (valid only for those opcodes listed here)
A0h return logical cursor row in next byte
A1h return logical cursor column in next byte
A2h return top row of scrolling region in next byte
A3h return left column of scrolling region in next byte
A4h return row of physical window origin in next byte
A5h return column of physical window origin in next byte
A6h return height of physcial window in next byte
A7h return width of physical window in next byte
A8h return row of viewport origin in next byte
A9h return column of viewport origin in next byte
AAh return height of virtual screen in next byte
ABh return width of virtual screen in next byte
AFh return ??? in next byte
C0h return current logical cursor position in next two bytes
C1h return top left corner of scrolling region in next two bytes
C2h return current window position in next two bytes
C3h return current window size in next two bytes
C4h return current viewport origin in next two bytes
C5h return current virtual screen size in next two bytes
D0h \ overwritten with D0h if ??? on

D1h / D1h if ??? off
D2h \ overwritten with D2h if ??? on
D3h / D3h if ??? off
D4h \ overwritten with D4h if window visible
D5h / D5h if window hidden
D6h \ overwritten with D6h if window has frame
D7h / D7h if window unframed
D8h \ overwritten with D8h if reading characters from window
D9h / D9h if reading attributes from window
DAh \ overwritten with DAh if using logical attributes
DBh / DBh if using physical attributes
DCh \ overwritten with DCh if TTY control char interpretation on
DDh / DDh if TTY control char interpretation off
DEh \ overwritten with DEh if writing both characters and attributes
DFh / DFh if leaving attributes untouched
E2h return current color in next byte
ECh get logical attributes for window contents
  BYTE ???
  BYTE which attributes to get
    bit 7 ???
    bits 4-6 first attribute to get - 1
    bits 0-3 # consecutive attributes
  N BYTES buffer to hold attributes
EDh get logical attributes for window frame
  BYTE ???
  BYTE which attributes to get
    bit 7 ???
    bits 4-6 first attribute to get - 1
    bits 0-3 # consecutive attributes
  N BYTES buffer to hold attributes
EEh get characters for window frame
  BYTE ???
  BYTE which attributes to get
    bit 7 ???
    bits 4-6 first char to get - 1
    bits 0-3 # consecutive chars
  N BYTES buffer to hold chars
EFh return current window name
  BYTE max length of returned name
  N BYTES buffer to hold window name
F3h return contents of input field
  BYTE field number
  N BYTES buffer to hold field contents (size exactly equal to field size)
F5h get field table entry
  BYTE field number
  7-8 BYTES buffer to hold field table entry
F6h get type of a field
  BYTE field number
  BYTE type
FCh get field table header
  6 BYTES buffer to store header

MODE 10h "MANAGER STREAM" (valid only for opcodes listed here)
00h allow window to be moved horizontally
01h allow window to be moved vertically
02h allow window to change width
03h allow window to change height
04h allow window to be scrolled horizontally
05h allow window to be scrolled vertically
06h allow "Close Window" menu selection
07h allow window to be hidden
08h allow "Mark" menu
0Eh allow "Scissors" menu
10h allow DESQview main menu to be popped up
11h allow "Switch Windows" menu
12h allow "Open Window" menu
13h allow "Quit" menu selection
20h-33h opposite of 00h-13h, disallow specified action
40h notify if horizontal position of window changes
41h notify if vertical position of window changes
42h notify if width of window changes
43h notify if height of window changes
44h notify if window scrolled horizontally
45h notify if window scrolled vertically
46h notify if window is closed--program has to clean up and exit itself
47h notify if window is hidden
48h notify if "?" on main menu selected
49h notify if colors changed??? (guess)
4Ah notify if window is made active
4Bh notify if window is switched away from
4Ch notify if video mode changes

4Dh notify if "Scissors" menu "Cut" option selected
4Eh notify if "Scissors" menu "Copy" option selected
4Fh notify if "Scissors" menu "Paste" option selected
50h notify if DESQview main menu popped up
51h notify if DESQview main menu popped down
60h-71h opposite of 40h-51h: don't notify on specified event
84h attach window to parent task's window (both move together)
85h detach window from parent task's window (may move independently)
86h disable background operation
87h enable running in background
88h set minimum size of physical window
  BYTE rows
  BYTE columns
89h set maximum size of physical window
  BYTE rows
  BYTE cols
8Ah set primary asynchronous notification routine
  DWORD address of routine, 0000:0000 means none
    on entry ES:DI = handle of window, DS:SI is secondary routine
      mailbox contains message indicating event
        Opcode
          40h horizontal movement
              DWORD object handle of window
              BYTE new row
              BYTE new col
          41h vertical movement
              DWORD object handle of window
              BYTE new row
              BYTE new col
          42h horizontal size change
              DWORD object handle of window
              BYTE new rows
              BYTE new cols
          43h vertical size change
              DWORD object handle of window
              BYTE new rows
              BYTE new cols
          44h scrolled horizontally
              DWORD object handle of window
              BYTE upper left row visible
              BYTE upper left column visible
              BYTE ???
              BYTE amount moved: >0 right, <0 left, 0 done
          45h scrolled vertically
              DWORD object hande of window
              BYTE upper left row visible
              BYTE upper left column visible
              BYTE ???
              BYTE amount moved: >0 down, <0 up, 0 done
          46h window closed
              DWORD object handle of window
              BYTE mouse pointer row
              BYTE mouse pointer column
              BYTE ???
          47h window hidden
          48h Help for Program selected
              DWORD object handle of window
              BYTE mouse pointer row
              BYTE mouse pointer column
              BYTE ???
          49h colors changed??? (guess)
          4Ah switched to window from another ("raise")
          4Bh switched away from the window ("lower")
          4Ch video mode changed
              BYTE new video mode
          4Dh Scissors/cUt selected
              DWORD object handle of window
              BYTE row of upper left corner
              BYTE column of upper left corner
              BYTE ???
              DWORD handle of mailbox to write???
              BYTE height of region
              BYTE width of region
          4Eh Scissors/Copy selected
              DWORD object handle of window
              BYTE row of upper left corner
              BYTE column of upper left corner
              BYTE ???
              DWORD handle of mailbox to write???
              BYTE height of region
              BYTE width of region
          4Fh Scissors/Paste selected

DWORD object handle of window
BYTE row of upper left corner
BYTE column of upper left corner
BYTE ???
DWORD handle of mailbox to read
BYTE height of region
BYTE width of region
50h main menu popped up
51h main menu popped down
routine should restore all registers before returning
8Bh set secondary async notification routine
DWORD address of routine, passed to primary routine in DS:SI,
rather than called directly
AEh ???
AFh set selected field marker character
BYTE character to display at left edge of selected fields
BCh disable use of cursor pad for navigating menus, maybe other???
BDh enable use of cursor pad for navigating menus, maybe other???
BEh disable ???
BFh enable ???
C0h make current window topmost in system
C1h force current process into foreground
C2h make current window topmost in process
C3h position mouse pointer relative to origin of current field
BYTE rows below upper left corner of field
BYTE columns to right of upper left corner of field
C4h position mouse pointer relative to origin of given field
BYTE field number
BYTE rows below upper left corner of field
BYTE columns to right of upper left corner of field
C5h hide current window
C6h show windows for this process
C7h hide all windows for this process
C8h suspend process and hide all its windows
C9h force current process into background
CAh make current window bottom-most in process
CBh ???
CCh close window
CEh reorder windows
DWORD pointer to null-terminated list of words
each word is segment of object handle for a window

MODES 14h to 1Fh "USER STREAMS"
normally NOPs, but may be defined by SETESC message to invoke FAR
routines, one for each mode number
on entry to handler,
DS:SI = first byte of actual stream (not header)
CX = #bytes in stream
ES:DI = window's handle

INT 15 - AH = 12h TopView - SEND MESSAGE - "SIZEOF" - GET OBJECT SIZE

BH = 08h
BL = object
00h handle in DWORD on top of stack
timer: elapsed time since timer started
pointer: number of messages queued to pointer object
panel: number of panels in panel file
01h total chars in current task's default window
02h number of messages in task's mailbox (task's handle on stack)
03h number of messages in current task's mailbox
04h number of input buffers queued in task's kbd (handle on stack)
05h number of input buffers queued for current task's default kbd
06h number of objects queued in OBJECTQ (task's handle on stack)
07h number of objects queued in current task's OBJECTQ
Return: DWORD on stack is result

INT 15 - AH = 12h TopView - SEND MESSAGE - "LEN" - GET OBJECT LENGTH

BH = 09h
BL = object
00h handle in DWORD on top of stack
window: chars/line
timer: timer remaining before timer expires
01h number of chars/line in current task's default window
Return: DWORD on top of stack is length

INT 15 - AH = 12h TopView - SEND MESSAGE - "ADDTO" - SET OBJECT BITS

BH = 0Ah
BL = object
00h handle is DWORD on top of stack
window: write characters and attributes
timer: start timer for specified interval
pointer: set control flags

01h write characters and attributes to task's default window
02h send message/status by value to mailbox (task's handle on stack)
03h send message/status by value to current task's default mailbox
04h set control flags on KEYBOARD object (handle on top of stack)
05h set control flags on task's default KEYBOARD object
STACK: (if mailbox) DWORD status
DWORD length of message
DWORD address
(if timer) DWORD duration in 1/100 seconds
(if window) DWORD count of characters
DWORD address of characters
DWORD count of attributes
DWORD address of attributes
(otherwise) DWORD bits to set
For keyboard objects, the bits have the following significance:
bit 15 reserved, can't be set
bit 14 unused
bit 13 reserved, can't be set
bit 12-5 unused
bit 4 filter all keys (used with handler established by SETESC)
bit 3 program continues executing while input in progress
bit 2 insert mode active
bit 1 keyboard is active
bit 0 keyboard is in field mode
For pointer objects, the bits have the following significance:
bit 15 reserved, can't be set
bit 14-8 unused
bit 7 mouse pointer is hidden while in window
bit 6 get messages even if window not topmost
bit 5 get messages even if window not foreground
bit 4 mouse button must be held 1/2 second before it "clicks"
bit 3 pointer position is relative to screen origin, not window origin
bit 2 send message on button release as well as button press
bit 1 unused???
bit 0 send message only on button activity, not movement
DV-specific, and INT 15h/AX=DE0Fh must have been called first

INT 15 - AH = 12hTopView - SEND MESSAGE - "SUBFROM" - RESET OBJECT BITS

BH = 0Bh
BL = object
00h handle is DWORD on top of stack
window: write attributes only
mailbox: send message by reference
pointer: reset control flags
01h write attributes only to task's default window
02h send msg/status by reference to mailbox (task's handle on stack)
03h send msg/status by reference to current task's mailbox
04h clear control flags on KEYBOARD object (handle on top of stack)
05h clear control flags on task's default KEYBOARD object
06h remove specific object from OBJECTQ (task's handle on stack)
07h remove specific object from task's default OBJECTQ
STACK: (if mailbox) DWORD status
DWORD length
DWORD address
(if window) DWORD number of attributes to write
DWORD address of attributes
(if objectq) DWORD handle of object to remove
(otherwise) DWORD indicates which bits to clear

INT 15 - AH = 12h TopView - SEND MESSAGE - "OPEN" - OPEN OBJECT

BH = 0Ch
BL = object
00h handle is DWORD on top of stack
window: fill with given character
keyboard: attach to a window
timer: open
pointer: start taking input for window
panel: associate with a panel file
01h fill task's default window with given character
02h open given task's mailbox for input (task's handle on stack)
03h open current task's mailbox
04h attach a KEYBOARD to a window (handle on top of stack)
05h attach task's default KEYBOARD to a window
06h open a task's OBJECTQ (task's handle on top of stack)
07h open current task's OBJECTQ
STACK: (if window) DWORD character to fill with
(if keyboard) DWORD handle of window to attach to
(if pointer) DWORD handle of window to attach to
(if panel) DWORD length of filename
DWORD address of filename
(otherwise) nothing
Notes: special action taken if first byte of panel file name is 1Bh

if first two bytes of panel file "name" are C0hC3h, then the "name" IS the panel file
result code of open may be retrieved with STATUS message

INT 15 - AH = 12h TopView - SEND MESSAGE - "CLOSE" - CLOSE OBJECT

BH = 0Dh
BL = object
00h handle is DWORD on top of stack
timer: close
keyboard: detach from window
pointer: stop taking input
panel: close
02h close given task's mailbox (task's handle on top of stack)
03h close task's default mailbox
04h close KEYBOARD object (handle on top of stack)
05h close task's default KEYBOARD
06h close givent task's OBJECTQ (task's handle on top of stack)
07h close current task's OBJECTQ

INT 15 - AH = 12h TopView - SEND MESSAGE - "ERASE" - ERASE OBJECT

BH = 0Eh
BL = object
00h handle is DWORD on top of stack
window: clear
keyboard: discard input
timer: cancel current interval
pointer: discard all pending messages
01h clear task's default window
02h discard all queued messages in mailbox (handle on top of stack)
03h discard all queued messages in current task's default mailbox
04h discard all input queued to KEYBOARD (handle on top of stack)
05h discard all input queued to task's default KEYBOARD
06h remove all objects from OBJECTQ (task's handle on top of stack)
07h remove all objects from current task's OBJECTQ

INT 15 - Ah = 12h TopView - SEND MESSAGE - "STATUS" - GET OBJECT STATUS

BH = 0Fh
BL = object
00h handle is DWORD on top of stack
timer: is it running?
pointer: return status of last message
panel: verify success of last OPEN or APPLY
02h return status of last msg READ from mailbox (handle on stack)
03h return status of last msg READ from task's default mailbox
04h get status of last msg from task's KEYBOARD (task handle on stk)
05h get status of last msg from task's default KEYBOARD
06h return whether OBJECTQ is open or not (handle on top of stack)
07h return whether task's default OBJECTQ is open or not
Return: DWORD on top of stack is status
Note: if object is a panel object, the status indicates the error code:
14h ???
15h ???
16h invalid panel format
17h panel file already open
95h ???
98h null panel file name

INT 15 - AH = 12h TopView - SEND MESSAGE - "EOF" - GET OBJECT EOF STATUS

BH = 10h
BL = object
00h handle is DWORD on top of stack
01h returns TRUE if logical cursor past end of task's def window
02h return ??? for task's mailbox (task's handle on top of stack)
03h return ??? for current task's mailbox
Return: DWORD on top of stack is status

INT 15 - AH = 12h TopView - SEND MESSAGE - "AT" - POSITION OBJECT CURSOR

BH = 11h
BL = object
00h window's handle is DWORD on top of stack
01h position logical cursor on task's default window
STACK: DWORD column
DWORD row

INT 15 - AH = 12h TopView - SEND MESSAGE - "SETNAME" - ASSIGN NAME TO MAILBOX

BH = 11h
BL = mailbox to name
00h DWORD on top of stack is mailbox handle
02h use given task's mailbox (task's handle on top of stack)
03h use current task's default mailbox
STACK: DWORD length of name
DWORD address of name

INT 15 - AH = 12h TopView - SEND MESSAGE - "SETSCALE" - SET POINTER SCALE FACTOR

BX = 1100h
STACK: DWORD object handle for pointer object
DWORD number of colums to scale pointer position to
DWORD number of rows to scale pointer position to

INT 15 - AH = 12h TopView - SEND MESSAGE - "READN" - GET NEXT N OBJECT BYTES

BH = 12h
BL = object
00h handle is DWORD on top of stack
01h read next N chars/attributes on task's default window
STACK: DWORD count
Return: STACK: DWORD width of screen line
DWORD address
DWORD count actually read

INT 15 AH = 12h - TopView - SEND MESSAGE - "GETSCALE" - GET POINTER SCALE FACTOR

BX = 1200h
STACK: DWORD object handle for pointer
Return: STACK: DWORD pointer pos scaled as if window were this many colums wide
DWORD pointer pos scaled as if window were this many rows high

INT 15 - AH = 12h TopView - SEND MESSAGE - "REDRAW" - REDRAW WINDOW

BH = 13h
BL = window object
00h DWORD on top of stack is handle for window to redraw
01h redraw task's default window

INT 15 - AH = 12h TopView - SEND MESSAGE - "SETICON" - SPECIFY POINTER ICON

BX = 1300h
STACK: DWORD object handle for pointer
DWORD character to use for pointer

INT 15 - AH = 12h TopView - SEND MESSAGE - "SETESC" - SET ESCAPE ROUTINE ADDRESS

BH = 14h
BL = message modifier
00h handle is DWORD on top of stack
01h define user stream
04h intercept keystrokes from KEYBOARD to a window (handle on stack)
05h intercept keystrokes from task's default KEYBOARD to a window
STACK: (if window) DWORD user stream number (14h-1Fh)
DWORD address of FAR user stream handler
(if keyboard) DWORD address of FAR filter function

The keyboard filter function is called when the keyboard is in field mode. On entry,
AL = character
AH = 0 or extended ASCII code if AL = 0
BX = field number
CH = cursor column
CL = cursor row
DL = field type modifier (sixth item in field table entry)
DH = ??? (seventh item in field table entry)
ES:SI = window's handle
(also, in DV 2.00, DS:DI points to the field table entry. This may change in other versions)
The filter function should return
AH = 0 use keystroke
1 ignore keystroke
>1 beep and ignore keystroke

INT 15 - AH = 12h TopView - SEND MESSAGE - "LOCK" - REQUEST EXCLUSIVE ACCESS TO RESOURCE

BH = 14h
BL = object
00h mailbox handle is DWORD on top of stack
02h use given task's mailbox (task's handle on top of stack)
03h use current task's default mailbox
Note: release exclusive access by sending CLOSE message to mailbox
access may be requested multiple times, and requires multiple CLOSEs

INT 15 - AH = 20h PRINT.COM - ??? (AT,XT286,PS50+)

AL = subfunction
00h ???
01h ???
10h setup of SYSREQ routine (OS hook)
11h completion of SYSREQ function (OS hook)
Note: AL = 0,1 set or reset some flags which affect what PRINT does when it tries to access the disk

INT 15 - AH = 21h SYSTEM - POWER-ON SELF-TEST ERROR LOG (PS50+)

AL = subfunction
00h read POST log

01h write POST log
BH = device ID
BL = error code
Return: CF set on error
AH = status (00h OK, 01h list full, 80h invalid cmd, 86h unsupported)
if function 00h:
BX = number of error codes stored
ES:DI -> error log
Note: the log is a series of words, the first byte of which identifies the error code and the second the device.

INT 15 - AH = 40h READ/MODIFY PROFILES (CONVERTIBLE)

AL = subfunction
0: get system profile in CX and BX
1: set system profile from CX and BX
2: get internal modem profile in BX
3: set internal modem profile from BX

INT 15 - AH = 41h SYSTEM - WAIT ON EXTERNAL EVENT (CONVERTIBLE)

AL = condition type
bits 0-2: condition to wait for
0 any external event
1 compare and return if equal
2 compare and return if not equal
3 test and return if not zero
4 test and return if zero
bit 3: reserved
bit 4: 1=port address, 0=user byte
bits 5-7: reserved
BH = condition compare or mask value
BL = timeout value times 55 milliseconds
0 means no timeout
DX = I/O port address if AL bit 4 set
ES:DI -> user byte if AL bit 4 clear

INT 15 - AH = 42h SYSTEM - REQUEST POWER OFF (CONVERTIBLE)

AL = 0 to use system profile
1 to force suspend regardless of system profile

INT 15 - AH = 43h SYSTEM - READ SYSTEM STATUS (CONVERTIBLE)

Return: AL = status bits
bit 0: LCD detached
bit 1: reserved
bit 2: RS232/parallel adapter powered on
bit 3: internal modem powered on
bit 4: power activated by alarm
bit 5: standby power lost
bit 6: external power in use
bit 7: power low

INT 15 - AH = 44h SYSTEM - (DE)ACTIVATE INTERNAL MODEM POWER (CONVERTIBLE)

AL = 0 to power off
1 to power on

INT 15 - AH = 4Fh OS HOOK - KEYBOARD INTERCEPT (AT model 3x9,XT2,XT286,CONV,PS)

AL = scan code
CF set
Return: CF set
AL = scan code
CF clear
scan code should not be used
Note: Called by INT 9 handler to translate scan codes

INT 15 - AH = 80h OS HOOK - DEVICE OPEN (AT,XT2,XT286,PS)

BX = device ID
CX = process type
Return: CF set on error
AH = status

INT 15 - AH = 81h OS HOOK - DEVICE CLOSE (AT,XT2,XT286,PS)

BX = device ID
CX = process type
Return: CF set on error
AH = status

INT 15 - AH = 82h OS HOOK - DEVICE PROGRAM TERMINATE (AT,XT2,XT286,PS)

BX = device ID
Return: CF set on error
AH = status
Note: closes all devices opened with function 80h

INT 15 - AH = 83h SYSTEM - EVENT WAIT (AT,XT286,CONV,PS)

AL = subservice
0 = set interval
1 = cancel
ES:BX -> event flag (bit 7 set when interval expires)
CX:DX = number of microseconds to wait (only accurate to 977 us)
Return: CF set if function already busy

INT 15 - AH = 84h SYSTEM - READ JOYSTICK (AT,XT2,XT286,PS)

DX = subservice
0 get switch settings
Return: AL = switch settings (bits 7-4)
1 read joystick inputs
Return: AX = A(x) value
BX = A(y) value
CX = B(x) value
DX = B(y) value

INT 15 - AH = 85h OS HOOK - SYSTEM REQUEST KEY PRESSED (AT,XT2,XT286,CONV,PS)

AL = 0 press
= 1 release
Return: CF set on error
AH = status
Note: called by keyboard decode routine

INT 15 - AH = 86h SYSTEM - WAIT (AT,XT2,XT286,CONV,PS)

CX,DX = number of microseconds to wait (only accurate to 977 us)
Return: CF clear: after wait elapses
CF set: immediately due to error

INT 15 - AH = 87h EXTENDED MEMORY - BLOCK MOVE (AT,XT286,PS)

CX = number of words to move
ES:SI -> global descriptor table
00h-0Fh zero
10h-11h source segment length in bytes (2*CX-1 or greater)
12h-14h 24-bit linear source address
15h access rights byte (93h)
16h-17h zero
18h-19h destination segment length in bytes (2*CX-1 or greater)
1Ah-1Ch 24-bit linear destination address
1Dh access rights byte (93h)
1Eh-2Fh zero
Return: CF set on error
AH = status
00h source copied into destination
01h parity error
02h interrupt error
03h address line 20 gating failed

INT 15 - AH = 88h EXTENDED MEMORY - GET MEMORY SIZE (AT,XT286,PS)

Return: AX = memory size in K

INT 15 - AH = 89h SYSTEM - SWITCH TO VIRTUAL MODE (AT,XT286,PS50+)

BL = interrupt number of IRQ0 (IRQ1-7 use next 7 interrupts)
BH = interrupt number of IRQ8 (IRQ9-F use next 7 interrupts)
DS:SI -> GDT for protected mode
offset 0h null descriptor
8h GDT descriptor
10h IDT descriptor
18h DS
20h ES
28h SS
30h CS
38h uninitialized, used to build descriptor for BIOS CS
CX = offset into protected-mode CS to jump to
Return: CF set on error
AH = 0FFh error enabling address line 20

INT 15 - AH = 90h OS HOOK - DEVICE BUSY LOOP (AT,XT2,XT286,CONV,PS)

AL = type code
00h: disk
01h: diskette
02h: keyboard
03h: PS/2 pointing device
80h: network
FCh: disk reset
FDh: diskette motor start
FEh: printer
ES:BX -> request block for type codes 80h through BFh
Return: CF set if wait time satisfied
CF clear if driver must perform wait
Note: type codes are allocated as follows:

00-7F non-reentrant devices; OS must arbitrate access
80-BF reentrant devices; ES:BX points to a unique control block
C0-FF wait-only calls, no complementary INT 15/AH=91h call

INT 15 - AH = 91h OS HOOK - SET FLAG AND COMPLETE INTERRUPT (AT,XT2,XT286,CONV,PS)

AL = type code, see AH=90h above
ES:BX -> request block for type codes 80h through BFh
Return: AH = 0

INT 15 - AH = C0h SYSTEM - GET CONFIGURATION (XT after 1/10/86,AT mdl 3x9,CONV,XT286,PS)

Return: CF set if BIOS doesn't support call
ES:BX -> ROM table

```
byte_count  dw   ?   ; number of bytes following
model    db   ?   ; PC=ff, XT=fe or fb, PCjr = fd, etc, etc
submodel          db   ?   ; distingushes between AT and XT/286, etc.
BIOS_rev          db   ?   ; 0 for first release, 1 for 2nd, etc.
featbyte          db   ?   ; 80h = DMA channel 3 used by hd BIOS
                           ; 40h = 2nd 8259 installed
                           ; 20h = Real-Time Clock installed
                           ; 10h = INT 15h/AH=4Fh called upon INT 9h
                           ;  8h = wait for external event supported
                           ;  4h = extended BIOS area allocated at 640K
                           ;  2h = bus is Micro Channel instead of PC
                           ;  1h   reserved
res1      dw   0
res2      dw   0
```

Note: the 1/10/86 XT BIOS returns an incorrect value for featbyte.

INT 15 - AH = C1h SYSTEM - RETURN EXTENDED-BIOS DATA-AREA SEGMENT ADDRESS (PS)

Return: CF set on error
ES = segment of data area

INT 15 - AH = C2h POINTING DEVICE BIOS INTERFACE (PS,DESQview 2.x)

AL = subfunction
00h enable/disable
BH = 00h disable
01h enable
01h reset
Return: BH = device ID
02h set sampling rate
BH = 00h 10/second
01h 20/second
02h 40/second
03h 60/second
04h 80/second
05h 100/second
06h 200/second
03h set resolution
BH = 00h one count per mm
01h two counts per mm
02h four counts per mm
03h eight counts per mm
04h get type
Return: BH = device ID
05h initialize
BH = data package size (1 - 8 bytes)
06h get/set scaling factor
BH = 00h return device status
Return: BL = status
bit 0: right button pressed
bit 1: reserved
bit 2: left button pressed
bit 3: reserved
bit 4: 0 if 1:1 scaling, 1 if 2:1 scaling
bit 5: device enabled
bit 6: 0 if stream mode, 1 if remote mode
bit 7: reserved
CL = resolution (see function 03h)
DL = sample rate, reports per second
01h set scaling at 1:1
02h set scaling at 2:1
07h set device handler address
ES:BX = user device handler
Return: CF set on error
AH = status
00h successful
01h invalid function
02h invalid input
03h interface error
04h need to resend
05h no device handler installed

INT 15 - AH = C3h ENABLE/DISABLE WATCHDOG TIMEOUT (PS50+)

AL = 00h disable
01h enable
BX = timer counter
Return: CF set on error
Note: the watchdog timer generates an NMI

INT 15 - AH = C4h PROGRAMMABLE OPTION SELECT (PS50+)

AL = 00h return base POS register address
01h enable slot
BL = slot number
02h enable adapter
Return: CF set on error
DX = base POS register address (if function 00h)

INT 15 - AX = DE00h DESQview - GET PROGRAM NAME

Return: AX = offset into DESQVIEW.DVO of current program's record:
BYTE length of name
N BYTES name
2 BYTES keys to invoke program (second = 00h if only one key used)
WORD ??? (I see 0 always)
BYTE end flag: 00h for all but last entry, which is FFh

INT 15 - AX = DE01h DESQview - UPDATE "OPEN WINDOW" MENU

Return: nothing
Note: reads DESQVIEW.DVO, disables Open menu if file not in current directory

INT 15 - AX = DE02h DESQview - UNIMPLEMENTED IN DV 2.0x

Return: nothing (NOP in DV 2.0x)

INT 15 - AX = DE03h DESQview - UNIMPLEMENTED IN DV 2.0x

Return: nothing (NOP in DV 2.0x)

INT 15 - AX = DE04h DESQview - GET AVAILABLE COMMON MEMORY

Return: BX = bytes of common memory available
CX = largest block available
DX = total common memory in bytes

INT 15 - AX = DE05h DESQview - GET AVAILABLE CONVENTIONAL MEMORY

Return: BX = K of memory available
CX = largest block available
DX = total conventional memory in K

INT 15 - AX = DE06h DESQview - GET AVAILABLE EXPANDED MEMORY

Return: BX = K of expanded memory available
CX = largest block available
DX = total expanded memory in K

INT 15 - AX = DE07h DESQview - "APPNUM" - GET CURRENT PROGRAM'S NUMBER

Return: AX = number of program as it appears on the "Switch Windows" menu

INT 15 - AX = DE08h DESQview - GET ???

Return: AX = 0 ???
1 ???

INT 15 - AX = DE09h DESQview - UNIMPLEMENTED IN DV 2.00

Return: nothing (NOP in DV 2.00)

INT 15 - AX = DE0Ah DESQview 2.0 - "DBGPOKE" - DISPLAY CHARACTER ON STATUS LINE

BL = character
Return: character displayed, next call will display in next position (which wraps back to the start of the line if off the right edge of screen)
Notes: displays character on bottom line of *physical* screen, regardless of current size of window (even entirely hidden) does not know about graphics display modes, just pokes the characters into display memory

INT 15 - AX = DE0Bh DESQview 2.0 - "APILEVEL" - DEFINE MINIMUM API LEVEL REQUIRED

BL = API level
>2 pops up "You need a newer version" error window in DV 2.00
BH = ???
Return: AX = maximum API level???

INT 15 - AX = DE0Ch DESQview 2.0 - "GETMEM" - ALLOCATE "SYSTEM" MEMORY

BX = number of bytes
Return: ES:DI -> allocated block

INT 15 - AX = DE0Dh DESQview 2.0 - "PUTMEM" - DEALLOCATE "SYSTEM" MEMORY

ES:DI -> previously allocated block
Return: nothing

INT 15 - AX = DE0Eh DESQview 2.0 - FIND MAILBOX BY NAME

ES:DI -> name to find
CX = length of name
Return: BX = 0 not found
1 found

DS:SI = object handle

INT 15 - AX = DE0Fh DESQview 2.0 - ENABLE DESQview EXTENSIONS

Return: AX and BX destroyed (seems to be bug, weren't saved&restored)
Notes: sends a manager stream with opcodes AEh, BDh, and BFh to task's window enables an additional mouse mode

INT 15 - AX = DE10h DESQview 2.0 - "PUSHKEY" - PUT KEY INTO KEYBOARD INPUT STREAM

BH = scan code
BL = character
Return: BX = ??? (sometimes, but not always, same as BX passed in)
Note: a later read will get the keystroke as if it had been typed by the user

INT 15 - AX = DE11h DESQview 2.0 - ENABLE/DISABLE AUTOMATIC JUSTIFICATION OF WINDOW

BL = 0 viewport will not move automatically nonzero viewport will move to keep cursor visible
Return: nothing

INT 15 - AX = DE12h DESQview 2.01 - ???

BX = 0 clear ???
nonzero set ???
Return: nothing

INT 16 - AH = 00h KEYBOARD - READ CHAR FROM BUFFER, WAIT IF EMPTY

Return: AH = scan code
AL = character

INT 16 - AH = 01h KEYBOARD - CHECK BUFFER, DO NOT CLEAR

Return: ZF = 0 character in buffer
AH = scan code
AL = character
ZF = 1 no character in buffer

INT 16 - AH = 02h KEYBOARD - GET SHIFT STATUS

AL = shift status bits
0 = right shift key depressed
1 = left shift key depressed
2 = CTRL depressed
3 = ALT depressed
4 = SCROLL LOCK active
5 = NUM LOCK active
6 = CAPS LOCK active
7 = INSERT state active

INT 16 - AH = 03h KEYBOARD - SET DELAYS (Jr,AT model 339,XT286,PS)

AL = subfunction
0 reset typematic (PCjr)
1 increase initial delay (PCjr)
2 increase continuing delay (PCjr)
3 increase both delays (PCjr)
4 turn off typematic (PCjr)
5 Set typematic rate (AT or PS/2)
BH = 00 - 03 for delays of 250ms, 500ms, 750ms, or 1s
BL = 00 - 1F for typematic rates of 30cps down to 2cps

INT 16 - AH = 04h KEYBOARD - KEYCLICK (Jr,CONV)

AL =
0 click off
1 click on

INT 16 - AH = 05h KEYBOARD - WRITE TO KEYBOARD BUFFER (AT model 339,XT2,XT286,PS)

CH = scan code
CL = character
Return: AL = 1 if buffer full

INT 16 - AH = 10h KEYBOARD - GET ENHANCED KEYSTROKE (AT model 339,XT2,XT286,PS)

Return: AH = scan code
AL = character

INT 16 - AH = 11h KEYBOARD - CHECK ENHANCED KEYSTROKE (AT model 339,XT2,XT286,PS)

Return: ZF = 0 if keystroke available
AH = scan code \ meaningless if ZF = 1
AL = character /
ZF = 1 if kbd buffer empty

INT 16 - AH = 12h KEYBOARD - GET ENHANCED SHIFT FLAGS (AT model 339,XT2,XT286,PS)

Return: AL (same as for AH=02h)
bit 7: Ins ON
bit 6: CapsLock ON
bit 5: NumLock ON
bit 4: ScrollLock ON
bit 3: Either ALT key down
bit 2: Either CTRL key down
bit 1: Left shift key down
bit 0: Right shift key down

AH
bit 7: SysReq key down
bit 6: CapsLock key down
bit 5: NumLock key down
bit 4: ScrollLock key down
bit 3: Right Alt key down
bit 2: Right Ctrl key down
bit 1: Left Alt key down
bit 0: Right Alt key down

INT 17 - AH = 00h PRINTER - OUTPUT CHARACTER

AL = character
DX = printer port (0-3)
Return: AH = status bits
0 = time out
1 = unused
2 = unused
3 = I/O error
4 = selected
5 = out of paper
6 = acknowledge
7 = not busy

INT 17 - AH = 01h PRINTER - INITIALIZE

DX = printer port (0-3)
Return: AH = status (see AH = 00h above)

INT 17 - AH = 02h PRINTER - GET STATUS

DX = printer port (0-3)
Return: AH = status (see AH = 00h above)

INT 18 - TRANSFER TO ROM BASIC

causes transfer to ROM-based BASIC (IBM-PC)
often reboots a compatible; often has no effect at all

INT 19 - DISK BOOT

causes reboot of disk system (no memory test performed)

INT 1A - AH = 00h CLOCK - GET TIME OF DAY

Return: CX:DX = clock count
AL = 0 if clock was read or written (via AH=0,1)
within the current 24-hour period
Otherwise, AL > 0

INT 1A - AH = 01h CLOCK - SET TIME OF DAY

CX:DX = clock count
Return: time of day set

INT 1A - AH - 02h CLOCK - READ REAL TIME CLOCK (AT,XT286,CONV,PS)

Return: CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD

INT 1A - AH = 03h CLOCK - SET REAL TIME CLOCK (AT,XT286,CONV,PS)

CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = 1, if daylight savings; 0 if standard time
Return: CMOS clock set

INT 1A - AH = 04h CLOCK - READ DATE FROM REAL TIME CLOCK (AT,XT286,CONV,PS)

Return: DL = day in BCD
DH = month in BCD
CL = year in BCD
CH = century (19h or 20h)

INT 1A - AH = 05h CLOCK - SET DATE IN REAL TIME CLOCK (AT,XT286,CONV,PS)

DL = day in BCD
DH = month in BCD
CL = year in BCD
CH = century (19h or 20h)
Return: CMOS clock set

INT 1A - AH = 06h CLOCK - SET ALARM (AT,XT286,CONV,PS)

CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
Return: CF set if alarm already set or clock inoperable
INT 4Ah will be called when alarm goes off, every 24 hours until reset

INT 1A - AH = 07h CLOCK - RESET ALARM (AT,XT286,CONV,PS)

Return: alarm disabled

INT 1A - CLOCK - AH = 08h SET RTC ACTIVATED POWER ON MODE (CONVERTIBLE)

CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD

INT 1A - AH = 09h CLOCK - READ RTC ALARM TIME AND STATUS (CONV,PS30)

Return: CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = alarm status
0 alarm not enabled
1 alarm enabled but will not power up system
2 alarm will power up system

INT 1A - AH = 0Ah CLOCK - READ SYSTEM-TIMER DAY COUNTER (XT2,PS)

Return: CF set on error
CX = count of days since Jan 1,1980

INT 1A - AH = 0Bh CLOCK - SET SYSTEM-TIMER DAY COUNTER (XT2,PS)

CX = count of days since Jan 1,1980
Return: CF set on error

INT 1A - AH = 80h SET UP SOUND MULTIPLEXOR (PCjr ONLY)

AL = 0 source is 8253 channel 2
1 source is cassette input
2 source is I/O channel "Audio IN"
3 source is sound generator chip

INT 1B - CTRL-BREAK KEY

This interrupt is called when the keyboard scanner of the IBM machines detects CTRL and BREAK pressed at the same time. It normally points to a short routine in DOS which sets the Ctrl-C flag, thus invoking INT 23h the next time DOS checks for Ctrl-C.

INT 1C - CLOCK TICK

This interrupt is called (in the IBM) at the end of each time-update operation by the time-of-day routines. It normally points to an IRET.

INT 1D -> 6845 VIDEO INIT TABLES

table for modes 0 and 1 \
table for modes 2 and 3 \ each table is 16 bytes long and
table for modes 4,5, and 6 / contains values for 6845 registers
table for mode 7 /
4 words -- size of video RAM for modes 0/1, 2/3, 4/5, and 6/7
8 bytes -- number of columns in each mode
8 bytes -- video controller mode byte for each mode

INT 1E -> DISKETTE PARAMS (BASE TABLE)

(Default at F000:EFC7 in PC and most compatibles)
DB step rate & head unload times
DB head load time & DMA
DB motor off time in clock ticks (36 or 37 typical)
DB sector size in bytes (0->128, 1->256, 2->512, 3->1024)
DB last sector number (8 or 9 typical)
DB inter-sector gap size on read/write (42 typical)
DB data transfer length (255 typical)
DB inter-sector gap size on format (80 typical)
DB sector fill on format (F6h typical)
DB head-settle time ms (typical 25, 1.10->0, 2.10->15, 3.10->1)
DB motor start-up time (1/8 secs) (typical 4, 2.10->2)

INT 1F -> GRAPHICS SET 2

(NOT a vector!) pointer to bitmaps for high 128 chars

INT 20 - Minix - SEND/RECEIVE MESSAGE

AX = process ID of other process
BX = pointer to message
CX = 1 send
2 receive
3 send&receive
Note: the message contains the system call number (numbered as in V7 Unix(tm)) and the call parameters

INT 20 - DOS - PROGRAM TERMINATION

returns to DOS--identical to INT 21/AH=00h

INT 21 - AH = 00h DOS - PROGRAM TERMINATION

Return: never

INT 21 - AH = 01h DOS - KEYBOARD INPUT

Return: AL = character read
Note: ^C/^Break are checked, and INT 23h executed if read character is echoed to standard output

INT 21 - AH = 02h DOS - DISPLAY OUTPUT

DL = character to send to standard output
Note: ^C/^Break are checked, and INT 23h executed if pressed

INT 21 - AH = 03h DOS - AUX INPUT

Return: AL = character read

INT 21 - AH = 04h DOS - AUX OUTPUT

DL = character to send

INT 21 - AH = 05h DOS - PRINTER OUTPUT

DL = character to print

INT 21 - AH = 06h DOS - DIRECT CONSOLE I/O CHARACTER OUTPUT

DL = character <> FFh

INT 21 - AH = 06h DOS - DIRECT CONSOLE I/O CHARACTER INPUT

DL = 0FFh
Return: ZF set = no character
ZF clear = character recieved
AL = character
Notes: Character is echoed to STDOUT if received. ^C/^Break are NOT checked

INT 21 - AH = 07h DOS - DIRECT STDIN INPUT, NO ECHO

Note: same as function 06h for input but char not echoed

INT 21 - AH = 08h DOS - KEYBOARD INPUT, NO ECHO

Return: AL = character
Note: same as function 07h, but ^C/^Break are checked

INT 21 - AH = 09h DOS - PRINT STRING

DS:DX = address of string terminated by "$"
Note: ^C/^Break checked, and INT 23h called if pressed

INT 21 - AH = 0Ah DOS - BUFFERED KEYBOARD INPUT

DS:DX = address of buffer
Note: first byte of buffer must contain maximum length on entry, second byte contains actual length of previous line which may be recalled with the DOS line-editing commands on return the second byte contains actual length, third and subsequent bytes contain the input line.

INT 21 - AH = 0Bh DOS - CHECK STANDARD INPUT STATUS

Return: AL = FFh if character available
00h if no character
Note: ^C/^Break checked, and INT 23h called if pressed

INT 21 - AH = 0Ch DOS - CLEAR KEYBOARD BUFFER

AL must be 1, 6, 7, 8, or 0Ah.
Notes: Flushes all typeahead input, then executes function specified by AL (effectively moving it to AH and repeating the INT 21 call). If AL contains a value not in the list above, the keyboard buffer is flushed and no other action is taken.

INT 21 - AH = 0Dh DOS - DISK RESET

Note: Flushes all disk buffers.

INT 21 - AH = 0Eh DOS - SELECT DISK

DL = new default drive number (0 = A, 1 = B, etc.)
Return: AL = number of logical drives

INT 21 - AH = 0Fh DOS - OPEN DISK FILE

DS:DX = address of FCB
Return: AL = 00h file found
FFh file not found
Note: (DOS 3.x) file opened in compatibility mode

INT 21 - AH = 10h DOS - CLOSE DISK FILE

DS:DX = address of FCB
Return: AL = 00h directory update successful
FFh file not found in directory

INT 21 - AH = 11h DOS - SEARCH FIRST USING FCB

DS:DX = address of FCB
Return: AL = status
00h file found
FFh file not found
Note: If file found, FCB is created at DTA address and set up to OPEN or DELETE it.

INT 21 - AH = 12h DOS - SEARCH NEXT USING FCB

DS:DX = address of FCB
Return: AL = status
00h file found
FFh file not found
Note: If file found, FCB is created at DTA address and set up to OPEN or DELETE it.

INT 21 - AH = 13h DOS - DELETE FILE via FCB

DS:DX = address of FCB with filename field filled with template for deletion ('?' wildcard allowed)

Return: AL = status
00h file found
FFh file not found

INT 21 - AH = 14h DOS - SEQUENTIAL DISK FILE READ

DS:DX = address of FCB
Return: AL = status
0 successful read
1 end of file
2 data transfer area too small
3 partial record, EOF

INT 21 - AH = 15h DOS - SEQUENTIAL DISK RECORD WRITE

DS:DX = address of FCB
Return: AL = status
0 successful write
1 disk full
2 data transfer area too small

INT 21 - AH = 16h DOS - CREATE A DISK FILE

DS:DX = address of FCB
Return: AL = status
00h successful creation
FFh directory full
Note: if file already exists, it is truncated to zero length

INT 21 - AH = 17H DOS - RENAME FILE via FCB

DS:DX = address of FCB
FCB contains new name starting at byte 17h.
Return: AL = status
00h file found
FFh file not found

INT 21 - AH = 18h DOS Internal - UNUSED

Return: AL = 0

INT 21 - AH = 19h DOS - GET DEFAULT DISK NUMBER

Return: AL = current drive number (letter - 'A')

INT 21 - AH = 1Ah DOS - SET DISK TRANSFER AREA ADDRESS

DS:DX = address of buffer

INT 21 - AH = 1Bh DOS - ALLOCATION TABLE INFORMATION

Return: DS:BX points to FAT ID byte for default drive
DX = number of allocation units on disk
AL = number of sectors per allocation unit (cluster)
CX = number of bytes per sector

INT 21 - AH = 1Ch DOS - ALLOCATION TABLE INFORMATION FOR SPECIFIC DEVICE

DL = Drive Number to check
Return: DS:BX points to FAT ID byte
DX = number of allocation units on disk
AL = number of sectors per allocation unit (cluster)
CX = number of bytes per sector

INT 21 - AH = 1Dh DOS Internal - UNUSED

Return: AL = 0

INT 21 - AH = 1Eh DOS Internal - UNUSED

Return: AL = 0

INT 21 - AH = 1Fh DOS Internal - GET DEFAULT DRIVE PARAMETER BLOCK

Return: AL = 00h No Error
FFh Error
DS:BX -> drive parameter block
Note: for DOS 2.x and 3.x, this just invokes function 32h with DL = 0

INT 21 - AH = 20h DOS Internal - UNUSED

Return: AL = 0

INT 21 - AH = 21h DOS - RANDOM DISK RECORD READ

DS:DX = address of FCB
Return: AL = status
0 successful read
1 end of file
2 data transfer area too small
3 partial record, EOF

INT 21 - AH = 22h DOS - RANDOM DISK RECORD WRITE

DS:DX = address of FCB
Return: AL = status (see AH = 21h above)

INT 21 - AH = 23h DOS - GET FILE SIZE

DS:DX = address of unopened FCB with filename and record size fields initialized
Return: AL = status
00h file found
FFh file not found
Note: FCB's random-record field set to number of records (rounded up)

INT 21 - AH = 24h DOS - SET RANDOM RECORD FIELD

DS:DX = address of FCB
Return: Random Record Field of FCB is set to be same as Current Block and Current Record.
Note: FCB must be OPEN already

INT 21 - AH = 25h DOS - SET INTERRUPT VECTOR

AL = interrupt number
DS:DX = new vector to be used for specified interrupt

INT 21 - AH = 26h DOS - CREATE PSP

DX = Segment number to set up PSP at
Return: Current PSP is copied to specified segment
Note: new PSP is updated with memory size information; INTs 22h, 23h, 24h taken from interrupt vector table

INT 21 - AH = 27h DOS - RANDOM BLOCK READ

DS:DX = address of FCB
CX = number of records to be read
Return: AL = status
0 successful read
1 end of file
2 data transfer area too small
3 partial record, EOF

INT 21 - AH = 28h DOS - RANDOM BLOCK WRITE

DS:DX = address of FCB
CX = number of records to be written
if zero, truncate file to current random file position
Return: AL = status
0 successful write
1 disk full
2 data transfer area too small

INT 21 - AH = 29h DOS - PARSE FILENAME

DS:SI -> string to parse
ES:DI -> buffer to fill with unopened FCB
AL = bit mask to control parsing
0 = 0: parsing stops if file separator found
1: leading separator ignored
1 = 0: drive number in FCB set to default drive if not present in string
1: drive number in FCB not changed
2 = 0: filename in FCB set to blanks if no filename in string
1: filename in FCB not changed if string does not contain a filename
3 = 0: extension in FCB set to blanks if no extension in string
1: extension left unchanged
Return: AL = 00h: no wildcards in name or extension
01h: wildcards appeared
FFh: drive specifier invalid
DS:SI -> first byte after parsed string
ES:DI buffer filled with unopened FCB

INT 21 - AH = 2Ah DOS - GET CURRENT DATE

Return: DL = day
DH = month
CX = year
AL = day of the week (0=Sunday, 1=Monday, etc.)

INT 21 - AH = 2Bh DOS - SET CURRENT DATE

DL = day
DH = month
CX = year
Return: AL = 00h if no error
= FFh if bad value sent to routine
Note: DOS 3.3 also sets CMOS clock

INT 21 - AH = 2Bh DESQview - INSTALLATION CHECK

AL = subfunction (DV v2.00+)
01h get version
Return: BX = version (BH = major, BL = minor)
Note: early copies of v2.00 return 0002h
02h get shadow buffer info, and start shadowing

Return: BH = rows in shadow buffer
BL = columns in shadow buffer
DX = segment of shadow buffer
04h get shadow buffer info
Return: BH = rows in shadow buffer
BL = columns in shadow buffer
DX = segment of shadow buffer
05h stop shadowing
CX = 4445h ('DE')
DX = 5351h ('SQ')
Return: AL = FFh if DESQview not installed
Note: in DESQview v1.x, there were no subfunctions; this call only identified whether or not DESQview was loaded

INT 21 - AH = 2Ch DOS - GET CURRENT TIME

Return: CH = hours
CL = minutes
DH = seconds
DL = hundredths of seconds
Note: time is updated approximately every 5/100 second

INT 21 - AH = 2Dh DOS - SET CURRENT TIME

CH = hours
CL = minutes
DH = seconds
DL = hundredths of seconds
Return: AL = 00h if no error
= FFh if bad value sent to routine
Note: DOS 3.3 also sets CMOS clock

INT 21 - AH = 2Eh DOS - SET VERIFY FLAG

DL = 0
AL = 1 VERIFY on 0 VERIFY off

INT 21 - DOS 2+ - GET DISK TRANSFER AREA ADDRESS

AH = 2Fh
Return: ES:BX = address of DTA

INT 21 - DOS 2+ - GET DOS VERSION

AH = 30h
Return: AL = Major Version number (0 for DOS 1.x)
AH = Minor Version number
BH = OEM number
00h IBM
16h DEC
BL:CX = 24-bit user number

INT 21 - AH = 31h DOS 2+ - TERMINATE BUT STAY RESIDENT

AL = exit code
DX = program size, in paragraphs

INT 21 - AH = 32h DOS Internal - GET DRIVE PARAMETER BLOCK

DL = drive number
0 = default, 1 = A, etc.
Return: AL = 0FFh if invalid drive number, else
DS:BX -> drive parameter block.

Structure of DOS Drive Parameter Block:

Offset	Size	Description
00h	BYTE	drive number (0 = A, etc.)
01h	BYTE	unit number within device driver
02h	WORD	number of bytes per sector
04h	BYTE	largest sector number in cluster (one less than sect/clust)
05h	BYTE	log base two of the cluster size
06h	WORD	number of reserved (boot) sectors
08h	BYTE	number of copies of the FAT
09h	WORD	number of root directory entries
0Bh	WORD	first data sector on medium
0Dh	WORD	largest possible cluster number (one more than # data clust)
0Fh	BYTE	number of sectors in one FAT copy
10h	WORD	first sector of root directory
12h	DWORD	address of device driver for this drive
16h	BYTE	media descriptor byte for medium
17h	BYTE	FFh indicates block must be rebuilt (DOS 3.x) 00h indicates block accessed
18h	DWORD	address of next device block, offset = FFFFh indicates last
---DOS 2.x only---		
1Ch	WORD	starting cluster of current directory (0 = root directory)
1Eh	64 BYTEs	ASCIZ current directory path string
---DOS 3.x---		
; this was always:		
1Ch	WORD = 0	probably unused, values left from before
1Eh	WORD = 0FFFFh	block was built

INT 21 - AH = 33h DOS 2+ - EXTENDED CONTROL-BREAK CHECKING

AL = subfunction
00h get state
01h set state
DL = 0 for OFF or 1 for ON
02h internal, called by PRINT.COM (DOS 3.1)
05h internal, return boot drive in DL (1=A:) (not in DOS 3.1)
Return: DL = current BREAK setting if AL = 00h
0 BREAK=OFF
1 BREAK=ON
AL = FFh if error

INT 21 - AH = 34h DOS Internal - RETURN CritSectFlag POINTER

Return: ES:BX -> 1-byte DOS "Critical Section Flag", also known as InDOS flag
Notes: when the critical section flag is nonzero, code within DOS is being executed. It is safe to enter DOS when both the critical section flag and the critical error flag are zero. The critical error flag is the byte after the critical section flag in DOS 2.x, and the byte BEFORE the critical section flag in DOS 3.x (except COMPAQ DOS 3.0, where the critical error lag is located 1AAh bytes BEFORE the critical section flag)

INT 21 - AH = 35h DOS 2+ - GET INTERRUPT VECTOR

AL = interrupt number
Return: ES:BX = value of interrupt vector

INT 21 - AH = 36h DOS 2+ - GET DISK SPACE

DL = drive code (0 = default, 1 = A, 2 = B, etc.)
Return: AX = number of sectors per cluster
or 0FFFFh if invalid drive
BX = number of available clusters
CX = bytes per sector
DX = total clusters
Note: multiply AX * CX * BX for free space on disk
multiply AX * CX * DX for total disk space

INT 21 - AH = 37h DOS Internal - SWITCHAR/AVAILDEV

AL = subfunction
0 Read switch character (returns current character in DL)
1 Set switch character (specify new character in DL)
2 (DOS 2.x only) Read device availability (as set by function AL=3)
3 (DOS 2.x only) Set device availability, where:
DL = 0 means /DEV/ must preceed device names
DL <> 0 means /DEV/ need not preceed device names
Return: DL = Switch character (if AL=0 or 1)
Device availability flag (if AL=2 or 3)
AL=0FFh means the value in AL was not in the range 0-3.

INT 21 - AH = 38h DOS 2+ - GET COUNTRY-DEPENDENT INFORMATION

--DOS 2.x--
AL = 0 get current-country info
DS:DX = segment:offset of buffer for returned info
Return: BX = country code
buffer at DS:DX filled as follows:
bytes 0-1 = date format 0 = USA mm dd yy
1 = Europe dd mm yy
2 = Japan yy mm dd
byte 2 = currency symbol
byte 3 = 00h
byte 4 = thousands separator char
byte 5 = 00h
byte 6 = decimal separator char
byte 7 = 00h
bytes 8-1Fh reserved

--DOS 3.x--
AL = 0 for current country
AL = 01h thru 0FEh for specific country with code <255
AL = 0FFh for specific country with code >= 255
BX = 16-bit country code
DS:DX = segment:offset of buffer for returned info
DX = 0FFFFh if setting country code, rather than getting info
Return: (if DX <> 0FFFFh)
BX = country code
DS:DX filled in:
bytes 0-1 = date format (see above)
bytes 2-6 = currency symbol string, ASCIZ
byte 7 = thousands seaprator char
byte 8 = 00h
byte 9 = decimal separator char
byte 0Ah = 00h
byte 0Bh = date separator char
byte 0Ch = 00h
byte 0Dh = time separator char
byte 0Eh = 00h

byte 0Fh = currency format
bit 2 = set if currency symbol replaces decimal pt
bit 1 = number of spaces between value and curr sym
bit 0 = 0 if currency symbol precedes value 1 if currency symbol follows value
byte 10h = number of digits after decimal in currency
byte 11h = time format
bit 0 = 0 if 12-hour clock
1 if 24-hour clock
bytes 12h-15h = address of case map routine (FAR CALL, AL = char)
byte 16h = data-list separator char
byte 17h = 00h
bytes 18h-21h reserved

If error:
CF set
AX = error code

INT 21 - AH = 39h DOS 2+ - CREATE A SUBDIRECTORY (MKDIR)

DS:DX = address of ASCIZ pathname
Return: CF set on error
AX = Error Code

INT 21 - AH = 3Ah DOS 2+ - REMOVE A DIRECTORY ENTRY (RMDIR)

DS:DX = address of ASCIZ pathname
Return: CF set on error
AX = Error Code

INT 21 - AH = 3Bh DOS 2+ - CHANGE THE CURRENT DIRECTORY (CHDIR)

DS:DX = address of ASCIZ directory name
Return: CF set on error
AX = Error Code

INT 21 - AH = 3Ch DOS 2+ - CREATE A FILE WITH HANDLE (CREAT)

CX = attributes for file
bit 0: read-only
1: hidden
2: system
3: volume label
4: reserved, must be zero (directory)
5: archive bit
7: if set, file is shareable under Novell NetWare
DS:DX = address of ASCIZ filename
Return: CF set on error
AX = error code
CF clear if successful
AX = file handle

INT 21 - AH = 3Dh DOS 2+ - OPEN DISK FILE WITH HANDLE

AL = access code
0 = Read Only
1 = Write Only
2 = Read/Write
AL bits 7-3 = file-sharing modes (DOS 3.x)
bit 7 = inheritance flag, set for no inheritance
bits 4-6 = sharing mode
000 compatibility mode
001 exclusive (deny all)
010 write access denied (deny write)
011 read access denied (deny read)
100 full access permitted (deny none)
bit 3 = reserved, should be zero
DS:DX = address of ASCIZ filename
Return: CF set on error
AX = error code
CF clear if successful
AX = file handle

INT 21 - AH = 3Eh DOS 2+ - CLOSE A FILE WITH HANDLE

BX = file handle
Return: CF set on error
AX = error code

INT 21 - AH = 3Fh DOS 2+ - READ FROM FILE WITH HANDLE

BX = file handle
CX = number of bytes to read
DS:DX = address of buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes read

INT 21 - AH = 40h DOS 2+ - WRITE TO FILE WITH HANDLE

BX = file handle
CX = number of bytes to write

DS:DX -> buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes written
Note: if CX is zero, no data is written, and the file is truncated or extended
to the current position

INT 21 - AH = 41h DOS 2+ - DELETE A FILE (UNLINK)

DS:DX -> ASCIZ name of file to delete
Return: CF set on error
AX = error code

INT 21 - AH = 42h DOS 2+ - MOVE FILE READ/WRITE POINTER (LSEEK)

AL = method value
0 = offset from beginning of file
1 = offset from present location
2 = offset from end of file
BX = file handle
CX:DX = offset in bytes
Return: CF set on error
AX = error code
CF clear if successful
DX:AX = new offset

INT 21 - AH = 43h DOS 2+ - GET/PUT FILE ATTRIBUTES (CHMOD)

AL =
0 get file attributes
1 put file attributes
CX = file attribute bits
0 = read only
1 = hidden file
2 = system file
3 = volume label
4 = subdirectory
5 = written since backup
8 = shareable (Novell NetWare)
DS:DX -> ASCIZ file name
Return: CF set on error
AX = error code
CX = file attributes on get

INT 21 - AX = 4400h DOS 2+ - IOCTL - GET DEVICE INFORMATION

BX = file or device handle
Return: CF set on error
AX = error code
CF clear if successful
DX = device info
If bit 7 set: (character device)
bit 0: console input device
1: console output device
2: NUL device
3: CLOCK$ device
4: device is special
5: binary (raw) mode
6: Not EOF
12: network device (DOS 3.x)
14: can process IOCTL control strings (func 2-5)
If bit 7 clear: (file)
bits 0-5 are block device number
6: file has not been written
12: Network device (DOS 3.x)
15: file is remote (DOS 3.x)

INT 21 - AX = 4401h DOS 2+ - IOCTL - SET DEVICE INFORMATION

BX = device handle
DH = 0
DL = device information to set (bits 0-7 from function 0)
Return: CF set on error
AX = error code

INT 21 - AX = 4402h DOS 2+ - IOCTL - READ CHARACTER DEVICE CONTROL STRING

BX = device handle
CX = number of bytes to read
DS:DX -> buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes read

INT 21 - AX = 4403h DOS 2+ - IOCTL - WRITE CHARACTER DEVICE CONTROL STRING

BX = device handle

CX = number of bytes to write
DS:DX -> buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes written

INT 21 - AX = 4404h DOS 2+ - IOCTL - READ BLOCK DEVICE CONTROL STRING

BL = drive number (0=default)
CX = number of bytes to read
DS:DX -> buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes read

INT 21 - AX = 4405h DOS 2+ - IOCTL - WRITE BLOCK DEVICE CONTROL STRING

BL = drive number (0=default)
CX = number of bytes to write
DS:DX -> buffer
Return: CF set on error
AX = error code
CF clear if successful
AX = number of bytes written

INT 21 - AX = 4406h DOS 2+ - IOCTL - GET INPUT STATUS

BX = file or device handle
Return: AL = FFh device ready
00h device not ready

INT 21 - AX = 4407h DOS 2+ - IOCTL - GET OUTPUT STATUS

BX = file or device handle
Return: AL = FFh device ready
00h device not ready
Note: for DOS 2.x, files are always ready for output

INT 21 - AX = 4408h DOS 3.x - IOCTL - BLOCK DEVICE CHANGEABLE

BL = drive number (0=default)
Return: AX = 00h removable
01h fixed
0Fh invalid drive

INT 21 - AX = 4409h DOS 3.x - IOCTL - BLOCK DEVICE LOCAL

BL = drive number (0=default)
Return: DX = attribute word, bit 12 set if device is remote

INT 21 - AX = 440Ah DOS 3.x - IOCTL - HANDLE LOCAL

BX = file handle
Return: DX = attribute word, bit 15 set if file is remote
Note: if file is remote, Novell Advanced NetWare 2.0 returns the number of
the file server on which the handle is located in CX

INT 21 - AX = 440Bh DOS 3.x - IOCTL - SET SHARING RETRY COUNT

CX = delay (default 1)
DX = retry count (default 3)
Return: CF set on error
AX = error code

INT 21 - AX = 440Ch DOS 3.2 - IOCTL - GENERIC

BX = device handle
CH = category code
00h unknown (DOS 3.3)
01h COMn: (DOS 3.3)
03h CON (DOS 3.3)
05h LPTn:
CL = function
45h set iteration count
4Ah select code page
4Ch start code-page preparation
4Dh end code-page preparation
65h get iteration count
6Ah query selected code page
6Bh query prepare list
DS:DX -> parameter block

for CL=45h	WORD	iteration count
for CL=4Ah,4Dh,6Ah	WORD	length of data
	WORD	code page ID
for CL=4Ch	WORD	flags
	WORD	length of remainder of parameter block
	WORD	number of code pages following
	N WORDs	code page 1,...,N
for CL=6Bh	WORD	length of following data
	WORD	number of hardware code pages

N WORDs hardware code pages 1,...,N
WORD number of prepared code pages
N WORDs prepared code pages 1,...,N
Return: CF set on error
AX = error code

INT 21 - AX = 440Dh DOS 3.2 - IOCTL - BLOCK DEVICE REQUEST

BL = drive number (0=default)
CH = category code
08h disk drive
CL = function
40h set device parameters
41h write logical device track
42h format and verify logical device track
60h get device parameters
61h read logical device track
62h verify logical device track
DS:DX -> parameter block
for functions 40h, 60h
BYTE special functions
bit 0 set if function to use current BPB, clear if Device BIOS Parameter Block field contains new default BPB
bit 1 set if function to use track layout fields only must be clear if CL=60h
bit 2 set if all sectors in track same size (should be set)
bits 3-7 reserved
BYTE device type
00h 320K/360K disk
01h 1.2M disk
02h 720K 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
WORD device attributes
bit 0 set if nonremovable medium
bit 1 set if door lock supported
bits 2-15 reserved
WORD number of cylinders
BYTE media type
00h 1.2M disk (default)
01h 320K/360K disk
31 BYTEs device BPB (see function 53h)
WORD number of sectors per track (start of track layout field)
N word pairs: number,size of each sector in track
for functions 41h, 61h
BYTE reserved, must be zero
WORD number of disk head
WORD number of disk cylinder
WORD number of first sector to read/write
WORD number of sectors
DWORD transfer address
for functions 42h, 62h
BYTE reserved, must be zero
WORD number of disk head
WORD number of disk cylinder
Return: CF set on error
AX = error code

INT 21 - AX = 440Eh DOS 3.2 - IOCTL - GET LOGICAL DRIVE MAP

BL = drive number (0=default)
Return: CF set on error
AX = error code
CF clear if successful
AL = 0 block device has only one logical drive assigned
1..26 the last letter used to reference the drive (1=A:,etc)

INT 21 - AX = 440Fh DOS 3.2 - IOCTL - SET LOGICAL DRIVE MAP

BL = physical drive number (0=default)
Return: CF set on error
AX = error code
Note: maps logical drives to physical drives, similar to DOS's treatment of a single physical floppy drive as both A: and B:

INT 21 - AH = 45h DOS 2+ - CREATE DUPLICATE HANDLE (DUP)

BX = file handle to duplicate
Return: CF set on error
AX = error code
CF clear if successful
AX = new file handle

INT 21 - AH = 46h DOS 2+ - FORCE DUPLICATE HANDLE (FORCDUP,DUP2)

BX = Existing file handle
CX = new file handle
Return: CF set on error
AX = error code

INT 21 - AH = 47h DOS 2+ - GET CURRENT DIRECTORY

DL = drive (0=default, 1=A, etc.)
DS:DI points to 64-byte buffer area
Return: CF set on error
AX = error code
Note: the returned path does not include the initial backslash

INT 21 - AH = 48h DOS 2+ - ALLOCATE MEMORY

BX = number of 16-byte paragraphs desired
Return: CF set on error
AX = error code
BX = maximum available
CF clear if successful
AX = segment of allocated memory block

INT 21 - AH = 49h DOS 2+ - FREE MEMORY

ES = Segment address of area to be freed
Return: CF set on error
AX = error code

INT 21 - AH = 4Ah DOS 2+ - ADJUST MEMORY BLOCK SIZE (SETBLOCK)

ES = Segment address of block to change
BX = New size in paragraphs
Return: CF set on error
AX = error code
BX = maximum size possible for the block

INT 21 - AH = 4Bh DOS 2+ - LOAD OR EXECUTE (EXEC)

AL = subfunction
0 = load and execute program
1 = load but do not execute (internal, DOS 3.x & DESQview only)
2 = load but do not execute (internal, DOS 2.x only)
3 = load overlay; do not create PSP
DS:DX = filename
ES:BX = parameter block
AL =
0: WORD segment of environment (0 = use current)
DWORD -> command line
DWORD -> FCB 1
DWORD -> FCB 2
1: WORD segment of environment (0 = use current)
DWORD -> command line
DWORD -> FCB 1
DWORD -> FCB 2
DWORD will hold SS:SP on return
DWORD will hold program entry point (CS:IP) on return
2: WORD segment of environment (0 = use current)
DWORD -> command line
DWORD -> FCB 1
DWORD -> FCB 2
3: WORD segment load address
WORD segment relocation factor
Return: CF set on error
AX = error code
CF clear if successful
if function 1, process ID set to new program's PSP; get with
function 62h
if function 2, new program's initial stack and entry point
returned in registers
Note: DOS 2.x destroys all registers, including SS:SP

Structure of .EXE file header:
WORD 0x4d, 0x5a signature (sometimes 5Ah, 4Dh)
WORD image size remainder (program size mod 512)
WORD file size in pages (program size div 512)
WORD number of relocation items
WORD header size in paragraphs
WORD minimum extra paragraphs needed
WORD maximum extra paragraphs needed
WORD stack segment
WORD stack offset
WORD word checksum of entire file
DWORD initial CS:IP
WORD offset of relocation table
WORD overlay number

INT 21 - AH = 4Ch DOS 2+ - QUIT WITH EXIT CODE (EXIT)

AL = exit code
Return: never returns

INT 21 - AH = 4Dh DOS 2+ - GET EXIT CODE OF SUBPROGRAM (WAIT)

Return: AL = exit code of subprogram (functions 31h or 4Ch)
AH = circumstance which caused termination
0 = Terminate/abort
1 = Control-C
2 = Hard error
3 = Terminate and stay resident

INT 21 - AH = 4Eh DOS 2+ - FIND FIRST ASCIZ (FIND FIRST)

CX = search attributes
DS:DX -> ASCIZ filename
Return: CF set on error
AX = error code
[DTA] = data block
undocumented fields
PC-DOS 3.10
byte 00h: drive letter
bytes 01h-0Bh: search template
byte 0Ch: search attributes
DOS 2.x (and DOS 3.x except 3.1???)
byte 00h: search attributes
byte 01h: drive letter
bytes 02h-0Ch: search template
bytes 0Dh-0Eh: entry count within directory
bytes 0Fh-12h: reserved
bytes 13h-14h: cluster number of parent directory
byte 15h: attribute of file found
bytes 16h-17h: file time
bytes 18h-19h: file date
bytes 1Ah-1Dh: file size
bytes 1Eh-3Ah: ASCIZ filename+extension

INT 21 - AH = 4Fh DOS 2+ - FIND NEXT ASCIZ (FIND NEXT)

[DTA] = data block from last AH = 4Eh/4Fh call
Return: CF set on error
AX = error code
[DTA] = data block, see AH = 4Eh above

INT 21 - AH = 50h DOS Internal - SET PSP SEGMENT

BX = Segment address of new PSP
Note: under DOS 2.xx, this function cannot be invoked inside an INT 28h handler without setting the Critical Error flag

INT 21 - AH = 51h DOS Internal - GET PSP SEGMENT

Return: BX = Current PSP Segment
Note: under DOS 2.xx, this function cannot be invoked inside an INT 28h handler without setting the Critical Error flag

Structure of PSP:

00h	2 BYTEs program exit point
02h	WORD memory size in paragraphs
04h	BYTE unused
05h	5 BYTEs CP/M entry point
0Ah	DWORD terminate address (old INT 22h)
0Eh	DWORD break address (old INT 23h)
12h	DWORD critical error handler (old INT 24h)
16h	WORD parent PSP segment
18h	20 BYTEs DOS 2+ open file table, FFh = unused
2Ch	WORD DOS 2+ environment segment
2Eh	DWORD process's SS:SP
32h	WORD DOS 3.x max open files
36h	DWORD DOS 3.x open file table address
38h	24 BYTEs unused by DOS versions <= 3.3
50h	3 BYTEs DOS function dispatcher (FAR routine)
53h	9 BYTEs unused
5Ch	16 BYTEs FCB #1, filled in from first commandline argument
6Ch	20 BYTEs FCB #2, filled in from second commandline argument
80h	128 BYTEs command tail / default DTA buffer

INT 21 - AH = 52h DOS Internal - GET LIST OF LISTS

Return: ES:BX points to DOS list of lists

List of Lists:

Bytes	Value
-2&-1	Segment of first memory control block
00h-03h	Pointer to first DOS Device Control Block (see function 32h)
04h-07h	Pointer to list of DOS file tables
	DWORD pointer to next file table
	WORD number of files in this table

35h bytes per file
00h-01h number of file handles referring to this file
02h access mode (see function 3Dh)
03h-04h ???
05h-06h device info word (see function 44h/AL=00h)
07h-0Ah Pointer to device driver header if character device
Pointer to DOS Device Control Block if block device (see func 32h for format)
0Bh-0Ch starting cluster of file
0Dh-0Eh file time in packed format
0Fh-10h file date in packed format
11h-14h file size
15h-18h current offset in file
19h-1Ah ???
1Bh-1Ch last cluster read
1Dh-1Fh ???
20h-2Ah filename in FCB format (no path, no period, blank-padded)
2Bh-30h unused??? I see 0 always
31h-32h PSP segment of file's owner
33h-34h unused??? I see 0 always
08h-0Bh Pointer to CLOCK$ device driver, whether installable or resident
0Ch-0Fh Pointer to actual CON: device driver, whether installable or resident
-----DOS 2.x
10h Number of logical drives in system
11h-12h Maximum bytes/block of any block device
13h-16h pointer to first disk buffer
10h bytes control info followed by the 512-byte buffer
00h-03h pointer to next disk buffer, FFFFh if last
04h-07h ???
08h-09h logical sector number
0Ah-0Bh ???
0Ch-0Fh pointer to DOS Device Control Block (see function 32h)
17h Beginning (not a pointer--the real beginning!) of NUL device driver.
This is the first device on DOS's linked list of device drivers.
-----DOS 3.x
10h-11h Maximum bytes/block of any block device
12h-15h Pointer to first disk buffer
10h bytes control info per disk buffer, followed by 512-byte buffer
00h-03h pointer to next disk buffer, FFFFh if last
04h drive (0=A:)
05h flags
bit 7: ???
bit 6: ???
bit 5: contents may be overwritten if set (buffer not dirty)
bit 4: ???
bit 3: sector in data area
bit 2: sector in root directory
bit 1: sector in FAT
bit 0: ???
06h-07h logical sector number
08h ???
09h ???
0Ah-0Dh pointer to DOS Device Control Block (see function 32h)
0Eh-0Fh unused??? (almost always 0)
16h-19h Pointer to array of drive info:
51h bytes per drive, starting with A: ...
00h-3Fh Current path as ASCIZ, starting with 'x:\'
40h-43h ??? I see zeros always
44h ??? I see 40h always
45h-48h pointer to DOS Disk Block for this drive
49h-4Ah starting cluster of current dir, 0 = root, -1 never accessed
4Bh-4Ch ??? I see FFFFh always
4Dh-4Eh ??? I see FFFFh always
4Fh-50h ??? I see 2 always
1Ah-1Dh Pointer to FCB table (if CONFIG.SYS contains FCBS=)
1Eh-1Fh Size of FCB table
20h Number of block devices
21h Value of LASTDRIVE command in CONFIG.SYS (default 5)
22h Beginning (not a pointer--the real beginning!) of NUL device driver.
This is the first device on DOS's linked list of device drivers.
Device driver header format:
DWORD pointer to next driver or -1 if last driver
WORD device attributes
bit 15 character device
bit 14 IOCTL supported
bit 13 output until busy
bit 12 reserved
bit 11 OPEN/CLOSE/RM calls supported
bit 10-5 reserved
bit 4 device is special
bit 3 device is CLOCK
bit 2 device is NUL

bit 1 device is standard output
bit 0 device is standard input
WORD device strategy entry point
WORD device interrupt entry point
8 BYTEs blank-padded character device name
WORD 0 (CD-ROM driver)
BYTE drive letter (CD-ROM driver)
BYTE number of units (CD-ROM driver)

INT 21 - AH = 53h DOS Internal - TRANSLATE BPB

DS:SI points to BPB (Bios Parameter Block)
ES:BP points to area for DOS Disk Block
Note: Translates BPB (Bios Parameter Block, see below) into a DOS Disk Block (see function 32h).
BPB structure:
WORD bytes/sector. Get from DDB bytes 2-3.
BYTE sectors/cluster. Get from (DDB byte 4) + 1
WORD reserved sectors. Get from DDB bytes 6-7
BYTE number of FATs. Get from DDB byte 8
WORD number of root dir entries. Get from DDB bytes 09h-0Ah
WORD total number of sectors. Get from:
((DDB bytes D-E) - 1) * (sectors per cluster (BPB byte 2))
+ (DDB Bytes B-C)
BYTE media descriptor byte. Get from DDB byte 16h
WORD number of sectors/FAT. Get from DDB byte 0Fh
---DOS 3.x---
WORD number of sectors per track
WORD number of heads
DWORD number of hidden sectors
11 BYTEs reserved

INT 21 - AH = 54h DOS 2+ - GET VERIFY FLAG

Return: AL = 0 if flag OFF
AL = 1 if flag ON

INT 21 - AH = 55h DOS Internal - CREATE PSP

DX = segment number at which to set up PSP
Note: Like func 26h but creates "child" PSP rather than copying existing one.

INT 21 - AH = 56h DOS 2+ - RENAME A FILE

DS:DX -> ASCIZ old name
ES:DI -> ASCIZ new name
Return: CF set on error
AX = error code
Note: allows move between directories on same logical volume
(DOS 3.x) allows renaming of directories

INT 21 - AH = 57h DOS 2+ - GET/SET FILE'S DATE/TIME

AL = function code
00h get date and time
Return: CX = time of last write
DX = date of last write
01h set date and time
CX = time to be set
DX = date to be set
BX = file handle
Return: CF set on error
AX = error code

INT 21 - AH = 58h DOS 3.x - GET/SET MEMORY ALLOCATION STRATEGY

AL = function code
0 = get allocation strategy
1 = set allocation strategy
BL = strategy code
0 first fit (use first memory block large enough)
1 best fit (use smallest memory block large enough)
2 last fit (use high part of last usable memory block)
Return: CF set on error
AX = error code
CF clear if successful
AX = strategy code
Note: the Set subfunction accepts any value in BL; 2 or greater means last fit.
the Get subfunction returns the last value set, so programs should check whether the value is >= 2, not just equal to 2.

INT 21 - AH = 59h DOS 3.x - GET EXTENDED ERROR CODE

BX = version code (0000 for DOS 3.x)
Return: AX = extended error code
BH = class of error
BL = suggested action code
CH = locus (where error occurred)
CL, DX, SI, DI, BP, DS, and ES destroyed

Error codes:
- 01h function number invalid
- 02h file not found
- 03h path not found
- 04h too many open files (no handles available)
- 05h access denied
- 06h invalid handle
- 07h memory control block destroyed
- 08h insufficient memory
- 09h memory block address invalid
- 0Ah environment invalid
- 0Bh format invalid
- 0Ch access code invalid
- 0Dh data invalid
- 0Fh invalid drive
- 10h attempted to remove current directory
- 11h not same device
- 12h no more files
- 13h disk write-protected
- 14h unknown unit
- 15h drive not ready
- 16h unknown command
- 17h data error (CRC)
- 18h bad request structure length
- 19h seek error
- 1Ah unknwon media type (non-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 disk change invalid
  - ES:DI -> ASCIZ volume label of required disk
- 23h FCB unavailable
- 24h sharing buffer overflow
- 25h-31h reserved
- 32h Network request not supported (DOS 3.1 + MS Networks)
- 33h Remote computer not listening
- 34h Duplicate name on network
- 35h Network name not found
- 36h Network busy
- 37h Network device no longer exists
- 38h Network BIOS command limit exceeded
- 39h Network adapter hardware error
- 3Ah Incorrect response from network
- 3Bh Unexpected network error
- 3Ch Incompatible remote adapter
- 3Dh Print queue full
- 3Eh Queue not full
- 3Fh Not enough space to print file
- 40h Network name was deleted
- 41h Network: Access denied
- 42h Network device type incorrect
- 43h Network name not found
- 44h Network name limit exceeded
- 45h Network BIOS session limit exceeded
- 46h Temporarily paused
- 47h Network request not accepted
- 48h Print/disk redirection paused (DOS 3.1 + MS Networks)
- 49h-4Fh reserved
- 50h file exists
- 51h reserved
- 52h cannot make directory
- 53h fail on INT 24h
- 54h (DOS 3.3) too many redirections
- 55h (DOS 3.3) duplicate redirection
- 56h (DOS 3.3) invalid password
- 57h (DOS 3.3) invalid parameter
- 58h (DOS 3.3) network write fault

Error Classes:
- 01h out of resource (storage space or I/O channels)
- 02h temporary situation (file or record lock)
- 03h authorization (denied access)
- 04h internal (system software bug)
- 05h hardware failure
- 06h system failure (configuration file missing or incorrect)
- 07h application program error
- 08h not found
- 09h bad format
- 0Ah locked

0Bh media error
0Ch already exists
0Dh unknown
Suggested Action:
01h retry
02h delayed retry
03h prompt user to reenter input
04h abort after cleanup
05h immediate abort
06h ignore
07h retry after user intervention
Error Locus:
01h unknown or not appropriate
02h block device (disk error)
03h network related
04h serial device (timeout)
05h memory related

INT 21 - AH = 5Ah DOS 3.x - CREATE UNIQUE FILE

DS:DX -> ASCIZ directory path name ending with a '\' + 13 bytes to
receive generated filename
CX = file attribute
Return: CF set on error
AX = error code
CF clear if successful
AX = file handle
DS:DX -> path name
Note: The file created is not truly "temporary". It MUST be removed by the user.

INT 21 - AH = 5Bh DOS 3.x - CREATE NEW FILE

DS:DX -> ASCIZ directory path name
CX = file attribute
Return: CF set on error
AX = error code
CF clear if successful
DS:DX -> path name
Note: Unlike function 3Ch, function 5Bh will fail if the file already exists.

INT 21 - AH = 5Ch DOS 3.x - LOCK/UNLOCK FILE ACCESS

AL = 0 if lock
1 if unlock
BX = file handle
CX:DX = starting offset of region to lock
SI:DI = size of region to lock
Return: CF set on error
AX = error code

INT 21 - AX = 5D06h DOS 3.x Internal - GET ADDRESS OF CRITICAL ERROR FLAG

Return: DS:SI -> critical error flag
CX = ???
DX = ???
Notes: this call also does a lot of other work in addition to returning the pointer setting CritErr flag allows use of functions 50h/51h from INT 28h under DOS 2.x by forcing use of correct stack

INT 21 - AH = 5Dh DOS 3.x Internal - ???

AL = subfunction
07h: ???
08h: (used by COMMAND.COM)
09h: (used by COMMAND.COM)

INT 21 - AH = 5D0Ah DOS 3.1+ internal - SET EXTENDED ERROR INFORMATION

DS:DX = address of 11-word error information
words 0 to 7: values of AX,BX,CX,DX,SI,DI,DS,ES that func 59h
will return
words 8 to 10: zero (reserved)

INT 21 - AX = 5E00h DOS 3.1 + Microsoft Networks - GET MACHINE NAME

DS:DX -> buffer for ASCIZ name (16 bytes)
Return: CF set on error
AX = error code
CH = 0 if name not defined
CL = NETBIOS name number
DS:DX -> ASCIZ machine name if CH <> 0

INT 21 - AX = 5E01h DOS 3.1 + Microsoft Networks - SET MACHINE NAME

DS:DX -> ASCIZ name
CL = name number
CH = ???

INT 21 - AX = 5E02h DOS 3.1 + Microsoft Networks - SET PRINTER SETUP

BX = Redirection list index
CX = length of setup string (<= 64)
DS:SI -> string buffer

Return: CF set on error
AX = error code

INT 21 - AX = 5E03h DOS 3.1 + Microsoft Networks - GET PRINTER SETUP

BX = Redirection list index
ES:DI -> string buffer
Return: CF set on error
AX = error code
CX = length of setup string (<= 64)

INT 21 - AX = 5F02h DOS 3.1 + Microsoft Networks - GET REDIRECTION LIST ENTRY

BX = Redirection list index
DS:SI -> 16 char local device name buffer
ES:DI -> 128 char network name buffer
Return: CF set on error
AX = error code
BH = Device status flag (BIT 0 = 0 if valid)
BL = device type (03 if printer, 04 if drive)
CX = stored parameter value (user data)
Note: DX and BP are destroyed by this call!

INT 21 - AX = 5F03h DOS 3.1 + Microsoft Networks - REDIRECT DEVICE

BL = device type
03 = printer device
04 = file device
CX = stored parameter value
DS:SI -> ASCIZ source device name
ES:DI -> destination ASCIZ network path + ASCIZ password
Return: CF set on error
AX = error code

INT 21 - AX = 5F04h DOS 3.1 + Microsoft Networks - CANCEL REDIRECTION

DS:SI -> ASCIZ device name or network path
Return: CF set on error
AX = error code

INT 21 - AH = 60h DOS 3.x Internal - RESOLVE PATH STRING TO FULLY QUALIFIED PATH STRING

DS:SI = relative path string
ES:DI = buffer for fully qualified name
Return: buffer filled with qualified name; may return error code, unknown.

INT 21 - AH = 61h DOS 3.x Internal - UNUSED

Return: AL = 0

INT 21 - AH = 62h DOS 3.x - GET PSP ADDRESS

Return: BX = segment address of PSP

INT 21 - AH = 63h DOS 2.25 only - GET LEAD BYTE TABLE

AL = subfunction
0 = get system lead byte table
1 = set/clear interim console flag
DL = 1/0 to set/clear interim console flag
2 = get interim console flag
Return: CF set on error
AX = error code
DS:SI -> lead byte table (AL = 0)
DL = interim console flag (AL = 2)
Note: does not preserve any registers other than SS:SP

INT 21 - AH = 64h DOS 3.2 Internal - ???

AL = subfunction
00h get ???
Return: DL = ???
01h set ???
DL = ???
02h get and set ???
DL = new ???
Return: DL = old ???

INT 21 - AH = 65h DOS 3.3 - GET EXTENDED COUNTRY INFORMATION

AL = info ID
01h get general internationalization info
02h get pointer to uppercase table
04h get pointer to filename uppercase table
06h get poiner to collating sequence table
BX = code page (-1=global code page)
DX = country ID (-1=current country)
ES:DI -> country information buffer
CX = size of buffer (>= 5)
Return: CF set on error
AX = error code
CF clear if succesful
CX = size of country information returned

ES:DI -> country information:
BYTE info ID
if info ID == 1
WORD size
WORD country ID
WORD code page
34 BYTEs see function 38h
if info ID == 2
DWORD pointer to uppercase table
WORD table size
128 BYTEs uppercase equivalents (if any) of chars 80h-FFh
if info ID == 4
DWORD pointer to collating table
WORD table size
256 BYTEs values used to sort characters 00h-FFh
if info ID == 6
DWORD pointer to filename uppercase table
WORD table size
128 BYTEs uppercase equivalents (if any) of chars 80h-FFh

INT 21 - AH = 66h DOS 3.3 - GET/SET GLOBAL CODE PAGE TABLE

AL = 01h get global code page
Return: AX = error code if carry flag set
BX = active code page
DX = system code page
= 02h set global page
BX = active code page
437 US
850 Multilingual
860 Portugal
863 Canada (French)
865 Norway/Denmark
DX = system code page (active page at boot time)
Return: AX = error code if carry flag set

INT 21 - AH = 67h DOS 3.3 - SET HANDLE COUNT

BX = desired number of handles (max 255)
Return: Carry set if error (and error code in AX)

INT 21 - AH = 68h DOS 3.3 - COMMIT FILE, WRITE ALL BUFFERED DATA TO DISK

BX = file handle
Return: carry flag set on error (and error code in AX)
Note: if BX <= 20, no action is taken

INT 21 - AX = 6C00h DOS 4.0 - EXTENDED OPEN/CREATE

BL = open mode as in AL for normal open
BH = 0WF00000
W = auto commit on write
F = return error rather than doing INT 24h
CX = create attribute
DL = action if file exists/does not exists
bits 7-4 action if file does not exist
0000 fail
0001 create
bits 3-0 action if file exists
0000 fail
0001 open
0010 replace/open
DH = 0
DS:SI -> ASCIZ file name
Return: CF set on error
AX = error code
CF clear if successful
AX = file handle
CX = 1 file opened
2 file created
3 file replaced

INT 21 - AH = B6 Novell NetWare SFT Level II - EXTENDED FILE ATTRIBUTES

AL = subfunction
00h get extended file attributes
01h set extended file attributes
CL = attributes
bit 4: transaction tracking file
5: indexing file (to be implemented)
6: read audit (to be implemented)
7: write audit (to be implemented)
DS:DX -> ASCIZ pathname
Return: CF set on error
AL = error code
FFh file not found
8Ch caller lacks privileges

CL = current extended file attributes

INT 21 - AH = B8h Novell Advanced NetWare 2.0+ - PRINT JOBS

AL = subfunction
00h get default print job flags
01h set default capture flags
02h get specific capture flags
03h set specific print job flags
04h get default local printer
05h set default local printer
06h set capture print queue
07h set capture print job
08h get banner user name
09h set banner user name
CX = buffer size
ES:BX -> buffer
Return: none

INT 21 - AH = BBh Novell NetWare 4.0 - SET END OF JOB STATUS

AL = new EOJ flag
00h disable EOJs
otherwise enable EOJs
Return: AL = old EOJ flag

INT 21 - AH = BCh Novell NetWare 4.6 - LOG PHYSICAL RECORD

AL = flags
bit 0: lock as well as log record
1: non-exclusive lock
BX = file handle
CX:DX = offset
BP = timeout in timer ticks (1/18 sec)
SI:DI = length
Return: AL = error code

INT 21 - AH = BDh Novell NetWare 4.6 - RELEASE PHYSICAL RECORD

BX = file handle
CX:DX = offset
Return: AL = error code

INT 21 - AH = BEh Novell NetWare 4.6 - CLEAR PHYSICAL RECORD

BX = file handle
CX:DX = offset
Return: AL = error code

INT 21 - AH = BFh Novell NetWare 4.6 - LOG RECORD (FCB)

AL = flags
bit 0: lock as well as log record
1: non-exclusive lock
DS:DX -> FCB
BX:CX = offset
BP = timeout in timer ticks (1/18 sec)
SI:DI = length
Return: AL = error code

INT 21 - AH = C0h Novell NetWare 4.6 - RELEASE RECORD (FCB)

DS:DX -> FCB
BX:CX = offset
Return: AL = error code

INT 21 - AH = C1h Novell NetWare 4.6 - CLEAR RECORD (FCB)

DS:DX -> FCB
BX:CX = offset
Return: AL = error code

INT 21 - AH = C2h Novell NetWare 4.6 - LOCK PHYSICAL RECORD SET

AL = flags
bit 1: non-exclusive lock
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = C3h Novell NetWare 4.6 - RELEASE PHYSICAL RECORD SET

Return: AL = error code

INT 21 - AH = C4h Novell NetWare 4.6 - CLEAR PHYSICAL RECORD SET

Return: AL = error code

INT 21 - AH = C5h Novell NetWare 4.6 - SEMAPHORES

AL = subfunction
00h open semaphore
DS:DX -> semaphore name
CL = initial value
01h examine semaphore
Return: CX = semaphore value (sign extended)

DL = open count
02h wait on semaphore
BP = timeout in timer ticks (1/18 sec)
03h signal semaphore
04h close semaphore
CX:DX = semaphore handle (except function 00h)
Return: AL = error code
if function 00h
CX:DX = semaphore handle
BL = open count

INT 21 - AH = C6h Novell NetWare 4.6 - GET OR SET LOCK MODE

AL = subfunction
00h set old "compatibility" mode
01h set new extended locks mode
02h get lock mode
Return: AL = current lock mode

INT 21 - AH = C7h Novell NetWare 4.0 - TTS

AL = subfunction
00h begin transaction (NetWare SFT level II)
Return: AL = error code
01h end transaction (NetWare SFT level II)
Return: AL = error code
CX:DX = transaction reference number
02h TTS available (NetWare SFT level II)
Return: AL = completion code
00h TTS not available
01h TTS available
FDh TTS available but disabled
03h abort transaction (NetWare SFT level II)
Return: AL = error code
04h transaction status
05h get application thresholds
06h set application thresholds
07h get workstation thresholds
08h set workstation thresholds
Return: ???

INT 21 - AH = C8h Novell NetWare 4.0 - BEGIN LOGICAL FILE LOCKING

if function C6h lock mode 00h:
DL = mode
00h no wait
01h wait
if function C6h lock mode 01h:
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = C9h Novell NetWare 4.0 - END LOGICAL FILE LOCKING

Return: AL = error code

INT 21 - AH = CAh Novell NetWare 4.0 - LOG PERSONAL FILE (FCB)

DS:DX -> FCB
if function C6h lock mode 01h:
AL = log and lock flag
00h log file only
01h lock as well as log file
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = CBh Novell NetWare 4.0 - LOCK FILE SET

if function C6h lock mode 00h:
DL = mode
00h no wait
01h wait
if function C6h lock mode 01h:
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = CCh Novell NetWare 4.0 - RELEASE FILE (FCB)

DS:DX -> FCB
Return: none

INT 21 - AH = CDh Novell NetWare 4.0 - RELEASE FILE SET

Return: none

INT 21 - AH = CEh Novell NetWare 4.0 - CLEAR FILE (FCB)

DS:DX -> FCB
Return: AL = error code

INT 21 - AH = CFh Novell NetWare 4.0 - CLEAR FILE SET

Return: AL = 00h

INT 21 - AH = D0h Novell NetWare 4.6 - LOG LOGICAL RECORD

DS:DX -> record string
if function C6h lock mode 01h:
AL = flags
bit 0: lock as well as log the record
bit 1: non-exclusive lock
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = D1h Novell NetWare 4.6 - LOCK LOGICAL RECORD SET

if function C6h lock mode 00h:
DL = mode
00h no wait
01h wait
if function C6h lock mode 01h:
BP = timeout in timer ticks (1/18 sec)
Return: AL = error code

INT 21 - AH = D2h Novell NetWare 4.0 - RELEASE LOGICAL RECORD

DS:DX -> record string
Return: AL = error code

INT 21 - AH = D3h Novell NetWare 4.0 - RELEASE LOGICAL RECORD SET

Return: AL = error code

INT 21 - AH = D4h Novell NetWare 4.0 - CLEAR LOGICAL RECORD

DS:DX -> record string
Return: AL = error code

INT 21 - AH = D5h Novell NetWare 4.0 - CLEAR LOGICAL RECORD SET

Return: AL = error code

INT 21 - AH = D6h Novell NetWare 4.0 - END OF JOB

Return: AL = error code

INT 21 - AH = D7h Novell NetWare 4.0 - SYSTEM LOGOUT

Return: AL = error code

INT 21 - AH = DAh Novell NetWare 4.0 - GET VOLUME STATISTICS

DL = volume number
ES:DI -> reply buffer
Return: AL = 00h
reply buffer
WORD sectors/block
WORD total blocks
WORD unused blocks
WORD total directory entries
WORD unused directory entries
16 BYTEs volume name, null padded
WORD removable flag, 0 = not removable

INT 21 - AH = DBh Novell NetWare 4.0 - GET NUMBER OF LOCAL DRIVES

Return: AL = number of local disks

INT 21 - AH = DCh Novell NetWare 4.0 - GET STATION NUMBER

Return: AL = station number
0 if NetWare not loaded or this machine is a non-dedicated server
CX = station number in ASCII

INT 21 - AH = DDh Novell NetWare 4.0 - SET ERROR MODE

DL = error mode
00h display critical I/O errors
01h extended errors for all I/O in AL
02h extended errors for critical I/O in AL
Return: AL = previous error mode

INT 21 - AH = DEh Novell NetWare 4.0 - SET BROADCAST MODE

AL = broadcast mode
00h receive console and workstation broadcasts
01h receive console broadcasts only
02h receive no broadcasts
03h store all broadcasts for retrieval
04h get broadcast mode
05h disable shell tiemr interrupt checks
06h enable shell timer interrupt checks
Return: AL = old broadcast mode

INT 21 - AH = DFh Novell NetWare 4.0 - CAPTURE

AL = subfunction
00h start LPT capture
01h end LPT capture
02h cancel LPT capture
03h flush LPT capture

04h start specific capture
05h end specific capture
06h cancel specific capture
07h flush specific capture
Return: AL = error code

INT 21 - AH = E0h Novell NetWare 4.0 - PRINT SPOOLING

DS:SI -> request buffer
ES:DI -> reply buffer
subfunction in third byte of request buffer
00h spool data to a capture file
01h close and queue capture file
02h set spool flags
03h spool existing file
04h get spool queue entry
05h remove entry from spool queue
06h get printer status
09h create a disk capture file
Return: AL = error code

INT 21 - AH = E1h Novell NetWare 4.0 - BROADCAST MESSAGES

DS:SI -> request buffer
ES:DI -> reply buffer
subfunction in third byte of request buffer
00h send broadcast message
01h get broadcase message
02h disable station broadcasts
03h enable station broadcasts
04h send personal message
05h get personal message
06h open message pipe
07h close message pipe
08h check pipe status
09h broadcast to console
Return: AL = error code

INT 21 - AH = E2h Novell NetWare 4.0 - DIRECTORY FUNCTIONS

DS:SI -> request buffer
ES:DI -> reply buffer
subfunction in third byte of request buffer
00h set directory handle
01h get directory path
02h scan directory information
03h get effective directory rights
04h modify maximum rights mask
05h get volume number
06h get volume name
0Ah create directory
0Bh delete directory
0Ch scan directory for trustees
0Dh add trustee to directory
0Eh delete trustee from directory
0Fh rename directory
10h purge erased files
11h restore erased file
12h allocate permanent directory handle
13h allocate temporary directory handle
14h deallocate directory handle
15h get volume info with handle
16h allocate special temporary directory handle
17h retrieve a short base handle (Advanced NetWare 2.0)
18h restore a short base handle (Advanced NetWare 2.0)
19h set directory information
Return: AL = error code

INT 21 - AH = E3h Novell NetWare 4.0 - CONNECTION CONTROL

DS:SI -> requst buffer
ES:DI -> reply buffer
subfunction in third byte of request buffer
00h login
01h change password
02h map user to station set
03h map object to number
04h map number to object
05h get station's logged information
06h get station's root mask (obsolete)
07h map group name to number
08h map number to group name
09h get memberset M of group G
0Ah enter login area
0Bh
0Ch
0Dh log network message

0Eh get disk utilization (Advanced NetWare 1.0)
0Fh scan file information (Advanced NetWare 1.0)
10h set file information (Advanced NetWare 1.0)
11h get file server information (Advanced NetWare 1.0)
12h
13h get internet address (Advanced NetWare 1.02)
14h login to file server (Advanced NetWare 2.0)
15h get object connection numbers (Advanced NetWare 2.0)
16h get connection information (Advanced NetWare 1.0)
32h create object (Advanced NetWare 1.0)
33h delete object (Advanced NetWare 1.0)
34h rename object (Advanced NetWare 1.0)
35h get object ID (Advanced NetWare 1.0)
36h get object name (Advanced NetWare 1.0)
37h scan object (Advanced NetWare 1.0)
38h change object security (Advanced NetWare 1.0)
39h create propery (Advanced NetWare 1.0)
3Ah delete property (Advanced NetWare 1.0)
3Bh change property security (Advanced NetWare 1.0)
3Ch scan property (Advanced NetWare 1.0)
3Dh read property value (Advanced NetWare 1.0)
3Eh write property value (Advanced NetWare 1.0)
3Fh verify object password (Advanced NetWare 1.0)
40h change object password (Advanced NetWare 1.0)
41h add object to set (Advanced NetWare 1.0)
42h delete object from set (Advanced NetWare 1.0)
43h is object in set? (Advanced NetWare 1.0)
44h close bindery (Advanced NetWare 1.0)
45h open bindery (Advanced NetWare 1.0)
46h get bindery access level (Advanced NetWare 1.0)
47h scan object trustee paths (Advanced NetWare 1.0)
C8h check console priviledges
C9h get file server description strings
CAh set file server date and time
CBh disable file server login
CCh enable file server login
CDh get file server login status
CEh purge all erased files
CFh disable transaction tracking
D0h enable transaction tracking
D1h send console broadcast
D2h clear connection number
D3h down file server
D4h get file system statistics
D5h get transaction tracking statistics
D6h read disk cache statistics
D7h get drive mapping table
D8h read physical disk statistics
D9h get disk channel statistics
DAh get connection's task information
DBh get list of connection's open files
DCh get list of connections using a file
DDh get physical record locks by connection and file
DEh get physical record locks by file
DFh get logical records by connection
E0h get logical record information
E1h get connection's semaphores
E2h get semaphore information
E3h get LAN driver's configuration information
E5h get connection's usage statistics
E6h get object's remaining disk space
E7h get server LAN I/O statistics
E8h get server miscellaneous information
E9h get volume information

Return: AL = error code

INT 21 - AH = E4h Novell NetWare 4.0 - SET FILE ATTRIBUTES (FCB)

CL = file attributes
bit 0: read only
1: hidden
2: system
7: shareable
DX:DX -> FCB

Return: AL = error code

INT 21 - AX = E400h DoubleDos - INSTALLATION CHECK

Return: AL <> 0 if DoubleDos is active

INT 21 - AH = E5h Novell NetWare 4.0 - UPDATE FILE SIZE (FCB)

DS:DX -> FCB

Return: AL = error code

INT 21 - AH = E6h Novell NetWare 4.0 - COPY FILE TO FILE (FCB)

CX:DX = number of bytes to copy
DS:SI -> source FCB
ES:DI -> destination FCB
Return: AL = error code

INT 21 - AH = E7h Novell NetWare 4.0 - GET FILE SERVER DATE AND TIME

DS:DX -> reply buffer
BYTE year - 1900
BYTE month
BYTE day
BYTE hours
BYTE minutes
BYTE seconds
BYTE day of week (0 = Sunday)
Return: AL = error code

INT 21 - AH = E8h Novell NetWare 4.6 - SET FCB RE-OPEN MODE

DL = mode
00h no automatic re-open
01h auto re-open
Return: AL = error code

INT 21 - AH = E9h Novell NetWare 4.6 - SHELL'S "GET BASE STATUS"

AL = subfunction
00h get directory handle
DX = drive number to check (0 = A:)
Return: AL = network pathbase
AH = base flags
00h drive not currently mapped to a base
01h drive is mapped to a permanent base
02h drive is mapped to a temporary base
03h drive exists locally

INT 21 - AH = EAh Novell NetWare 4.6 - RETURN SHELL VERSION

AL = subfunction
00h return code in AL
Return: AL = hardware type
00h IBM PC
01h Victor 9000
01h get workstation environment string
ES:DI -> 40-byte buffer
Return: buffer filled with three null-terminated entries:
major operating system
version
hardware type
Return: AH = 0 if DOS

INT 21 - AH = EAh DoubleDos - TURN OFF TASK SWITCHING

Return: task switching turned off

INT 21 - AH = EBh Novell NetWare 4.6 - LOG FILE

DS:DX -> ASCIZ filename
if function C6h lock mode 01h:
AL = flags
00h log file only
01h lock as well as log file
BP = timeout in timer ticks (1/18 second)
Return: AL = error code

INT 21 - AH = EBh DoubleDos - TURN ON TASK SWITCHING

Return: task switching turned on

INT 21 - EH = ECh Novell NetWare 4.6 - RELEASE FILE

DS:DX -> ASCIZ filename
Return: none

INT 21 - AH = ECh DoubleDos - GET VIRTUAL SCREEN ADDRESS

Return: ES = segment of virtual screen
Note: Screen address can change if task-switching is on!!

INT 21 - AH = EDh Novell NetWare - CLEAR FILE

DS:DX -> ASCIZ filename
Return: AL = error code

INT 21 - AH = EEh Novell NetWare 4.6 - GET PHYSICAL STATION NUMBER

Return: CX:BX:AX = six-byte address

INT 21 - AH = EEh DoubleDos - GIVE AWAY TIME TO OTHER TASKS

AL = number of 55ms time slices to give away
Return: returns after giving away time slices

INT 21 - AH = EFh Novell Advanced NetWare 1.0+ - GET DRIVE INFO

AL = subfunction
00h get drive handle table
01h get drive flag table
02h get drive connection ID table
03h get connection ID table
04h get file server name table
Return: ES:DI -> shell status table

INT 21 - AH = F0h Novell Advanced NetWare 1.0+ - CONNECTION ID

AL = subfunction
00h set preferred connection ID
01h get preferred connection ID
02h get default connection ID
03h LPT capture active
04h set primary connection ID
05h get primary connection ID
DL = preferred file server
Return: AL = selected file server

INT 21 - AH = F1h Novell Advanced NetWare 1.0+ - FILE SERVER CONNECTION

AL = subfunction
00h attach to file server
DL = preferred file server
01h detach from file server
02h logout from file server
Return: AL = completion code

INT 21 - AH = F2h Novell NetWare - ???

???
Return: ???

INT 21 - AH = F3h Novell Advanced NetWare 2.0+ - FILE SERVER FILE COPY

ES:DI -> request string
WORD source file handle
WORD destination file handle
DWORD starting offset in source
DWORD starting offset in destination
DWORD number of bytes to copy
Return: AL = status/error code
CX:DX = number of bytes copied

INT 21 - AH = FFh CED - INSTALLABLE COMMANDS

AL = 0 add installable command
BL = mode - bit 0 = 1 callable from DOS prompt
bit 1 = 1 callable from application
DS:SI -> CR-terminated command name
ES:DI -> FAR routine entry point
AL = 1 remove installable command
DS:SI -> CR-terminated command name
AL = 2 reserved, may be used to test for CED installation
Return: CF set on error
AX = 01h invalid function
02h command not found (subfunction 1 only)
08h insufficient memory (subfunction 0 only)
0Eh bad data (subfunction 0 only)
AH = 0FFh if CED not installed

INT 22 - DOS - TERMINATE ADDRESS

FAR (DWORD) address of routine to be executed when program "returns to DOS". Should NEVER be called directly.

INT 23 - DOS - CONTROL "C" EXIT ADDRESS

Automatically called from keyboard scanner when CTRL-C or CTRL-BREAK is detected. Normally aborts program and returns to DOS, but may be changed.

INT 24 - DOS - FATAL ERROR HANDLER ADDRESS

Automatically called upon detection of unrecoverable I/O error. Normally points to routine in resident part of COMMAND.COM that prints "Abort, Retry, Ignore?" message and takes the reply, but may be overridden if desired.

Provides the following values in registers on entry to interrupt handler:
AH: bit 7 = 0 disk I/O error
= 1 other error -- if block device, bad FAT
-- if char device, code in DI
bit 6 unused
bit 5 = 1 if Ignore allowed, 0 if not (DOS 3.x)
bit 4 = 1 if Retry allowed, 0 if not (DOS 3.x)
bit 3 = 1 if Fail allowed, 0 if not (DOS 3.x)
bit 2 \ disk area of error 00 = DOS area 01 = FAT
bit 1 / 10 = root dir 11 = data area
bit 0 = 1 if write, 0 if read
AL = drive number if AH bit 7 = 1, otherwise undefined
BP:SI = address of device header for which error occurred

block device if high bit of BP:[SI+4] set
low byte of DI:
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
0Dh (DOS 3.x) sharing violation
0Eh (DOS 3.x) lock violation
0Fh (DOS 3.x) invalid disk change
10h (DOS 3.x) FCB unavailable
11h (DOS 3.x) sharing buffer overflow
Handler must return
AL = 00 ignore error
= 01 retry operation
= 02 terminate program through INT 22h
= 03 fail system call in progress (DOS 3.x)

INT 25 - DOS - ABSOLUTE DISK READ (except DOS 4.0/COMPAQ DOS 3.31 >32M partitn)

AL = Drive number (0=A, 1=B, etc)
DS:BX = Disk Transfer Address (buffer)
CX = Number of sectors to read
DX = First relative sector to read
Return: CF set on error
AL = error code issued to INT 24h in low half of DI
AH = 80h if attachment failed to respond
40h if seek operation failed
20h if controller failed
10h if data error (bad CRC)
08h if DMA failure
04h if requested sector not found
03h if write-protected disk
02h if bad address mark
01h if bad command
Note: ORIGINAL FLAGS ON STACK! May destroy all registers except segment regs

INT 25 - DOS 4.0/COMPAQ DOS 3.31 - ABSOLUTE DISK READ (>32M hard-disk partitn)

AL = Drive number (0=A, 1=B, etc)
CX = FFFFh
DS:BX = Packet address
DWORD sector number
WORD number of sectors to read
DWORD transfer address
Return: same as above???
Note: partition is potentially >32M (and requires this form of the call) if bit 1 of device attribute word in device driver is set

INT 26 - DOS - ABSOLUTE DISK WRITE (except DOS 4.0/COMPAQ DOS 3.31 >32M partn)

AL = Drive number (0=A, 1=B, etc)
DS:BX = Disk Transfer Address (buffer)
CX = Number of sectors to write
DX = First relative sector to write
Return: CF set on error
AL = error code issued to INT 24h in low half of DI
AH = same error codes as for INT 25h
Note: ORIGINAL FLAGS ON STACK!

INT 26 - DOS 4.0/COMPAQ DOS 3.31 - ABSOLUTE DISK WRITE (>32M hard-disk partitn)

AL = Drive number (0=A, 1=B, etc)
CX = FFFFh
DS:BX = Packet address
DWORD sector number
WORD number of sectors to write
DWORD transfer address
Return: same as above???
Note: partition is potentially >32M (and requires this form of the call) if bit 1 of device attribute word in device driver is set

INT 27 - DOS - TERMINATE BUT STAY RESIDENT

CS = current program segment
DX = last program byte + 1
Return: never

INT 28 - DOS Internal - KEYBOARD BUSY LOOP

This interrupt is called from inside the "get input from keyboard" routine in DOS, if and only if it is safe to use INT 21 to access the disk at that time. It is used primarily by the PRINT.COM routines and TSR programs, but any number of other routines could be chained to it by saving the

original vector, and calling it with a FAR call (or just JMPing to it) at the end of the new routine. The INT 28h handler may invoke any INT 21h function except functions 00h through 0Ch (and 50h/51h under DOS 2.xx unless DOS CritErr flag is set). Until some program installs its own routine, this interrupt vector simply points to an IRET opcode.

INT 29 - DOS Internal - FAST PUTCHAR

This interrupt is called from the DOS output routines if output is going to a device rather than a file, and the device driver's attribute word has bit 3 (04h) set to "1".

INT 2A - AH = 00h Microsoft Networks - NETWORK INSTALLATION CHECK

Return: AH <> 0 if installed

INT 2A - AX = 0300h Microsoft Networks - CHECK DIRECT I/O

DS:SI -> ASCIZ disk device name
Return: CF clear if allowed

INT 2A - AH = 04h Microsoft Networks - EXECUTE NETBIOS

AL = 0 for error retry, 1 for no retry
ES:BX -> NCB
Return: AX = 0 for no error
AH = 1, AL = error code

INT 2A - AX = 0500h Microsoft Networks - GET NETWORK RESOURCE INFORMATION

Return: AX = reserved
BX = number of network names
CX = number of commands
DX = number of sessions

INT 2A - AH = 06h NETBIOS 1.10 - NETWORK PRINT-STREAM CONTROL

???
Return: ???

INT 2A - ???

AX = 2001h
???
Return: ???
Note: intercepted by DESQview 2.0

INT 2A - AH = 80h Microsoft Networks - BEGIN DOS CRITICAL SECTION

AL = 1 to 6

INT 2A - AH = 81h Microsoft Networks - END DOS CRITICAL SECTION

AL = 1 to 6

INT 2A - AH = 82h Microsoft Networks - SERVER HOOK

???
Return: ???
Note: Called by the INT 21h function dispatcher for function 0 and functions greater than 0Ch except 59h

INT 2A - AH = 84h Microsoft Networks - KEYBOARD BUSY LOOP

Note: similar to DOS's INT 28h

INT 2B - Internal routine for MSDOS (IRET)

INT 2C - Internal routine for MSDOS (IRET)

INT 2D - Internal routine for MSDOS (IRET)

INT 2E - DOS 2+ Internal - EXECUTE COMMAND

DS:SI -> counted CR-terminated command string

The top-level command.com executes the command; all registers are destroyed as in EXEC (INT 21/AH=4Bh).

INT 2F - TSR ident

Notes: AH identifies which program is to handle the interrupt
00h-7Fh reserved for DOS
C0h-FFh reserved for applications
AL is the function code
This is a general mechanism for verifying the presence of a TSR and communicating with it.

INT 2F - BMB Compuscience Canada Utilities Interface

AH = xx (dynamically assigned based upon a search for a multiplex
number which doesn't answer installed)
AL = 00h install check
ES:DI = EBEB:BEBE
Return: AL = 00h - not installed
01h - not installed, no ok to install
FFh - installed and is ES:DI=EBEB:BEBE then ES:DI will point
to a string 'BMB xxxx' where xxxx is a product name and
version

INT 2F - AX = 0100h Multiplexor - PRINT - INSTALLATION CHECK

Return: AL =
00h not installed, OK to install
01h not installed, not OK to install
FFh installed

INT 2F - AX = 0101h Multiplexor - PRINT - SUBMIT FILE

DS:DX -> packet
BYTE level (must be 0)
DWORD pointer to ASCIZ filename (no wildcards)
Return: CF set on error
AX = error code

INT 2F - AX = 0102h Multiplexor - PRINT - REMOVE FILE

DS:DX -> ASCIZ file name (wildcards allowed)
Return: CF set on error
AX = error code

INT 2F - AX = 0103h Multiplexor - PRINT - REMOVE ALL FILES

Return: CF set on error
AX = error code

INT 2F - AX = 0104h Multiplexor - PRINT - HOLD QUEUE/GET STATUS

Return: CF set on error
AX = error code
01h function invalid
02h file not found
03h path not found
04h too many open files
05h access denied
08h queue full
09h spooler busy
0Ch name too long
0Fh drive invalid
DX = error count
DS:SI -> print queue (null-string terminated
list of 64-byte ASCIZ file names)

INT 2F - AX = 0105h Multiplexor - PRINT - RESTART QUEUE

Return: CF set on error
AX = error code

INT 2F - AX = 0500h Multiplexor - DOS 3.x CRITICAL ERROR HANDLER - INSTALLATION CHECK

Return: AL = 00h not installed, OK to install
01h not installed, can't install
FFh installed
Note: this set of functions allows a user program to partially or completely override the default critical error handler in COMMAND.COM

INT 2F - AH = 05h Multiplexor - DOS 3.x CRITICAL ERROR HANDLER - HANDLE ERROR

AL = extended error code (not zero)
Return: CF clear
ES:DI -> ASCIZ error message
AL = ???
CF set: use default error handler

INT 2F - AX = 0600h Multiplexor - ASSIGN - INSTALLATION CHECK

Return: AH <> 0 if installed

INT 2F - AX = 0601h Multiplexor - ASSIGN - GET MEMORY SEGMENT

Return: ES = segment of ASSIGN work area

INT 2F - AH = 08h Multiplexor - DRIVER.SYS

???

INT 2F - AX = 1000h Multiplexor - SHARE - INSTALLATION CHECK

Return: AL = 00h not installed, OK to install
01h not installed, not OK to install
FFh installed

INT 2F - AX = 1100h Multiplexor - NETWORK REDIRECTOR - INSTALLATION CHECK

Return: AL = 00h not installed, OK to install
01h not installed, not OK to install
FFh installed

INT 2F - AX = 1101h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1103h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1105h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1106h Multiplexor - NETWORK REDIRECTOR - CLOSE REMOTE FILE

???
Return: ???

INT 2F - AX = 1107h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1108h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1109h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 110Ah Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: CF set on error

INT 2F - AX = 110Bh Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: CF set on error???

INT 2F - AX = 110Ch Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 110Eh Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: ???

INT 2F - AX = 110Fh Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1111h Multiplexor - NETWORK REDIRECTOR - RENAME FILE???

???
Return: ???

INT 2F - AX = 1113h Multiplexor - NETWORK REDIRECTOR - ???

Return: ???

INT 2F - AX = 1116h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1117h Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: ???

INT 2F - AX = 1118h Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: ???

INT 2F - AX = 1119h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 111Bh Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 111Ch Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 111Dh Multiplexor - NETWORK REDIRECTOR - ???

DS???
Return: ???

INT 2F - AX = 111Eh Multiplexor - NETWORK REDIRECTOR - DO REDIRECTION

STACK: WORD function to execute
Return: CF set on error

INT 2F - AX = 111Fh Multiplexor - NETWORK REDIRECTOR - PRINTER SETUP

STACK: WORD function???
Return: CF set on error???

INT 2F - AX = 1120h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1121h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: CF set on error???

INT 2F - AX = 1122h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1123h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: CF set on error???

INT 2F - AX = 1124h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: ???

INT 2F - AX = 1125h Multiplexor - NETWORK REDIRECTOR - ???

STACK: WORD ???
Return: ???
CF set on error???

INT 2F - AX = 1126h Multiplexor - NETWORK REDIRECTOR - ???

???
Return: CF set on error???

INT 2F - AX = 1200h Multiplexor - DOS 3.x internal services - INSTALLATION CHECK

Return: AL = FFh (for compatibility with other INT 2F functions)

INT 2F - AX = 1201h Multiplexor - DOS 3.x internal services - CLOSE FILE???

STACK: WORD ???
Return: BX???
CX???
ES:DI -> ???
Note: can be called only from within DOS

INT 2F - AX = 1202h Multiplexor - DOS 3.x internal services - GET INTERRUPT ADDRESS

STACK: WORD vector number
Return: ES:BX -> interrupt vector
Stack unchanged

INT 2F - AX = 1203h Multiplexor - DOS 3.x internal services - GET DOS DATA SEGMENT

Return: DS = segment of IBMDOS

INT 2F - AX = 1204h Multiplexor - DOS 3.x internal services - NORMALIZE PATH SEPARATOR

STACK: WORD character to normalize
Return: AL = normalized character (forward slash turned to backslash)
Stack unchanged

INT 2F - AX = 1205h Multiplexor - DOS 3.x internal services - OUTPUT CHARACTER

STACK: WORD character to output
Return: Stack unchanged
Note: can be called only from within DOS

INT 2F - AX = 1206h Multiplexor - DOS 3.x internal services - INVOKE CRITICAL ERROR

Return: AL = 0-3 for Abort, Retry, Ignore, Fail
Note: can be called only from within DOS

INT 2F - AX = 1207h Multiplexor - DOS 3.x internal services - MOVE DISK BUFFER???

DS:DI -> disk buffer
Return: buffer moved to end of buffer list
Note: can be called only from within DOS

INT 2F - AX = 1208h Multiplexor - DOS 3.x internal services - DECREMENT WORD

ES:DI -> word to decrement
Return: AX = new value of word
word pointed to by ES:DI decremented, skipping zero

INT 2F - AX = 1209h Multiplexor - DOS 3.x internal services - ???

DS:DI -> disk buffer???
Return: ???
Note: can be called only from within DOS

INT 2F - AX = 120Ah Multiplexor - DOS 3.x internal services - ???

???
Return: ???
Note: can be called only from within DOS

INT 2F - AX = 120Bh Multiplexor - DOS 3.x internal services - ???

ES:DI -> system file table entry???

???
Return: AX = ???
Note: can be called only from within DOS

INT 2F - AX = 120Ch Multiplexor - DOS 3.x internal services - ???

???
Return: ???
Note: can be called only from within DOS

INT 2F - AX = 120Dh Multiplexor - DOS 3.x internal services - GET DATE AND TIME

Return: AX = current date in packed format
DX = current time in packed format
Note: can be called only from within DOS

INT 2F - AX = 120Eh Multiplexor - DOS 3.x internal services - ??? ALL DISK BUFFERS

Return: DS:DI -> first disk buffer
Note: can be called only from within DOS

INT 2F - AX = 120Fh Multiplexor - DOS 3.x internal services - ???

DS:DI -> ???
Return: DS:DI -> ???
Note: can be called only from within DOS
calls on function 1207h

INT 2F - AX = 1210h Multiplexor - DOS 3.x internal services - FIND DIRTY BUFFER

DS:DI -> first disk buffer
Return: DS:DI -> first disk buffer which has clean flag clear
ZF clear if found, set if not found

INT 2F - AX = 1211h Multiplexor - DOS 3.x internal services - NORMALIZE ASCIZ FILENAME

DS:SI -> ASCIZ filename to normalize
ES:DI -> buffer for normalized filename
Return: destination buffer filled with uppercase filename, with slashes turned
to backslashes

INT 2F - AX = 1212h Multiplexor - DOS 3.x internal services - GET LENGTH OF ASCIZ STRING

ES:DI -> ASCIZ string
Return: CX = length of string

INT 2F - AX = 1213h Multiplexor - DOS 3.x internal services - UPPERCASE CHARACTER

STACK: WORD character to convert to uppercase
Return: AL = uppercase character
Stack unchanged

INT 2F - AX = 1214h Multiplexor - DOS 3.x internal services - COMPARE FAR POINTERS

DS:SI = first pointer
ES:DI = second pointer
Return: ZF set if pointers are equal, ZF clear if not equal

INT 2F - AX = 1215h Multiplexor - DOS 3.x internal services - ???

DS:DI -> disk buffer
STACK: WORD ???
Return: Stack unchanged
Note: can be called only from within DOS

INT 2F - AX = 1216h Multiplexor - DOS 3.x internal services - GET ADDRESS OF SYSTEM FCB

BX = system file table entry number
Return: ES:DI -> system file table entry

INT 2F - AX = 1217h Multiplexor - DOS 3.x internal services - SET DEFAULT DRIVE ???

STACK: WORD drive (0 = A:, 1 = B:, etc)
Return: DS:SI -> drive data block for specified drive
Stack unchanged
Note: can be called only from within DOS

INT 2F - AX = 1218h Multiplexor - DOS 3.x internal services - GET ???

Return: DS:SI -> ???

INT 2F - AX = 1219h Multiplexor - DOS 3.x internal services - ???

STACK: WORD drive (0 = default, 1 = A:, etc)
Return: ???
Stack unchanged
Note: can be called only from within DOS calls function 1217h

INT 2F - AX = 121Ah Multiplexor - DOS 3.x internal services - GET FILE'S DRIVE

DS:SI -> filename
Return: AL = drive (0 = default, 1 = A:, etc, FFh = invalid)

INT 2F - AX = 121Bh Multiplexor - DOS 3.x internal services - SET ???

CL = ???
Return: AL = ???
Note: can be called only from within DOS

INT 2F - AX = 121Ch Multiplexor - DOS 3.x internal services - CHECKSUM MEMORY

DS:SI -> start of memory to checksum
CX = number of bytes
DX = initial checksum
Return: DX = checksum
Note: can be called only from within DOS

INT 2F - AX = 121Dh Multiplexor - DOS 3.x internal services - ???

DS:SI -> ???
CX = ???
DX = ???
Return: AX = ???
CX = ???
DX = ???

INT 2F - AX = 121Eh Multiplexor - DOS 3.x internal services - COMPARE FILENAMES

DS:SI -> first ASCIZ filename
ES:DI -> second ASCIZ filename
Return: ZF set if filenames equivalent, ZF clear if not

INT 2F - AX = 121Fh Multiplexor - DOS 3.x internal services - BUILD DRIVE INFO BLOCK

STACK: WORD drive letter
Return: ES:DI -> drive info block (will be overwritten by next call)
Stack unchanged
Note: can be called only from within DOS

INT 2F - AX = 1220 h Multiplexor - DOS 3.x internal services - GET SYSTEM FILE TABLE NUMBER

BX = file handle
Return: CF set on error
AL = 6 (invalid file handle)
CF clear if successful
BYTE ES:[DI] = system file table entry number for file handle

INT 2F - AX = 1221h Multiplexor - DOS 3.x internal services - ???

DS:SI -> ???
Return: ???
Note: can be called only from within DOS

INT 2F - AX = 1222h Multiplexor - DOS 3.x internal services - ???

SS:SI -> ???
Return: nothing???
Note: can be called only from within DOS

INT 2F - AX = 1223h Multiplexor - DOS 3.x internal services - CHECK IF CHARACTER DEVICE???

???
Return: DS:SI -> device driver with same name as ???
Note: can be called only from within DOS

INT 2F - AX = 1224h Multiplexor - DOS 3.x internal services - DELAY

Return: after delay of ??? ms
Note: can be called only from within DOS

INT 2F - AX = 1225h Multiplexor - DOS 3.x internal services - GET LENGTH OF ASCIZ STRING

DS:SI -> ASCIZ string
Return: CX = length of string

INT 2F - AH = 14h Multiplexor - NLSFUNC.COM

???

INT 2F - AX = 1500h Multiplexor - CDROM - INSTALLATION CHECK

BX = 0
Return: BX = number of CDROM drive letters used
CX = strating drive letter (0=A:)
Note: this installation check DOES NOT follow the format used by other software

INT 2F - AX = 1501h Multiplexor - CDROM - GET DRIVE DEVICE LIST

ES:BX -> bufer to hold drive letter list (5 bytes per drive letter)
Return: buffer filled, for each drive letter
BYTE subunit number in driver
DWORD address of device driver header

INT 2F - AX = 1502h Multiplexor - CDROM - GET COPYRIGHT FILE NAME

ES:BX -> 38-byte buffer for name of copyright file
CX = drive number (0=A:)
Return: CF set if drive is not a CDROM drive
AX = 15 (invalid drive)

INT 2F - AX = 1503h Multiplexor - CDROM - GET ABSTRACT FILE NAME

ES:BX -> 38-byte buffer for name of abstract file
CX = drive number (0=A:)
Return: CF set if drive is not a CDROM drive
AX = 15 (invalid drive)

INT 2F - AX = 1504h Multiplexor - CDROM - GET BIBLIOGRAPHIC DOC FILE NAME

ES:BX -> 38-byte buffer for name of bibliographic documentation file
CX = drive number (0=A:)
Return: CF set if drive is not a CDROM drive
AX = 15 (invalid drive)

INT 2F - AX = 1505h Multiplexor - CDROM - READ VTOC

ES:BX -> 2048-byte buffer
CX = drive number (0=A:)
DX = sector index (0=first volume descriptor,1=second,...)
Return: CF set on error
AX = error code (15=invalid drive,21=not ready)
CF clear if successful
AX = volume descriptor type (1=standard,FFh=terminator,0=other)

INT 2F - AX = 1506h Multiplexor - CDROM - TURN DEBUGGING ON

BX = debugging function to enable
Note: reserved for development

INT 2F - AX = 1507h Multiplexor - CDROM - TURN DEBUGGING OFF

BX = debugging function to disable
Note: reserved for development

INT 2F - AX = 1508h Multiplexor - CDROM - ABSOLUTE DISK READ

ES:BX -> buffer
CX = drive number (0=A:)
SI:DI = starting sector number
DX = number of sectors to read
Return: CF set on error
AL = error code (15=invalid drive,21=not ready)

INT 2F - AX = 1509h Multiplexor - CDROM - ABSOLUTE DISK WRITE

ES:BX -> buffer
CX = drive number (0=A:)
SI:DI = starting sector number
DX = number of sectors to write
Note: corresponds to INT 26h and is currently reserved and nonfunctional

INT 2F - AX = 150Ah Multiplexor - CDROM - RESERVED

INT 2F - AX = 150Bh Multiplexor - CDROM 2.00 - DRIVE CHECK

CX = drive number (0=A:)
Return: BX = ADADh if MSCDEX.EXE installed
AX = 0 if drive not supported
<> 0 if supported

INT 2F - AX = 150Ch Multiplexor - CDROM 2.00 - GET MSCDEX.EXE VERSION

Return: BH = major version
BL = minor version
Note: MSCDEX.EXE versions prior to 1.02 return BX=0

INT 2F - AX = 150Dh Multiplexor - CDROM 2.00 - GET CDROM DRIVE LETTERS

ES:BX -> buffer for drive letter list (1 byte per drive)
Return: buffer filled with drive numbers (0=A:). Each byte corresponds to the drive in the same position for function 1501h

INT 2F - AX = 150Eh Multiplexor - CDROM 2.00 - GET/SET VOLUME DESCRIPTOR PREFERENCE

BX = subfunction
00h get preference
DX = 0
Return: DX = preference settings
01h set preference
DH = volume descriptor preference
1 = primary volume descriptor
2 = supplementary volume descriptor
DL = supplementary volume descriptor preference
1 = shift-Kanji
CX = drive number (0=A:)
Return: CF set on error
AX = error code (15=invalid drive,1=invalid function)

INT 2F - AX = 150Fh Multiplexor - CDROM 2.00 - GET DIRECTORY ENTRY

CX = drive number (0=A:)
ES:BX -> ASCIZ path name
SI:DI -> 255-byte buffer for directory entry
Return: CF set on error
AX = error code
CF clear if succesful
AX = disk format (0=High Sierra,1=ISO 9660)

directory entry
BYTE length of directory entry
BYTE length of XAR in LBN's (don't ask me what that means...)
DWORD LBN of data, Intel (little-endian) format

DWORD LBN of data, Motorola (big-endian) format
DWORD length of file, Intal format
DWORD length of file, Motorola format
---High Sierra---
6 BYTEs date and time
BYTE bit flags
BYTE reserved
---ISO 9660---
7 BYTEs data and time
BYTE bit flags
---both formats---
BYTE interleave size
BYTE interleave skip factor
WORD volume set sequence number, Intel format
WORD volume set sequence number, Motorola format
BYTE length of file name
N BYTEs file name
BYTE (optional) padding if filename is odd length
N BYTEs system data

INT 2F - AX = 4300h Multiplexor - XMS - INSTALLATION CHECK

Return: AL = 80h XMS driver installed
AL <> 80h no driver
Note: XMS gives access to extended memory and noncontiguous/nonEMS memory above 640K

INT 2F - AX = 4310h Multiplexor - XMS - GET DRIVER ADDRESS

Return: ES:BX -> driver entry point

Perform a FAR call to the driver entry point with AH set to the function code
AH function
00h Get XMS version number
Return: AX = XMS version (in BCD)
BX = internal revision number
DX = 1 if HMA (1M to 1M + 64K) exists
0 if HMA does not exist
01h Request High Memory Area (1M to 1M + 64K)
DX = memory in bytes (for TSR or device drivers)
FFFFh if application program
Return: AX = 1 success
= 0 failure
BL = error code
02h Release High Memory Area
Return: AX = 1 success
= 0 failure
BL = error code
03h Global enable A20, for using the HMA
Return: AX = 1 success
= 0 failure
BL = error code
04h Global disable A20
Return: AX = 1 success
= 0 failure
BL = error code
05h Local enable A20, for direct access to extended memory
Return: AX = 1 success
= 0 failure
BL = error code
06h Local disable A20
Return: AX = 1 success
= 0 failure
BL = error code
07h Query A20
Return: AX = 1 enabled
= 0 disabled
BL = error code (0 = successful)
08h Query free extended memory, not counting HMA
Return: AX = size of largest extended memory block in K
DX = total extended memory in K
BL = error code
09h Allocate extended memory block
DX = Kbytes needed
Return: AX = 1 success
DX = handle for memory block
= 0 failure
BL = error code
0Ah Free extended memory block
DX = handle of block to free
Return: AX = 1 success
= 0 failure
BL = error code
0Bh Move extended memory block

DS:SI -> EMM structure
DWORD number of bytes to move (must be even)
WORD source handle
DWORD offset into source block
WORD destination handle
DWORD offset into destination block
Note: if either handle is 0000h, the corresponding offset is considered to be an absolute Segment:Offset address in directly addressable memory
Return: AX = 1 success
= 0 failure
BL = error code

0Ch Lock extended memory block
DX = handle of block to lock
Return: AX = 1 success
DX:BX = 32-bit linear address of locked block
= 0 failure
BL = error code

0Dh Unlock extended memory block
DX = handle of block to unlock
Return: AX = 1 success
= 0 failure
BL = error code

0Eh Get handle information
DX = handle for which to get info
Return: AX = 1 success
BH = block's lock count
BL = number of free handles left
DX = block size in K
= 0 failure
BL = error code

0Fh Reallocate extended memory block
DX = handle of block
BX = new size of block in K
Return: AX = 1 success
= 0 failure
BL = error code

10h Request upper memory block (nonEMS memory above 640K)
DX = size of block in paragraphs
Return: AX = 1 success
BX = segment address of UMB
DX = actual size of block
= 0 failure
BL = error code
DX = largest available block

11h Release upper memory block
DX = segment address of UMB to release
Return: AX = 1 success
= 0 failure
BL = error code

Note: HIMEM.SYS requires at least 256 bytes stack

Error codes returned in BL:
80h Function not implemented
81h Vdisk was detected
82h An A20 error occurred
8Eh a general driver error
8Fh unrecoverable driver error
90h HMA does not exist
91h HMA is already in use
92h DX is less than the /HMAMIN= parameter
93h HMA is not allocated
94h A20 line still enabled
A0h all extended memory is allocated
A1h all available extended memory handles are allocated
A2h Invalid handle
A3h Source handle is invalid
A4h Source offset is invalid
A5h Destination handle is invalid
A6h Destination offset is invalid
A7h Length is invalid
A8h Move has an invalid overlap
A9h Parity error occurred
AAh Block is not locked
ABh Block is locked
ACh Block lock count overflowed
ADh Lock failed
B0h Only a smaller UMB is available
B1h No UMB's are available
B2h UMB segment number is invalid

INT 2F - AX = 6400h Multiplexor - SCRNSAV2.COM - INSTALLATION CHECK

Return: AL = 00h not installed
FFh installed
Note: SCRNSAV2.COM is a screen saver for PS/2's with VGA by Alan Ballard

INT 2F - AX = 7A00h Multiplexor - Novell NetWare - INSTALLATION CHECK

Return: AL = 00h not installed
= FFh installed
ES:DI -> FAR entry point for routines otherwise accessed
through INT 21h

INT 2F - AX = AA00h Multiplexor - VIDCLOCK.COM - INSTALLATION CHECK

Return: AL = 00h not installed
FFh installed
Note: VIDCLOCK.COM is a memory-resident clock by Thomas G. Hanlin III

INT 2F - AH = B0h Multiplexor - GRAFTABL.COM or DISPLAY.SYS

???

INT 2F - AX = B700h Multiplexor - APPEND - INSTALLATION CHECK

Return: AH <> 0 if installed

INT 2F - AX = B701h Multiplexor - APPEND - ???

???

INT 2F - AX = B702h Multiplexor - APPEND - VERSION CHECK

???
Return: ???

INT 2F - AX = B800h Multiplexor - Network - INSTALLATION CHECK

Return: AH = 0 not installed
<> 0 installed
BX = installed component flags (test in this order!)
bit 6 server
bit 2 messenger
bit 7 receiver
bit 3 redirector

INT 2F - AX = B803h Multiplexor - Network - GET CURRENT POST ADDRESS

Return: ES:BX = post address

INT 2F - AX = B804h Multiplexor - Network - SET NEW POST ADDRESS

ES:BX = new post address

INT 2F - AX = B809h Multiplexor - Network - VERSION CHECK

???
Return: ???

INT 2F - AX = F700h Multiplexor - AUTOPARK.COM - INSTALLATION CHECK

Return: AL = 00h not installed
FFh installed
Note: AUTOPARK.COM is a resident hard disk parker by Alan D. Jones

INT 2F - AX = F701h Multiplexor - AUTOPARK.COM - SET PARKING DELAY

BX:CX = 32 bit count of 55ms timer ticks

INT 30 - (NOT A VECTOR!) FAR JuMP instruction for CP/M-style calls the CALL 5 entry point does a FAR jump to here

INT 31 - overwritten by CP/M jump instruction in INT 30h

INT 32 - reserved

INT 33 - AX = 0000h MS MOUSE - RESET DRIVER AND READ STATUS

Return: AX = status
0 hardware/driver not installed
-1 hardware/driver installed
BX = number of buttons
-1 two buttons
0 other than two
3 Mouse Systems mouse

INT 33 - AX = 0001h MS MOUSE - SHOW MOUSE CURSOR

INT 33 - AX = 0002h MS MOUSE - HIDE MOUSE CURSOR

Note: multiple calls to hide the cursor will require multiple calls to function 01h to unhide it.

INT 33 - AX = 0003h MS MOUSE - RETURN POSITION AND BUTTON STATUS

Return: BX = button status
bit 0 left button pressed if 1
bit 1 right button pressed if 1
bit 2 middle button pressed if 1 (Mouse Systems mouse)

CX = column
DX = row

INT 33 - AX = 0004h MS MOUSE - POSITION MOUSE CURSOR

CX = column
DX = row

INT 33 - AX = 0005h MS MOUSE - RETURN BUTTON PRESS DATA

BX = button
0 left
1 right
2 middle (Mouse Systems mouse)
Return: AX = button states
bit 0 left button pressed if 1
bit 1 right button pressed if 1
bit 2 middle button pressed if 1 (Mouse Systems mouse)
BX = number of times specified button has been pressed since last call
CX = column at time specified button was last pressed
DX = row at time specified button was last pressed

INT 33 - AX = 0006h MS MOUSE - RETURN BUTTON RELEASE DATA

BX = button
0 left
1 right
2 middle (Mouse Systems mouse)
Return: AX = button states
bit 0 left button pressed if 1
bit 1 right button pressed if 1
bit 2 middle button pressed if 1 (Mouse Systems mouse)
BX = number of times specified button has been released since last call
CX = column at time specified button was last released
DX = row at time specified button was last released

INT 33 - AX = 0007h MS MOUSE - DEFINE HORIZONTAL CURSOR RANGE

CX = minimum column
DX = maximum column

INT 33 - AX = 0008h MS MOUSE - DEFINE VERTICAL CURSOR RANGE

CX = minimum row
DX = maximum row

INT 33 - AX = 0009h MS MOUSE - DEFINE GRAPHICS CURSOR

BX = column of cursor hot spot in bitmap (-16 to 16)
CX = row of cursor hot spot (-16 to 16)
ES:DX -> bitmap
16 words screen mask
16 words cursor mask
each word defines the sixteen pixels of a row, low bit rightmost

INT 33 - AX = 000Ah MS MOUSE - DEFINE TEXT CURSOR

BX = hardware/software text cursor
0 software
CX = screen mask
DX = cursor mask
1 hardware
CX = start scan line
DX = end scan line
Note: when the software cursor is selected, the char/attribute data at the current screen position is ANDed with the screen mask and then XORed with the cursor mask

INT 33 - AX = 000Bh MS MOUSE - READ MOTION COUNTERS

Return: CX = number of mickeys mouse moved horizontally since last call DX = number of mickeys mouse moved vertically
Notes: a mickey is the smallest increment the mouse can sense positive values indicate up/right

INT 33 - AX = 000Ch MS MOUSE - DEFINE INTERRUPT SUBROUTINE PARAMETERS

CX = call mask
bit 0 call if mouse moves
bit 1 call if left button pressed
bit 2 call if left button released
bit 3 call if right button pressed
bit 4 call if right button released
bit 5 call if middle button pressed (Mouse Systems mouse)
bit 6 call if middle button released (Mouse Systems mouse)
ES:DX = address of FAR routine
Note: when the subroutine is called, it is passed the following values:
AX = condition mask (same bit assignments as call mask)
BX = button state
CX = cursor column
DX = cursor row
DI = horizontal mickey count
SI = vertical mickey count

INT 33 - AX = 000Dh MS MOUSE - LIGHT PEN EMULATION ON

INT 33 - AX = 000Eh MS MOUSE - LIGHT PEN EMULATION OFF

INT 33 - AX = 000Fh MS MOUSE - DEFINE MICKEY/PIXEL RATIO

CX = number of mickeys per 8 pixels horizontally
DX = number of mickeys per 8 pixels vertically

INT 33 - AX = 0010h MS MOUSE - DEFINE SCREEN REGION FOR UPDATING

CX,DX = X,Y coordinates of upper left corner
SI,DI = X,Y coordinates of lower right corner
Note: mouse cursor is hidden during updating, and needs to be explicitly turned on again

INT 33 - AX = 0012h PCMOUSE - SET LARGE GRAPHICS CURSOR BLOCK

BH = cursor width in words
CH = rows in cursor
BL = horizontal hot spot (-16 to 16)
CL = vertical hot spot (-16 to 16)
ES:DX -> bit map of screen and cursor maps
Return: AX = -1 if successful

INT 33 - AX = 0013h MS MOUSE - DEFINE DOUBLE-SPEED THRESHOLD

DX = threshold speed in mickeys/second, 0 = default of 64/second
Note: if speed exceeds threshold, the cursor's on-screen motion is doubled

INT 33 - AX = 0014h MS MOUSE - EXCHANGE INTERRUPT SUBROUTINES

???

INT 33 - AX = 0015h MS MOUSE - RETURN DRIVER STORAGE REQUIREMENTS

Return: BX = size of buffer needed to store driver state

INT 33 - AX = 0016h MS MOUSE - SAVE DRIVER STATE

ES:DX -> buffer for driver state

INT 33 - AX = 0017h MS MOUSE - RESTORE DRIVER STATE

ES:DX -> buffer containing saved state

INT 33 - AX = 001DhMS MOUSE - DEFINE DISPLAY PAGE NUMBER

???

INT 33 - AX = 001Eh MS MOUSE - RETURN DISPLAY PAGE NUMBER

Return: ???

INT 33 - AX = 0042h PCMOUSE - GET MSMOUSE STORAGE REQUIREMENTS

Return: AX = FFFFh successful
BX = buffer size in bytes for functions 50h and 52h
= 0 MSMOUSE not installed
= 42h functions 42h, 50h, and 52h not supported

INT 33 - AX = 0050h PCMOUSE - SAVE MSMOUSE STATE

BX = buffer size
ES:DX -> buffer
Return: AX = FFFFh if successful

INT 33 - AX = 0052h PCMOUSE - RESTORE MSMOUSE STATE

BX = buffer size
ES:DX -> buffer
Return: AX = FFFFh if successful

INT 34 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode D8h

INT 35 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode D9h

INT 36 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DAh

INT 37 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DBh

INT 38 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DCh

INT 39 - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DDh

INT 3A - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DEh

INT 3B - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates opcode DFh

INT 3C - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates instructions with an ES segment override

INT 3D - Turbo C/Microsoft languages - Floating Point emulation

This interrupt emulates a standalone FWAIT instruction

INT 3E - Turbo C/Microsoft languages - Floating Point emulation

INT 3F - Overlay manager interrupt (Microsoft LINK.EXE)

INT 40 - Hard disk - Relocated Floppy Handler (original INT 13h)

INT 41 - FIXED DISK PARAMETERS (XT,AT,XT2,XT286,PS except ESDI disks)

WORD	cylinders
BYTE	heads
WORD	starting reduced write current cylinder (XT only, 0 for others)
WORD	starting write pre-comp cylinder
BYTE	maximum ECC burst length
BYTE	control byte
	bits 0-2: drive option (XT only, 0 for others)
	bit 3: set if more than 8 heads
	bit 4: always 0
	bit 5: set if manufacturer's defect map on max cylinder+1
	bit 6: disable ECC retries
	bit 7: disable access retries
BYTE	standard timeout (XT only, 0 for others)
BYTE	formatting timeout (XT only, 0 for others)
BYTE	timeout for checking drive (XT only, 0 for others)
WORD	landing zone (AT/PS2)
BYTE	sectors/track (AT/PS2)
BYTE	0

INT 42 - EGA/VGA/PS - Relocated (by EGA) Video Handler (original INT 10h)

INT 42 - Z100 - ???

INT 43 - EGA/VGA/PS - User font table

INT 44 - EGA/VGA/CONV/PS - EGA/PCjr fonts, characters 00h to 7Fh

INT 44 - Novell NetWare - HIGH-LEVEL LANGUAGE API

INT 44 - Z100 - ???

INT 45 - Z100 - ???

INT 46 - Secondary Fixed Disk Params (see INT 41h) (AT,XT286,PS except ESDI)

INT 46 - Z100 - ???

INT 47 - reserved

INT 48 - PCjr - Cordless Keyboard Translation

INT 49 - PCjr - Non-keyboard Scan Code Translation Table

INT 4A - AT/CONV/PS - User Alarm

Invoked by BIOS when real-time clock alarm occurs

INT 4B - reserved

INT 4C - reserved

INT 4D - reserved

INT 4E - reserved

INT 4F - reserved

INT 50 to 57 - IRQ0-IRQ7 relocated by DESQview

INT 50 to 57 - IRQ0-IRQ7 relocated by IBM 3278 emulation control program

INT 58 - reserved

INT 59 - GSS Computer Graphics Interface (GSS*CGI)

DS:DX = Pointer to block of 5 array pointers
Return: CF set on error
AX = error code
CF clear if successful
AX = return code
Note: INT 59 is the means by which GSS*CGI language bindings communicate with GSS*CGI device drivers and the GSS*CGI device driver controller. Also used by the IBM Graphic Development Toolkit

INT 5A - Cluster adapter BIOS entry address

???

INT 5B - Used by cluster adapter

INT 5C - NETBIOS INTERFACE

ES:BX -> Network Control Block
Subfunction in first NCB field (or with 80h for non-waiting call)
10h start session with NCB_NAME name (call)
11h listen for call
12h end session with NCB_NAME name (hangup)
14h send data via NCB_LSN
15h receive data from a session
16h receive data from any session
17h send multiple data buffers
20h send unACKed message (datagram)
21h receive datagram
22h send broadcast datagram
23h receive broadcast datagram
30h add name to name table
31h delete name from name table
32h reset adapter card and tables
33h get adapter status
34h status of all sessions for name
35h cancel
36h add group name to name table
70h unlink from IBM remote program (no F0h function)
71h send data without ACK
72h send multiple buffers without ACK
78h find name
79h token-ring protocol trace
Return: AL = status
00h successful
01h bad buffer size
03h invalid NETBIOS command
05h timeout
06h receive buffer too small
08h bad session number
09h LAN card out of memory
0Ah session closed
0Bh command has been cancelled
0Dh name already exists
0Eh local name table full
0Fh name still in use, can't delete
11h local session table full
12h remote PC not listening
13h bad NCB_NUM field
14h no answer to CALL or no such remote
15h name not in local name table
16h duplicate name
17h bad delete
18h abnormal end
19h name error, multiple identical names in use
1Ah bad packet
21h network card busy

22h too many commands queued
23h bad LAN card number
24h command finished while cancelling
26h command can't be cancelled
FFh NETBIOS busy

Structure of Network Control Block:
BYTE ncb_command
BYTE ncb_retcode
BYTE ncb_lsn
BYTE ncb_num
DWORD -> ncb_buffer
WORD ncb_length
16 BYTEs ncb_callname
16 BYTEs ncb_name
BYTE ncb_rto
BYTE ncb_sto
DWORD -> ncb_post /* int (far *ncb_post)(); */
BYTE ncb_lana_num
BYTE ncb_cmd_cplt
14 BYTEs ncb_reserve

Structure name:
16 BYTEs nm_name
BYTE nm_num
BYTE nm_status

Structure astatus:
6 BYTEs as_id
BYTE as_jumpers
BYTE as_post
BYTE as_major
BYTE as_minor
WORD as_interval
WORD as_crcerr
WORD as_algerr
WORD as_colerr
WORD as_abterr
DWORD as_tcount
DWORD as_rcount
WORD as_retran
WORD as_xresrc
8 BYTEs as_res0
WORD as_ncbfree
WORD as_ncbmax
WORD as_ncbx
4 BYTEs as_res1
WORD as_sespend
WORD as_msp
WORD as_sesmax
WORD as_bufsize
WORD as_names
16 name structures as_name

Note: Sytek PCnet card uses DMA 3.

INT 5C - TOPS INTERFACE

ES:BX -> Network Control Block
Note: TOPS card uses DMA 1, 3 or none.

INT 5D - reserved

INT 5E - reserved

INT 5F - reserved

INT 60 - reserved for user interrupt

INT 60 - FTP Driver - PC/TCP Packet Driver Specification

The handler for the interrupt will start with a 3-byte jump instruction, followed by the ASCIZ string "PKT DRVR". To find the interrupt being used by the driver, an application should scan through interrupt vectors 60h to 80h until it finds one with the "PKT DRVR" string.

Network Interface classes/types:
Class 01h Ethernet/IEEE 802.3
01h 3COM 3C500/3C501
02h 3COM 3C505
03h MICOM-Interlan NI5010
04h BICC Data Networks 4110
05h BICC Data Networks 4117
06h MICOM-Interlan NP600
08h Ungermann-Bass PC-NIC
09h Univation NC-516

0Ah TRW PC-2000
0Bh MICOM-Interlan NI5210
0Ch 3COM 3C503
0Dh 3COM 3C523
0Eh Western Digital WD8003
0Fh Spider Systems S4
Class 02h ProNET-10
01h Proteon p1300
Class 03h IEEE 802.5/ProNet-4
01h IBM Token-Ring Adapter
02h Proteon p1340
03h Proteon p1344
Class 04h Omninet
Class 05h Appletalk
Class 06h Serial Line
Class 07h StarLAN
Class 08h ARCnet
01h Datapoint RIM

INT 60 - FTP Driver - DRIVER INFO

AX = 01FFh
BX = handler returned by function 02h
Return: CF set on error
DH = error code
01h invalid handle number
02h no interfaces of the specified class found
03h no interfaces of the specified type found
04h no interfaces of the specified number found
05h bad packet type
06h interface does not support multicast messages
07h this packet driver cannot terminate
08h invalid receiver mode
09h insufficient space
0Ah type accessed but never released
0Bh bad command
0Ch packet could not be sent
CF clear if successful
BX = version
CH = class
DX = type
CL = number
DS:SI -> name
AL = driver type
01h basic
02h extended
FFh not installed

INT 60 - FTP Driver - ACCESS TYPE

AH = 02h
AL = interface class
BX = interface type
DL = interface number
DS:SI -> type
CX = length of type
ES:DI -> receiver
Return: CF set on error
DH = error code (see above)
CF clear if successful
AX = handle

Receiver called with
AX = subfunction
00h application to return pointer to buffer in ES:DI
ES:DI = 0:0 means throw away packet
01h copy to DS:SI buffer completed
BX = handle
CX = buffer length
when a packet is received

INT 60 - AH = 03h FTP Driver - RELEASE TYPE

BX = handle
Return: CF set on error
DH = error code (see above)
CF clear if successful

INT 60 - AH = 04h FTP Driver - SEND PACKET

DS:SI -> buffer
CX = length
Return: CF set on error
DH = error code (see above)

INT 60 - AH = 05h FTP Driver - TERMINATE DRIVER FOR HANDLE

BX = handle

Return: CF set on error
DH = error code (see above)

INT 60 - AH = 60h FTP Driver - GET ADDRESS

BX = handle
ES:DI -> buffer
CX = length
Return: CF set on error
DH = error code (see above)
CF clear if successful
CX = length
Note: copies the local net address associated with the handle into the buffer

INT 60 - AH = 07h FTP Driver - RESET INTERFACE

BX = handle
Return: CF set on error
DH = error code (see above)

INT 60 - AH = 11h 10-NET - LOCK AND WAIT

AL = drive number or 0
DX = number of seconds to wait
ES:SI = Ethernet address or 0
DS:BX -> 31-byte ASCIZ semaphore name
Return: AL = status
0 successful
1 timeout
2 server not responding
3 invalid semaphore name
4 semaphore list is full
5 invalid drive ID
6 invalid Ethernet address
7 not logged in
8 write to network failed
9 semaphore already logged for this CPU

INT 60 - AH = 12h 10-NET - LOCK

AL = drive number or 0
ES:SI = Ethernet address or 0
DS:BX -> 31-byte ASCIZ semaphore name
Return: AL = status (see function 11h)
1 semaphore currently logged
Note: unlike function 11h, this function returns immediately

INT 60 - AH = 13h 10-NET - UNLOCK

AL = drive number or 0
ES:SI = Ethernet address or 0
DS:BX -> 31-byte ASCIZ semaphore name
Return: AL = status (see function 11h)
1 semaphore not logged

INT 60 - AH = 20h FTP Driver - SET RECEIVE MODE

BX = handle
CX = mode
01h turn off receiver
02h receive only packets sent to this interface
03h mode 2 plus broadcast packets
04h mode 3 plus limited multicast packets
05h mode 3 plus all multicast packets
06h all packets
Return: CF set on error
DH = error code

INT 60 - AH = 21h FTP Driver - GET RECEIVE MODE

BX = handle
Return: CF set on error
DH = error code (see function 01h above)
CF clear if successful
AX = mode

INT 60 - AH = 24h FTP Driver - GET STATISTICS

BX = handle
Return: CF set on error
DH = error code
CF clear if successful
DS:SI -> statistics
DWORD packets in
DWORD packets out
DWORD bytes in
DWORD bytes out
DWORD errors in
DWORD errors out
DWORD packets dropped

INT 61 - reserved for user interrupt

INT 62 - reserved for user interrupt

INT 63 - reserved for user interrupt

INT 64 - reserved for user interrupt

INT 65 - reserved for user interrupt

INT 66 - reserved for user interrupt

INT 67 - AH = 40h LIM EMS - GET MANAGER STATUS

Return: AH = status
- 00h successful
- 80h internal error
- 81h hardware malfunction
- 82h Memory Manager busy
- 83h Invalid handle
- 84h Undefined function requested by application
- 85h No more handles available
- 86h Error in save or restore of mapping context
- 87h Allocation request specified more logical pages than physically available in system; no pages allocated.
- 88h Allocation request specified more pages than currently available in system (request does not exceed physical pages that exist but some are already allocated to other handles. No pages allocated.

Note: this call can be used only after establishing that the EMS driver is in fact present

INT 67 - AH = 41h LIM EMS - GET PAGE FRAME SEGMENT

Return: AH = 00h function successful
BX = segment of page frame
AH = error code (see AH=40h above)

INT 67 - AH = 42h LIM EMS - GET NUMBER OF PAGES

Return: AH = 00h function successful
BX = number of unallocated pages
DX = total number of pages
AH = error code (see AH=40h above)

INT 67 - AH = 43h LIM EMS - GET HANDLE AND ALLOCATE MEMORY

BX = number of logical pages to allocate
Return: AH = status
- 00h function successful
  - DX = handle
- 80h internal error
- 81h hardware malfunction
- 84h undefined function requested
- 85h no more handles available
- 87h more pages requested than physically exist
- 88h more pages requested than currently available
- 89h zero pages requested

INT 67 - AH = 44h LIM EMS - MAP MEMORY

AL = physical page number (0-3)
BX = logical page number
DX = handle
Return: AH = status
- 00h function successful
- 80h internal error
- 81h hardware malfunction
- 83h invalid handle
- 84h undefined function requested
- 8Ah invalid logical page number
- 8Bh illegal physical-page number

INT 67 - AH = 45h LIM EMS - RELEASE HANDLE AND MEMORY

DX = EMM handle
Return: AH = status
- 00h successful
- 80h internal error
- 81h hardware malfunction
- 83h invalid handle
- 84h undefined function requested
- 86h error in save or restore of mapping context

INT 67 - AH = 46h LIM EMS - GET EMM VERSION

Return: AH = status
- 00h successful
  - AL = EMM version number
- 80h internal error

81h hardware malfunction
84h undefined function requested

INT 67 - AH = 47h LIM EMS - SAVE MAPPING CONTEXT

DX = handle
Return: AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested
8Ch page-mapping hardware state save area is full
8Dh save of mapping context failed

INT 67 - AH = 48h LIM EMS - RESTORE MAPPING CONTEXT

DX = handle
Return: AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested
8Eh restore of mapping context failed

INT 67 - AH = 49h LIM EMS - reserved - GET I/O PORT ADDRESSES

Note: defined in EMS 3.0, but undocumented in EMS 3.2

INT 67 - AH = 4Ah LIM EMS - reserved - GET TRANSLATION ARRAY

Note: defined in EMS 3.0, but undocumented in EMS 3.2

INT 67 - AH = 4Bh LIM EMS - GET NUMBER OF EMM HANDLES

Return: AH = status
00h successful
BX = number of EMM handles
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested

INT 67 - AH = 4Ch LIM EMS - GET PAGES OWNED BY HANDLE

DX = EMM handle
Return: AH = status
00h successful
BX = number of logical pages
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested

INT 67 - AH = 4Dh LIM EMS - GET PAGES FOR ALL HANDLES

ES:DI -> array to receive information
Return: AH = status
00h successful
BX = number of active EMM handles
array filled with 2-word entries, consisting of a handle
and the number of pages allocated to that handle
80h internal error
81h hardware malfunction
84h undefined function requested

INT 67 - AH = 4Eh LIM EMS - GET OR SET PAGE MAP

AL = 00h if getting mapping registers
01h if setting mapping registers
02h if getting and setting mapping registers at once
03h if getting size of page-mapping array
DS:SI -> array holding information (AL=01/02)
ES:DI -> array to receive information (AL=00/02)
Return: AH = status
00h successful
AL = bytes in page-mapping array (subfunction 03h only)
array pointed to by ES:DI receives mapping info (AL=00/02)
80h internal error
81h hardware malfunction
84h undefined function requested
8Fh undefined subfunction parameter
A3h contents of source array corrupted (EMS 4.0?)
Note: this function was designed to be used by multitasking operating systems and should not ordinarily be used by application software.

INT 67 - AH = 4Fh LIM EMS 4.0 - GET/SET PARTIAL PAGE MAP

AL = subfunction
00h get partial page map
DS:SI -> structure containing list of segments whose mapping

contexts are to be saved
ES:DI -> array to receive page map
01h set partial page map
DS:SI -> structure containing saved partial page map
02h get size of partial page map
BX = number of mappable segments in the partial map to be saved
Return: AH = status
00h successful
80h internal error
81h hardware malfunction
84h undefined function requested
8Bh one of specified segments is not mappable
8Fh undefined subfunction parameter
A3h contents of partial page map corrupted or count of mappable
segments exceeds total number of mappable segments in system
AL = size of partial page map for subfunction 02h

INT 67 - AH = 50h LIM EMS 4.0 - MAP/UNMAP MULTIPLE HANDLE PAGES

AL = subfunction
00h
01h
DX = handle
CX = number of entries in array
DS:SI -> mapping array
Return: AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested
8Ah one or more logical pages are invalid
8Bh one or more physical pages are invalid
8Fh invalid subfunction

INT 67 - AH = 51h LIM EMS 4.0 - REALLOCATE PAGES

DX = handle
BX = number of pages to be allocated to handle
Return: BX = actual number of pages allocated to handle
AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested
87h more pages requested than present in system
88h more pages requested than currently available

INT 67 - AH = 52h LIM EMS 4.0 - GET/SET HANDLE ATTRIBUTES

AL = subfunction
00h get handle attributes
01h set handle attributes
BL = new attribute (see returned AL)
02h get attribute capability
DX = handle
Return: AL = attribute (for subfunction 00h)
00h handle is volatile
01h handle is nonvolatile
AL = attribute capability (for subfunction 02h)
00h only volatile handles supported
01h both volatile and non-volatile supported
AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested
8Fh undefined subfunction
90h undefined attribute type
91h feature not supported

INT 67 - AH = 53h LIM EMS 4.0 - GET/SET HANDLE NAME

AL = subfunction
00h get handle name
ES:DI -> 8-byte handle name array
01h set handle name
DS:SI -> 8-byte handle name
DX = handle
Return: AH = status
00h successful
80h internal error
81h hardware malfunction
83h invalid handle
84h undefined function requested

8Fh undefined subfunction
A1h duplicate handle name

INT 67 - AH = 54h LIM EMS 4.0 - GET HANDLE DIRECTORY

AL = subfunction
00h get handle directory
ES:DI -> buffer for handle directory
Series of 10 byte entries, one per handle 2bytes handle no, 8 bytes handle name
01h search for named handle
DS:SI -> 8-byte name
02h get total number of handles
Return: AL = number of entries in handle directory (subfunction 00h)
DX = value of named handle (subfunction 01h)
BX = total number of handles (subfunction 02h)
AH = status
00h successful
80h internal error
81h hardware malfunction
84h undefined function requested
8Fh undefined subfunction
A0h no such handle name
A1h a handle found had no name

INT 67 - AH = 55h LIM EMS 4.0 - ALTER PAGE MAP AND JUMP

AL = subfunction
00h physical page numbers provided by caller
01h segment addresses provided by caller
DX = handle
DS:SI -> structure containing map and jump address
see function 56 offsets 00h to 08h
Return: (at target address unless error)
AH = status
00h successful
80h internal error
81h hardware failure
83h invalid handle
84h undefined function requested
8Ah invalid logical page number encountered
8Bh invalid physical page number encountered
8Fh invalid subfunction

INT 67 - AH = 56h LIM EMS 4.0 - ALTER PAGE MAP AND CALL

AL = subfunction
00h physical page numbers provided by caller
DX = handle
DS:SI -> structure containing page map and call address

Offset	Length	Description
00h	4	far pointer to call target
04h	1	number of pages to map before call
05h	4	far pointer to list of pages to map before call
09h	1	number of pages to map before return
0Ah	4	far pointer to list of pages to map before return
0Eh	8	reserved (0)

list of pages consists of series of DWORD (32bit) entries one per page
first word is logical page no, second physical page no or segment,
depending on subfunction in AL
01h segment addresses provided by caller
DX = handle
DS:SI -> structure containing page map and call address
02h get page map stack space required
Return: (if successful, the target address is called. Use a RETF to return and restore mapping context)
BX = stack space required (subfunction 02h)
AH = status
00h successful
80h internal error
81h hardware failure
83h invalid handle
84h undefined function requested
8Ah invalid logical page number encountered
8Bh invalid physical page number encountered
8Fh undefined subfunction

INT 67 - AH = 57h LIM EMS 4.0 - MOVE/EXCHANGE MEMORY REGION

AL = subfunction
00h move memory region
01h exchange memory region
DS:SI -> structure describing source and destination

Offset	Length	Description
00h	4	region length in bytes
04h	1	source memory type (0=conventional 1=expanded)
05h	2	source memory handle
07h	2	source memory offset
09h	2	source memory segment or physical page number

0Bh 1 destination memory type (0=conventional 1=expanded)
0Ch 2 destination memory handle
0Eh 2 destination memory offset
10h 2 destination memory segment or physical page number

Return: AH = status
00h successful
80h internal error
81h hardware failure
83h invalid handle
84h undefined function requested
8Ah invalid logical page number encountered
8Fh undefined subfunction
92h successful, but a portion of the source region has been overwritten
93h length of source or destination region exceeds length of region allocated to either source or destination handle
94h conventional and expanded memory regions overlap
95h offset within logical page exceeds size of logical page
96h region length exceeds 1M
97h source and destination EMS regions have same handle and overlap
98h memory source or destination type undefined
A2h attempted to wrap around 1M conventional address space

INT 67 - AH = 58h LIM EMS 4.0 - GET MAPPABLE PHYSICAL ADDRESS ARRAY

AL = subfunction
00h get mappable physical address array
ES:DI -> buffer to be filled with array
01h get number of entries in m.p.a. array

Return: CX = number of entries in array
AH = status
00h successful
80h internal error
81h hardware failure
84h undefined function requested
8Fh undefined subfunction

INT 67 - AH = 59h LIM EMS 4.0 - GET EXPANDED MEMORY HARDWARE INFORMATION

AL = subfunction
00h get hardware configuration array
ES:DI -> buffer to be filled with array
01h get unallocated raw page count

Return: BX = unallocated raw pages (subfunction 01h)
DX = total raw pages (subfunction 01h)
AH = status
00h successful
80h internal error
81h hardware failure
84h undefined function requested
8Fh undefined subfunction
A4h access denied by operating system

Note: subfunction 00h is for use by operating systems only, and can be enabled or disabled at any time by the operating system

INT 67 - AH = 5Ah LIM EMS 4.0 - ALLOCATE STANDARD/RAW PAGES

AL = subfunction
00h allocate standard pages
01h allocate raw pages
BX = number of pages to allocate

Return: DX = handle
AH = status
00h successful
80h internal error
81h hardware failure
84h undefined function requested
85h no more handles available
87h insufficient memory pages in system
88h insufficient memory pages available
8Fh undefined subfunction

INT 67 - AH = 5Bh LIM EMS 4.0 - ALTERNATE MAP REGISTER SET

AL = subfunction
00h get alternate map register set
01h set alternate map register set
BL = new alternate map register set number
ES:DI -> map register context save area if BL=0
02h get alternate map save array size
03h allocate alternate map register set
04h deallocate alternate map register set
BL = number of alternate map register set

Return: BL = current active alternate map register set number if nonzero (AL=0)
ES:DI -> map register context save area if BL=0 (AL=0)
DX = array size in bytes (subfunction 02h)
BL = number of alternate map register set; zero if not supported (AL=3)
AH = status

00h successful
80h internal error
81h hardware malfunction
84h undefined function requested
8Fh undefined subfunction
9Ah specified alternate map register set not supported
9Bh all alternate map register sets currently allocated
9Ch alternate map register sets not supported
9Dh undefined or unallocated alternate map register set
A3h source array corrupted
A4h operating system denied access

Note: this function is for use by operating systems only, and can be enabled or disabled at any time by the operating system

INT 67 - AH = 5Bh LIM EMS 4.0 - ALTERNATE MAP REGISTER SET - DMA REGISTERS

AL = subfunction
05h allocate DMA register set
06h enable DMA on alternate map register set
BL = DMA register set number
DL = DMA channel number
07h disable DMA on alternate map register set
BL = DMA register set number
08h deallocate DMA register set
BL = DMA register set number

Return: BL = DMA register set number; zero if not supported (subfunction 05h)
AH = status
00h successful
80h internal error
81h hardware malfunction
84h undefined function requested
8Fh undefined subfunction
9Ah specified DMA register set not supported
9Bh all DMA register sets currently allocated
9Ch alternate DMA sets not supported
9Dh undefined or unallocated DMA register set
9Eh dedicated DMA channels not supported
9Fh specified dedicated DMA channel not supported
A3h source array corrupted
A4h operating system denied access

Note: this function is for use by operating systems only, and can be enabled or disabled at any time by the operating system

INT 67 - AH = 5Ch LIM EMS 4.0 - PREPARE EXPANDED MEMORY HARDWARE FOR WARM BOOT

Return: AH = status
00h successful
80h internal error
81h hardware malfunction
84h undefined function requested

INT 67 - AH = 5Dh LIM EMS 4.0 - ENABLE/DISABLE OS FUNCTION SET FUNCTIONS

AL = subfunction
00h enable OS Function Set
01h disable OS Function Set
02h return access key (resets memory manager, returns access key at next invocation)
BX,CX = access key returned by first invocation

Return: BX,CX = access key, returned only on first invocation of function
AH = status
00h successful
80h internal error
81h hardware malfunction
84h undefined function requested
8Fh undefined subfunction
A4h operating system denied access

INT 67 - AH = 60h EEMS - GET PHYSICAL WINDOW ARRAY

ES:DI -> buffer

Return: AH = status
AL = number of entries
buffer at ES:DI filled

INT 67 - AH = 61h EEMS - GENERIC ACCELERATOR CARD SUPPORT

???

Return: ???

Note: can be used by accelerator card manufacturer to flush RAM cache, ensuring that the cache accurately reflects what the processor would see without the cache.

INT 67 - AH = 68h EEMS - GET ADDRESSES OF ALL PAGE FRAMES IN SYSTEM

ES:DI -> buffer

Return: AH = status
AL = number of entries
buffer at ES:DI filled

Note: equivalent to LIM 4.0 function 58h

INT 67 - AH = 69h EEMS - MAP PAGE INTO FRAME

AL = frame number
BX = page number
DX = handle
Return: AH = status
Note: similar to EMS function 44h

INT 67 - AH = 6Ah EEMS - PAGE MAPPING

AL = subfunction
00h save partial page map
CH = first page frame
CL = number of frames
ES:DI -> buffer which is to be filled
01h restore partial page map
CH = first page frame
CL = number of frames
DI:SI -> previously saved page map
02h save and restore partial page map
CH = first page frame
CL = number of frames
ES:DI = buffer for current page map
DI:SI = new page map
03h get size of save array
CH = first page frame
CL = number of frames
Return: AL = size of array in bytes
04h switch to standard map register setting
05h switch to alternate map register setting
06h deallocate pages mapped to frames in conventional memory
CH = first page frame
CL = number of frames
Return: AH = status
Note: similar to EMS function 4Eh, except that a subrange of pages can be specified

INT 68 - AH = 01h APPC/PC

DS:DX -> control block
12 BYTEs reserved
WORD verb (action)
6 BYTEs 0
DWORD (high byte first) return code

0000h	successful
0001h	BAD_TP_ID
0002h	BAD_CONV_ID
0003h	bad logical unit ID
0008h	no physical unit attached
0110h	bad state
01B1h	BAD_PART_LUNAME
01B2h	bad mode name
0201h	physical unit already active
0211h	logical unit already active
0212h	BAD_PART_SESS
0213h	BAD_RU_SIZES
0214h	BAD_MODE_SESS
0216h	BAD_PACING_CNT
0219h	EXTREME_RUS
021Ah	SNASVCMG_1
0223h	SSCP_CONNECTED_LU
0230h	invalid change
0243h	too many TPs
0272h	adapter close failure
0281h	GET_ALLOC_BAD_TYPE
0282h	unsuccessful
0283h	DLC failure
0284h	unrecognized DLC
0286h	duplicate DLC
0301h	SSCP_PU_SESSION_NOT_ACTIVE
0302h	data exceeds RU size
0401h	invalid direction
0402h	invalid type
0403h	segment overlap
0404h	invalid first character
0405h	table error
0406h	conversion error
F0010000h	APPC disabled
F0020000h	APPC busy
F0030000h	APPC abended
F0040000h	incomplete

if verb = 1B00h (DISPLAY), control block continues
WORD 0
8 BYTEs (high byte first) logical unit ID
8 BYTEs (high byte first) partner logical unit name

8 BYTEs (high byte first) mode name
BYTE logical unit session limit
BYTE partner logical unit session limit
BYTE mode maximum negotiable session limit
BYTE current session limit
BYTE minimum negotiated winner limit
BYTE maximum negotiated loser limit
BYTE active session count
BYTE active CONWINNER session count
BYTE active CONLOSER session count
BYTE session termination count
BYTE bit 7: SESSION_TERMINATION_TARGET_DRAIN
bit 6: SESSION_TERMINATION_SOURCE_DRAIN
if verb=2000h (Attach Physical Unit), control block continues
WORD 0
BYTE version
BYTE release
8 BYTEs (high byte first) net name
8 BYTEs (high byte first) physical unit name
8 BYTEs 0
DWORD pointer to SYSTEM_LOG_EXIT routine, FFFFFFFFh = don't log errors
DWORD 0
BYTE 0 RETURN_CONTROL: COMPLETE
1 RETURN_CONTROL: INCOMPLETE
if verb=2100h (Attach Logical Unit), control block continues
WORD 70 offset to partner logical unit record
8 BYTEs (high byte first) logical unit name
8 BYTEs (high byte first) logical unit ID
BYTE logical unit local address
BYTE logical unit session limit
DWORD pointer to CREATE_TP_EXIT routine,
FFFFFFFFh = reject incoming ALLOCATEs
00000000h = queue ALLOCATEs
DWORD 0
DWORD pointer to SYSTEM_LOG_EXIT routine, FFFFFFFFh = don't log errors
DWORD 0
BYTE maximum TPs
BYTE queue depth
DWORD pointer to LU_LU_PASSWORD_EXIT routine, FFFFFFFFh = no pswd exit
DWORD 0
WORD total length of partner records
for each partner logical unit:
WORD length of this partner logical unit record
WORD 42 offset to mode records
8 BYTEs (high byte first) partner logical unit name
BYTE partner logical unit security capabilities
bit 7: already verified
bit 6: conversation level security
bit 5: session level security
BYTE partner logical unit session limit
WORD partner logical unit maximum MC_SEND_LL
8 BYTEs (high byte first) partner logical unit DLC name
BYTE partner logical unit adapter number
17 BYTEs (counted string) partner logical unit adapter address
WORD total length of mode records
for each mode:
WORD 16 length of this mode record
8 BYTEs (high byte first) mode name
WORD RU_SIZE high bound
WORD RU_SIZE low bound
BYTE mode maximum negotiable session limit
BYTE pacing size for receive
if verb=2200h (Detach Logical Unit), control block continues:
8 BYTEs (high byte first) logical unit ID
BYTE 0
if verb=2700h (Detach Physical Unit), control block continues:
BYTE 0 type: hard
1 type: soft
if verb=2B00h (Activate DLC), control block continues:
8 BYTEs (high byte first) DLC name
BYTE adapter number

Routines defined by LU_LU_PASSWORD_EXIT, CREATE_TP_EXIT, and SYSTEM_LOG_EXIT pointers are called by pushing the DWORD pointer to the verb on the stack and then performing a FAR call.

ACCESS_LU_LU_PW verb:
12 BYTEs reserved
WORD 1900h
8 BYTEs (high byte first) logical unit ID
8 BYTEs (high byte first) logical unit name
8 BYTEs (high byte first) partner logical unit name
17 BYTEs (counted string) partner fully qualified logical unit name

BYTE password available (0=no, 1=yes)
8 BYTEs password
CREATE_TP verb:
12 BYTEs reserved
WORD 2300h
6 BYTEs 0
DWORD (high byte first) sense code
00000000h Ok
080F6051h SECURITY_NOT_VALID
084B6031h TP_NOT_AVAIL_RETRY
084C0000h TP_NOT_AVAIL_NO_RETRY
10086021h TP_NAME_NOT_RECOGNIZED
10086034h CONVERSATION_TYPE_MISMATCH
10086041h SYNC_LEVEL_NOT_SUPPORTED
8 BYTEs (high byte first) TP ID
8 BYTEs (high byte first) logical unit ID
DWORD (high byte first) conversation ID
BYTE 0 basic conversation, 1 mapped conversation
BYTE 0 no sync level, 1 confirm
BYTE reserved
65 BYTEs (counted string) transaction program name
6 BYTEs 0
WORD length of ERROR_LOG_DATA to return
DWORD pointer to ERROR_LOG_DATA buffer
8 BYTEs (high byte first) partner logical unit name
18 BYTEs (counted string) partner fully qualified logical unit name
8 BYTEs (high byte first) mode name
12 BYTEs 0
11 BYTEs (counted string) password
11 BYTEs (counted string) user ID
BYTE 0 verification should be performed
1 already verified
SYSLOG verb:
12 BYTEs reserved
WORD 2600h
10 BYTEs 0
WORD (high byte first) type
DWORD (high byte first) subtype
DWORD pointer to ADDITIONAL_INFO
DWORD (high byte first) conversation ID
8 BYTEs (high byte first) TP ID
8 BYTEs (high byte first) physical unit or logical unit name
WORD length of data
DWORD pointer to data
BYTE 0

INT 68 - APPC/PC

AH = 02h
DS:DX -> control block
12 BYTEs reserved
WORD verb (action)
BYTE 1 if MC_ (mapped conversation) form of verb
0 if basic verb
5 BYTEs 0
WORD (high byte first) primary return code
0000h successful
0001h parameter check
0002h state check
0003h allocation error
0005h deallocate abended
0006h deallocate abended program
0007h deallocate abended SVC
0008h deallocate abended timer
0009h deallocate normal return
000Ah data posting blocked
000Bh posting not active
000Ch PROG_ERROR_NO_TRUNC
000Dh PROG_ERROR_TRUNC
000Eh PROG_ERROR_PURGING
000Fh CONV_FAILURE_RETRY
0010h CONV_FAILURE_NO_RETRY
0011h SVC_ERROR_NO_TRUNC
0012h SVC_ERROR_TRUNC
0013h SVC_ERROR_PURGING
0014h unsuccessful
0018h CNOS partner logical unit reject
0019h conversation type mixed
F001h APPC disabled
F002h APPC busy
F003h APPC abended
F004h incomplete
DWORD (high byte first) error code

0001h bad TP ID
0002h bad conversation ID
0004h allocation error, no retry
0005h allocation error, retry
0006h data area crosses segment boundary
0010h bad TPN length
0011h bad CONV length
0012h bad SYNC level
0013h bad security selection
0014h bad return control
0015h SEC_TOKENS too big
0016h PIP_LEN incorrect
0017h no use of SNASVCMG
0018h unknown partner mode
0031h confirm: SYNC_NONE
0032h confirm: bad state
0033h confirm: NOT_LL_BDY
0041h confirmed: bad state
0051h deallocate: bad type
0052h deallocate: flush bad state
0053h deallocate: confirm bad state
0055h deallocate: NOT_LL_BDY
0057h deallocate: log LL_WRONG
0061h flush: not send state
0091h post on receipt: invalid length
0092h post on receipt: not in receive state
0093h post on receipt: bad fill
00A1h prepare to receive:invalid type
00A2h prepare to receive: unfinished LL
00A3h prepare to receive: not in send state
00B1h receive and wait: bad state
00B2h receive and wait: NOT_LL_BDY
00B5h receive and wait: bad fill
00C1h receive immediate: not in receive state
00C4h receive immediate: bad fill
00E1h request to send: not in receive state
00F1h send data: bad LL
00F2h send data: not in send state
0102h send error: log LL wrong
0103h send error: bad type
0121h test: invalid type
0122h test: not in receive state
8 BYTEs (high byte first) TP_ID
DWORD (high byte first) conversation ID
if verb=0100h (Allocate or MC_Allocate), control block continues:
BYTE (MC_Allocate only) 0 basic conversation
1 mapped conversation
BYTE 0 SYNC_LEVEL = none
1 SYNC_LEVEL = confirm
WORD 0
BYTE 0 RETURN_CONTROL: when session allocated
1 RETURN_CONTROL: immediate
2 RETURN_CONTROL: when session free
8 BYTEs 0
8 BYTEs (high byte first) partner logical unit name
8 BYTEs (high byte first) mode name
65 BYTEs (counted string) TP name
BYTE 0 security: none
1 security: same
2 security: pgm
11 BYTEs 0
11 BYTEs (counted string) password
11 BYTEs (counted string) user ID
WORD PIP_DATA length
DWORD pointer to PIP_DATA
if verb=0300h (Confirm or MC_Confirm), then control block continues:
BYTE request to send received (0=no, 1=yes)
if verb=0400h (Confirmed or MC_Confirmed), no additional fields
if verb=0500h (Deallocate or MC_Deallocate), then control block continues:
BYTE 0
BYTE type 0 SYNC_LEVEL
1 FLUSH
2 ABEND_PROC
3 ABEND_SVC
4 ABEND_TIMER
5 ABEND
WORD (MC_Deallocate only) length of error log data
DWORD (MC_Deallocate only) pointer to error log data
if verb=0600h (Flush or MC_Flush), no additional fields
if verb=0700h (Get_Attributes or MC_Get_Attributes), control block continues:
8 BYTEs (high byte first) logical unit ID
BYTE 0

BYTE SYNC_LEVEL (0=none, 1=confirm)
8 BYTEs (high byte first) mode name
8 BYTEs (high byte first) own net name
8 BYTEs (high byte first) own logical unit name
8 BYTEs (high byte first) partner logical unit name
18 BYTEs (counted string) partner's fully qualified logical unit name
BYTE 0
11 BYTEs (counted string) user ID
if verb=0800h (Get_Type), then control block continues:
BYTE type (0=basic conversation, 1=mapped conversation)
if verb=0900h (Post_on_Receipt), then control block continues:
WORD maximum length
BYTE fill (0=buffer, 1=LL)
if verb=0A00h (Prepare_to_Receive or MC_Prepare_to_Receive):
BYTE type (0=SYNC_LEVEL, 1=FLUSH)
BYTE locks (0=short, 1=long)
if verb=0B00h (Receive_and_Wait or MC_Receive_and_Wait), control block cont:
BYTE what received
0 data
1 data complete
2 data incomplete
3 confirm
4 confirm send
5 confirm deallocate
6 send
BYTE (MC_Receive_and_Wait only) fill (0=buffer, 1=LL)
BYTE Request_to_Send_Received (0=no, 1=yes)
WORD maximum length
WORD data length
DWORD pointer to data
if verb=0C00h (Receive_Immediate or MC_Receive_Immediate), control block:
BYTE what received
0 data
1 data complete
2 data incomplete
3 confirm
4 confirm send
5 confirm deallocate
6 send
BYTE (MC_Receive_Immediate only) fill (0=buffer, 1=LL)
BYTE Request_to_Send_Received (0=no, 1=yes)
WORD maximum length
WORD data length
DWORD pointer to data
if verb=0E00h (Request_to_Send or MC_Request_to_Send), no additional fields
if verb=0F00h (Send_Data or MC_Send_Data), then control block continues:
BYTE request to send received (0=no, 1=yes)
BYTE 0
WORD data length
DWORD pointer to data
if verb=1000h (Send_Error or MC_Send_Error)
BYTE request to send received (0=no, 1=yes)
BYTE type (0=program, 1=SVC)
DWORD 0
WORD (MC_Send_Error only) LOG_DATA length
DWORD (MC_Send_Error only) pointer to LOG_DATA
if verb=1200h (Test or MC_Test), then control block continues:
BYTE (MC_Test only) test (0=posted, 1=request_to_send received)
Note: error code has different interpretations for:
0 posted data
1 posted not data (primary return code = 0)
1 bad TP_ID (primary return code = 1)
if verb=1300h (Wait), then control block continues:
BYTE number of conversations to wait on
Note: error codes have interpretations as for 1200h (Test) above

INT 68 - AH = 03h APPC/PC

DS:DX -> control block
12 BYTEs reserved
WORD verb (action)
6 BYTEs 0
DWORD (high byte first) return code (see AH=01h)
WORD 0
8 BYTEs (high byte first) logical unit ID
if verb=2400h (TP Started), control block continues:
8 BYTEs (high byte first) TP ID
if verb=2800h (Get ALLOCATE), control block continues:
BYTE type
0 dequeue
1 test
DWORD pointer to CREATE_TP record
if verb=2A00h (Change Logical Unit). control block continues:

DWORD pointer to CREATE_TP_EXIT routine
FFFFFFFFh reject incoming ALLOCATEs
00000000h queue ALLOCATEs
DWORD 0
DWORD pointer to SYSTEM_LOG_EXIT routine, FFFFFFFFh = don't log errors
DWORD 0
BYTE maximum TPs
BYTE 0 stop QUEUE_ALLOCATEs
1 resume QUEUE_ALLOCATEs
DWORD pointer to LU_LU_PASSWORD_EXIT routine, FFFFFFFFh = no exit
DWORD 0

INT 68 - AH = 04h APPC/PC

DS:DX -> control block
12 BYTEs reserved
WORD verb (action)
2500h TP_ENDED
2900h TP_VALID
6 BYTEs 0
DWORD (high byte first) return code (see AH=01h)
WORD 0
8 BYTEs (high byte first) TP_ID
DWORD -> CREATE_TP record (only if verb = 2900h)

INT 68 - AH = 05h APPC/PC - TRANSFER MSG DATA

DS:DX -> control block
12 BYTEs reserved
WORD 1C00h
BYTE 0 user defined
1 NMVT
2 alert subvectors
3 PDSTATS subvectors
5 BYTEs 0
DWORD (high byte first) return code (see AH=01h)
12 BYTEs 0
BYTE if bit 0 clear, add correlation subvector
if bit 1 clear, add product set ID subvector
if bit 2 clear, do SYSLOG
if bit 3 clear, send SSCP_PU_SESSION
BYTE 0
WORD length of data
N BYTEs data

INT 68 - AH = 06h APPC/PC - CHANGE NUMBER OF SESSIONS

DS:DX -> control block
12 BYTEs reserved
WORD 1500h
6 BYTEs 0
WORD (high byte first) primary return code (see AH=02h)
DWORD (high byte first) secondary return code (see AH=01h)
0000h accepted
0001h negotiated
0003h bad logical unit ID
0004h allocation failure, no retry
0005h allocation failure, retry
0151h can't raise limits
0153h all modes must reset
0154h bad SNASVCMG limits
0155h minimum greater than total
0156h mode closed (prim return code = 1)
CNOS mode closed (prim return code = 18h)
0157h bad mode name (prim return code = 1)
CNOS bad mode name (prim return code = 18h)
0159h reset SNA drains
015Ah single not SRC response
015Bh bad partner logical unit
015Ch exceeds maximum allowed
015Dh change SRC drains
015Eh logical unit detached
015Fh CNOS command race reject
8 BYTEs (high byte first) logical unit ID
8 BYTEs blanks
8 BYTEs (high byte first) partner logical unit name
8 BYTEs (high byte first) mode name
BYTE bit 7: use MODE_NAME_SELECT_ALL rather than MODE_NAME
bit 6: set negotiable values
BYTE partner logical unit mode session limit
BYTE minimum CONWINNERS_SOURCE
BYTE maximum CONWINNERS_TARGET
BYTE automatic activation
BYTE 0
BYTE bit 7: drain targer

bit 6: drain source
bit 5: target responsible, not source

INT 68 - AH = 07h APPC/PC - PASSTHROUGH

DS:DX -> control block (format depends on application subsystem)

INT 68 - AH = FAh APPC/PC - ENABLE/DISABLE APPC

AL bit 0 = 0 enable
1 disable

INT 68 - AH = FBh APPC/PC - CONVERT

DS:DX -> control block
12 BYTEs reserved
WORD 1A00h
6 BYTEs 0
DWORD (high byte first) return code
BYTE conversion
0 ASCII to EBCDIC
1 EBCDIC to ASCII
BYTE character set
0 AE
1 A
2 G
WORD length of string to convert
DWORD pointer to source
DWORD pointer to target

INT 68 - AH = FCh APPC/PC - ENABLE/DISABLE MESSAGE TRACING

AL = 00h disable tracing
= 01h enable tracing
DX = number of bytes to keep (0=all)

INT 68 - AH = FDh APPC/PC - ENABLE/DISABLE API VERB TRACING

AL = 00h disable tracing
01h enable tracing

INT 68 - AH = FEh APPC/PC - TRACE DESTINATION

AL = trace destinations
bit 0 storage (DS:DX -> trace stats record)
bit 1 display
bit 2 file (trace written to file OUTPUT.PC)
bit 3 printer

Trace Statistics Record
DWORD pointer to storage trace buffer
WORD max number of 80-byte records in trace
WORD (high-order byte first!) current record number (must init to 0)
DWORD (high-order byte first!) number of records written (init to 0)
DWORD reserved

Note: do not move record while trace is active

INT 68 - AH = FFh APPC/PC - SET PASSTHROUGH

DS:DX -> passthrough exit routine

INT 69 - unused

INT 6A - unused

INT 6B - unused

INT 6C - system resume vector (CONVERTIBLE)

INT 6C - DOS 3.2 Realtime Clock update

INT 6D - Paradise VGA - internal

INT 6E - unused

INT 6F - Novell NetWare - PCOX API (3270 PC terminal interface)

INT 6F - AH = 00h 10-NET - LOGIN

DS:DX -> login record
8 BYTEs user name
8 BYTEs password
12 BYTEs name of SuperStation

Return: CL = security level
AX = status
0000h successful
01FFh time out on response
02FFh network (hardware) error
03FFh invalid password
04FFh local rexource not available
05FFh server resource not available

06FFh already logged in under different name
07FFh login security failure (node)
08FFh not logged in
09FFh position calc error
0AFFh receive subfunction not = send subfunction (i.e. read, write)
0BFFh request function not in range
0CFFh no more server file handle entries left
0DFFh no more shared file table entries left
0EFFh no more user file handle entries left
0FFFh chat permit not on
10FFh not a server on request
11FFh no transporter board error
12FFh time out on send
13FFh item not found (spool item not on queue)
14FFh dos access incompatible
15FFh record already locked
16FFh invalid parameter
17FFh record lock time out error
18FFh currently spooling to named device
19FFh dropped receive message (throttle)
1AFFh open sharing violation
1BFFh no more tuf entries left
1CFFh not file owner on open
1DFFh read security not passed
1EFFh write security not passed
1FFFh group security not passed
20FFh security file failure
21FFh activity file failure
22FFh spool cntrl file failure
23FFh device not mounted (spooling)
24FFh spool file has not been terminated
25FFh device not mounted or is not being shared
26FFh duplicate node id
27FFh file not found error
28FFh no more files
29FFh unknown internal system error
2AFFh print queue is full or corrupted
2BFFh invalid function
2CFFh invalid handle
2DFFh too many files opened
2EFFh path not found
2FFFh named file is active

/* I've gotten one submission which says FFxxh, and another with xxFFh */
/* I don't know which way around these should be, does somebody else know? */

FF01h timeout
FF02h network error
FF03h invalid password
FF04h no local buffer
FF05h superstation not available
FF06h node already logged in
FF07h login not valid from this node
FF08h node ID already in use
FF16h invalid parameter (bad length, invalid node ID, etc)
FF17h record locked by another user
FF18h sent message has been dropped

INT 6F - AH = 01h 10-NET - LOGOFF

DS:DX -> superstation ID or nulls (12 bytes)
Return: CX = number of files closed
AX = status (see function 00h)
FF08h superstation ID not already logged in

INT 6F - AH = 02h 10-NET - STATUS OF NODE

DS:DX -> 512-byte record
8 BYTEs user name (0 if none)
BYTE station type
0 workstation
1 superstation
2 gateway station
3 gateway active
4 logged into multiple superstations
5 reserved
24 BYTEs list of superstations logged into more than one superstation
12 BYTEs node ID
WORD message count for this station (send for user node, receive for superstations)

for superstations only:
WORD drives allocated (bit 0=A:, bit 1=B:,...)
BYTE user service flag
bit 7: gate
6: print permit on

4: SUBMIT is on
3: mail waiting for node
2: calendar waiting for you
1: news waiting for you
0: mail waiting for you
BYTE printers allocated (bit 0=LPT1,...)
BYTE number of unprinted spool files
BYTE number of opened files
BYTE number of logged on nodes
BYTE primary drive (1=A:)
BYTE reserved
N BYTEs list of logged on node IDs (each 12 bytes, max 37 IDs)
(continues at offset 1F4h)
3 BYTEs time: sec/min/hrs
3 BYTEs date: day/mon/year-1980
Return: CF set on error
AX = error code (see function 00h)

INT 6F - AH = 03h 10-NET - GET ADDRESS OF CONFIGURATION TABLE

DS:DI -> node ID (optional)
Return: ES:BX -> record (actually starts at [BX-41])
WORD local device table address
WORD extended network error mapping table address
WORD shared device table address
WORD mounted device table address
BYTE receive buffer counter
BYTE collect buffer counter
WORD TUF address
BYTE enable flag
BYTE FCB keep flag
WORD reserved
---up to here, 10-Net v3.3---
WORD count of dropped Send6F
WORD buffer start address
WORD comm driver base address
WORD send/receive retry count
BYTE number of 550ms loops before timeout
WORD UFH address
WORD CDIR address
WORD LTAB address
WORD SFH address
WORD FTAB address
WORD RLTAB address
WORD SMI address
WORD NTAB address
ES:BX -> WORD address of first CT_DRV
BYTE number of DRV entries
8 BYTEs login name
12 BYTEs node ID (blank-padded)
6 BYTEs node address
BYTE flag
BYTE CT_CFLG (chat permit)
bit 1: sound bell
bit 0: CHAT permit
BYTE CT_PSFLG
bit 5: PRINT permit
bit 4: KB initiated
bit 3: CHAT called FOXPTRM
bit 2: SUBMIT active
bit 1: SUBMIT received
bit 0: SUBMIT permit
BYTE in 10Net flag
WORD receive message count
WORD send message count
WORD retry count
WORD failed count
WORD driver errors
WORD dropped responses/CHATs
9 BYTEs LIST ID/NTAB address (3 entries--LPT1-3)
6 BYTEs AUX ID/NTAB address (2 entries--COM1-2)
BYTE active CB channel
BYTE received 6F messages on queue
9 BYTEs activity counters for channels 1-9
---beyond here, 10-Net v3.3---
BYTE bit 0 = RS232 gate
1 = Send6F gate (user set)
DWORD pointer into gate (user set)
DWORD pointer into 10Net send
N WORDs addresses of timer blocks

INT 6F - AH = 04h 10-NET - SEND

DS:BX -> record

12 BYTEs receiving node's ID
if first byte has high-order bit set, message is directed to the CT_RGATE vector at the receiver
if second byte is 00h, first byte is taken as a CB channel number and delivered to all nodes on same channel
WORD length of data at DX
DS:DX -> data (max 1024 bytes)
Return: CF set on error
AX = error code (see function 00h)

INT 6F - AH = 05h 10-NET - RECEIVE

CX = number of seconds before timeout
DS:DX -> receive buffer
12 BYTEs sending node's ID
WORD length of message
N BYTEs message (maximum 1024 bytes)
Return: CF set on error
AX = error code (see function 00h)
CF clear if successful
AH = FEh if dequeued message is a CB message

INT 6F - AH = 07h 10-NET - LOCK HANDLE

BX = file handle
CX:DX = starting offset in file
SI = record length
Return: CF set on error
AX = error code (see also function 00h)
0002h file not found

INT 6F - AH = 08h 10-NET - UNLOCK HANDLE

BX = file handle
AL = mode
0 unlock all
1 unlock record at CX:DX
Return: CF set on error
AX = error code (see also function 00h)
2 file not found

INT 6F - AH = 09h 10-NET - SUBMIT

DS:BX -> record
12 BYTEs destination node ID (must be logged in)
WORD length+2 of following 'command line' text
N BYTEs command line text (<=100 bytes), system adds CR

INT 6F - AH = 0Ah 10-NET - CHAT

DS:BX -> control parameters
8 BYTEs sender ID, if nulls defaults to node's userID
8 BYTEs destination user ID, 'EVERYONE' may be used
12 BYTEs destination node ID
DS:DX -> chat message
WORD length+2 of following text
N BYTEs text, max 101 bytes

INT 6F - AH = 0Bh 10-NET - LOCK SEMAPHORE, RETURN IMMEDIATELY

AL = drive number or 0
ES:SI = Ethernet address or 0
DS:BX -> 31-byte ASCIZ semaphore name
Return: AL = status
0 successful
1 semaphore currently locked
2 server not responding
3 invalid semaphore name
4 semaphore list is full
5 invalid drive ID
6 invalid Ethernet address
7 not logged in
8 write to network failed
9 semaphore already logged in this CPU
Note: same as INT 60/AH=12h

INT 6F - AH = 0Ch 10-NET - UNLOCK SEMAPHORE

AL = drive number or 0
ES:SI = Ethernet address or 0
DS:BX -> 31-byte ASCIZ semaphore name
Return: AL = status (see AH=0Bh)
1 semaphore not locked
Note: same as INT 60/AH=13h

INT 6F - AH = 0Dh 10-NET - WHO

AL = type code
01h return superstations only
02h return non-superstations only

otherwise return all
CX = length of data
DS:DX -> array of records to be filled
12 BYTEs node ID
BYTE flags
bit 1 = workstation
2 = superstation
3 = xgate
4 = active gate
(if AL=01h, record continues)
BYTE version number
WORD level number of 10Net software in responding node
(if AL=02h, record continues)
8 BYTEs user ID
BYTE version number
WORD level number
Return: CL = number of records returned (responding stations)

INT 6F - AH = 0Eh 10-NET - SPOOL/PRINT

DS:DX -> record
WORD operation code
0 initiate spool
1 abort print
2 close spool
3 delete spool
4 print
5 get report info
6 set chat template
7 queue
8 return queue
9 queue non-spooled file for printing
11 BYTEs file name in FCB format
(if operation code = 00h or 06h, record continues)
BYTE notification
bit 7: queue to top
bit 6: do ID page
bit 5: no form feed
bit 4: reserved
bit 3: explicity queuing only
bit 2: notify at print completion
bit 1: notify server operator/reply
bit 0: notify at print start
BYTE days to keep (FFh=forever)
BYTE bits 0,1: device (1=LPT1)
bits 4-7: remote drive to store spool file (1=A,...)
WORD length of following data area
N BYTEs up to 64 bytes of description
(if operation code = 03h, record continues)
8 BYTEs user ID to associate with filename
(if operation code = 04h, record continues)
WORD block number
8 BYTEs user ID to associate with filename
(if operation code = 05h, record continues)
BYTE RRN to start retrieve
BYTE bits 0,1: local print device (LPTx)
bit 3: if set, return entries for all users
WORD length of following area
N BYTEs up to 1500 bytes to receive $SCNTL records returned
(if operation code = 07h, record continues)
BYTE queue number
BYTE bits 0,1: local print device (LPTx)
WORD number of bytes of test print to be done
BYTE code: 01h prnt device
02h test print count
03h prn
(if operation code = 08h, record continues)
BYTE queue location or $SCNTL location to start access
returns next item for access:
00h-7Fh queued items
80h-FEh non-queued, non-printed items
FFh no more items
WORD unused
WORD length of following area
N BYTEs up to 64 bytes to receive $SCNTL records
(if operation code = 09h, record continues)
3 BYTEs unused
N BYTEs path to non-spooled file to be queued for printing
Return: CF set on error
AX = error code (see also function 00h)
FF17h device not mounted
FF18h already spooling to named device
$SCNTL record:

8 BYTEs user ID
11 BYTEs filename in FCB format
6 BYTEs node ID
3 BYTEs creation date
BYTE flags
bit 7: queue to top
6: do ID page
5: no form feed at end
4: reserved
3: explicit queueing only
2: notify at completion
1: notify server operator/reply
0: notify at start
BYTE retention time in days
BYTE printing device (LPTx)
3 BYTEs date last printed (0 = never)
BYTE device containing spoolfile WORD bytes to print for test print
WORD block number to start print BYTE reserved

INT 6F - AH = 10h 10-NET - ATTACH/DETACH PRINTER

AL = subfunction
00h initiate spooling if LPT1 is mounted
01h terminate spooling if LPT1 is mounted

INT 6F - AH = 11h 10-NET - LOCK FCB

AL = mode
1 sequential 2 random 3 random block
CX = number of records
DS:DX -> FCB
Return: CF set on error
AX = error code (see also function 00h)
2 file not found

INT 6F - AH = 12h 10-NET - UNLOCK FCB

AL = mode
0 sequential
1 random
2 random block
CX = number of records
DS:DX -> FCB
Return: CF set on error
AX = error code (see also function 00h)
2 file not found

INT 6F - AH = 13h 10-NET v3.3 - GET REMOTE CONFIGURATION TABLE ADDRESS

DS:DX -> node ID, 12 bytes blank-padded
Return: CF set on error
AX = error code (see function 00h)
CF clear if successful
ES:BX = configuration table address on given machine

INT 6F - AH = 14h 10-NET v3.3 - GET REMOTE MEMORY

BX:SI = address of remote memory
CX = length (<=1024 bytes)
DS:DX -> node ID, 12 bytes blank-padded
DS:DI -> area to receive remote memory image
Return: CF set on error
AX = error code (see function 00h)
CF clear if successful
CX = amount of memory copied to DS:SI

INT 6F - AX = 1501h 10-NET v3.3 - GET SHARED DEVICE ENTRY

BX = zero-based index
DS:SI -> node ID, 12 bytes blank-padded
ES:DI -> 85-byte buffer
Return: CF set on error
AX = error code (see function 00h)
CF clear if successful
ES:DI buffer contains shared device table entry of BXth device:
8 BYTEs device 8 BYTEs alias 64 BYTEs path
8 BYTEs password BYTE access 4 BYTEs mask

INT 6F - AX = 1502h 10-NET v3.3 - SET SHARED DEVICE ENTRY

DS:SI -> node ID, 12 bytes blank-padded
ES:DI -> valid shared device table entry
Return: CF set on error
AX = error code (see function 00h)

INT 6F - AX = 1503h 10-NET v3.3 - DELETE SHARED DEVICE ENTRY

BX = zero-based index
DS:SI -> node ID, 12 bytes blank-padded
Return: CF set on error
AX = error code (see function 00h)

INT 6F - AH = 17h 10-NET v3.3 - MOUNT

AL = local drive number (0=A:)
BL = remote drive letter or '1'..'3' for LPTn or '4' or '5' for COMx
DS:DX -> node ID, 12 bytes blank-padded
Return: CF set on error
AX = error code (see function 00h)

INT 6F - AH = 18h 10-NET v3.3 - UNMOUNT

AL = local drive number (0=A:)
BL = type
00h disk 01h-03h LPTn 04h,05h COMx
Return: CF set on error
AX = error code (see function 00h)

INT 70 - IRQ8 (AT/XT286/PS50+) - REAL-TIME CLOCK

INT 71 - IRQ9 (AT/XT286/PS50+) - LAN ADAPTER 1

rerouted to INT 0A by BIOS

INT 72 - IRQ10 (AT/XT286/PS50+) - RESERVED

INT 73 - IRQ11 (AT/XT286/PS50+) - RESERVED

INT 74 - IRQ12 (PS50+) - MOUSE INTERRUPT

INT 75 - IRQ13 (AT/XT286/PS50+) - 80287 ERROR

rerouted to INT 02 by BIOS

INT 76 - IRQ14 (AT/XT286/PS50+) - FIXED DISK

INT 77 - IRQ15 (AT/XT286/PS50+) - RESERVED

INT 78 - not used

INT 79 - not used

INT 7A - Novell NetWare - LOW-LEVEL API

INT 7A - AutoCAD Device Interface

INT 7B - not used

INT 7C - not used

INT 7D - not used

INT 7E - not used

INT 7F - not used

INT 80 - reserved for BASIC

INT 81 - reserved for BASIC

INT 82 - reserved for BASIC

INT 83 - reserved for BASIC

INT 84 - reserved for BASIC

INT 85 - reserved for BASIC

INT 86 - Relocated (by NETBIOS) INT 18

INT 86 to F0 - used by BASIC while in interpreter

INT E0 - CP/M-86 function calls

INT E4 - AX = 0005h Logitech Modula v2.0 - MonitorEntry

BX = priority

INT E4 - AX = 0006h Logitech Modula v2.0 - MonitorExit

INT EF - GEM - INTERFACE

CX = 0473h
DS:DX -> GEM parameter block

INT F0 - used by BASIC while in interpreter

INT F1 - reserved for user interrupt

INT F2 - reserved for user interrupt

INT F3 - reserved for user interrupt

INT F4 - reserved for user interrupt

INT F5 - reserved for user interrupt

INT F6 - reserved for user interrupt

INT F7 - reserved for user interrupt

INT F8 - 10 ms INTERVAL TIMER (TANDY???)

INT F9 - reserved for user interrupt

INT FA - USART READY (RS-232C) (TANDY???)

INT FB - USART Rx READY (keyboard) (TANDY???)

INT FC - reserved for user interrupt

INT FD - reserved for user interrupt

INT FE - AT/XT286/PS50+ - destroyed by return from protected mode

INT FF - AT/XT286/PS50+ - destroyed by return from protected mode

INT FF - Z100 - WARM BOOT

End of listing