HelpPC Reference Library

David Jurgens

Topics:

- Assembler Programming Topics
- C Programming Topics
- Hardware Data and Specifications
- Interrupt Services DOS/BIOS/EMS/Mouse
- Uncategorized/Miscellaneous Topics
- Tables and Formats used by DOS & BIOS
- Graphic ASCII table from QBasic

This site is a HTML version of HelpPC Reference Library, a DOS-based hypertext program by David Jurgens. The conversion program was written by Stanislav Sokolov and the C source is available for download under GNU General Public Licence. In the process of conversion several dead links and inconsistencies were fixed in the original library. You can download the off-line version of the library, compressed in ZIP format (approx 270K). The archive includes the original HelpPC for DOS and the program (with source) to convert it to HTML.

The assembly pages are indexed by FreeFind as a part of my site. This means that you can...

powered by FreeFind

Match ALL words and Search assembly site for

What's New Site Map

Assembler Programming Topics

8086 architecture	8086 flags	aaa	aad	aam
aas	adc	add	and	arpl
asm directives	assembler options	bound	bsf	bsr
bswap	bt	btc	btr	bts
call	cbw	cdq	clc	cld
cli	clts	cmc	cmp	cmps
cmpsb	cmpsd	cmpsw	cmpxchg	cwd
cwde	daa	das	dec	directives
div	enter	esc	flags register	fwait
hlt	idiv	imul	in	inc
ins	insb	insd	Instruction Timing	insw
int	into	invd	invlpg	iret
iretd	j...	ja	jae	jb
jbe	jc	jcxz	je	jecxz
jg	jge	jl	jle	jmp
jna	jnae	jnb	jnbe	jnc
jne	jng	jnge	jnl	jnle
jno	jnp	jns	jnz	jo
jp	jpe	jpo	js	jz
lahf	lar	lds	lea	leave
les	lfs	lgdt	lgs	lidt
lldt	lmsw	lock	lods	lodsb
lodsd	lodsw	loop	loope	loopne
loopnz	loopz	lsl	lss	ltr
machine status word	masm options	models	mov	movs
movsb	movsd	movsw	movsx	movzx
msw	mul	neg	nop	not
or	out	outs	outsb	outsd
outsw	pop	popa	popad	popf
popfd	push	pusha	pushad	pushf
pushfd	rcl	rcr	rep	repe

repne repnz repz ret retf
retn rol ror sahf sal
sar sbb scas scasb scasd
scasw segment names setae setb setbe
setc sete setg setge setl
setle setna setnae setnb setnc
setne setng setnge setnl setnle
setno setnp setns setnz seto
setp setpe setpo sets setz
sgdt shl shld shr shrd
sidt sldt smsw stc std
sti stos stosb stosd stosw
str sub test verr verw
wait wbinvd xchg xlat xlatb
xor

Intel 8086 Family Architecture

General Purpose Registers

AH/AL	AX	(EAX)	Accumulator
BH/BL	BX	(EBX)	Base
CH/CL	CX	(ECX)	Counter
DH/DL	DX	(EDX)	Data

(Exx) indicates 386+ 32 bit register

Segment Registers

CS	Code Segment
DS	Data Segment
SS	Stack Segment
ES	Extra Segment
(FS)	386 and newer
(GS)	386 and newer

Pointer Registers

SI (ESI)	Source Index
DI (EDI)	Destination Index
IP	Instruction Pointer

Stack Registers

SP (ESP)	Stack Pointer
BP (EBP)	Base Pointer

Status Registers

FLAGS Status Flags (see FLAGS)

Special Registers (386+ only)

CR0	Control Register 0	DR0	Debug Register 0
CR2	Control Register 2	DR1	Debug Register 1
CR3	Control Register 3	DR2	Debug Register 2
		DR3	Debug Register 3
TR4	Test Register 4	DR6	Debug Register 6
TR5	Test Register 5	DR7	Debug Register 7
TR6	Test Register 6		
TR7	Test Register 7		

Register	Default Segment	Valid Overrides
BP	SS	DS, ES, CS
SI or DI	DS	ES, SS, CS

DI strings	ES	None
SI strings	DS	ES, SS, CS

- see CPU DETECTING Instruction Timing

Keyboard Flags Bytes 0 and 1

```
|7|6|5|4|3|2|1|0|  40:17  Keyboard Flags Byte 0
 | | | | | | | `---- right shift key depressed
 | | | | | | `----- left shift key depressed
 | | | | | `------ CTRL key depressed
 | | | | `------- ALT key depressed
 | | | `-------- scroll-lock is active
 | | `--------- num-lock is active
 | `---------- caps-lock is active
 `----------- insert is active

|7|6|5|4|3|2|1|0|  40:18  Keyboard Flags Byte 1
 | | | | | | | `---- left CTRL key depressed
 | | | | | | `----- left ALT key depressed
 | | | | | `------ system key depressed and held
 | | | | `------- suspend key has been toggled
 | | | `-------- scroll lock key is depressed
 | | `--------- num-lock key is depressed
 | `---------- caps-lock key is depressed
 `----------- insert key is depressed
```

Keyboard Flags Bytes 2 and 3

```
|7|6|5|4|3|2|1|0|  40:97  LED Indicator Flags
 | | | | | | | `---- scroll lock indicator
 | | | | | | `----- num-lock indicator
 | | | | | `------ caps-lock indicator
 | | | | `------- circus system indicator
 | | | `-------- ACK received
 | | `--------- re-send received flag
 | `---------- mode indicator update
 `----------- keyboard transmit error flag

|7|6|5|4|3|2|1|0|  40:96  Keyboard Mode/Type
 | | | | | | | `---- last code was the E1 hidden
code
 | | | | | | `----- last code was the E0 hidden code
 | | | | | `------ right CTRL key depressed
 | | | | `------- right ALT key depressed
```

```
              | | | `-------- 101/102 enhanced keyboard installed
              | | `---------- force num-lock if Rd ID & KBX
              | `------------ last char was first ID char
              `-------------- read ID in process
```

- see INT 9 BDA

INT 9 - Keyboard Interrupt (Hardware Handler)

no input data

related memory:

40:17 = updates keyboard flag byte 0
40:18 = updates keyboard flag byte 1
40:1A = queue head ptr is set to buffer start if Ctrl-Break is hit
40:1C = updates buffer tail pointer for each keystroke; sets queue tail ptr is set to queue start if Ctrl-Break is hit
40:1E = updates keyboard buffer (32 bytes)
40:71 = updates bit 7 of the BIOS break flag if Ctrl-Break is hit
40:72 = updates reset flag with 1234H if Ctrl-Alt-Del pressed
40:96 = indicates keyboard type (AT,PS/2)
40:97 = updates keyboard LED flags (AT,PS/2)
FFFF:0 = reboot code called if Ctrl-Alt-Del pressed

related interrupts:

INT 5 invoked if print screen key pressed
INT 1B invoked if Ctrl-Break key sequence pressed
INT 15,85 invoked on AT if system request key is pressed
INT 15,4F invoked on machines after PC/AT with AL = scan code

- records key press and key release via IRQ1/8259 and stores scan codes in the BIOS buffer located at 40:1C
- keyboard controllers also buffer data when interrupts are disabled at the 8259 interrupt controller
- keyboard controller is capable of storing 16 keystrokes even when interrupts are disabled at the 8259
- normal INT 9 execution takes approximately 500 microseconds; at least one standard XT BIOS is known to take up to 1.3 milliseconds to execute

- see MAKE CODES KB FLAGS

INT 5 - Print Screen

```
        no input data
```

related memory:

```
        50:00   = 00    Print screen has not been called, or upon
return
                        from a call there were no errors
                = 01    Print screen is already in progress
                = FF    Error encountered during printing
```

- invoked from INT 9

INT 1B - BIOS Ctrl-Break Handler Address

- holds address of the BIOS Ctrl-Break interrupt handler
- called by INT 9 after Ctrl-Break key is pressed; INT 9 sets the
  BIOS keyboard head and tail pointers to the buffer start and
  places a zero word in the buffer
- the BIOS initially sets this value to point to a simple IRET but
  DOS re-vectors this to its own code, usually 2 lines of code that
  sets an internal flag (to 3) and then returns via IRET.
- DOS checks this flag on entry to many of its subfunctions. If the
  flag is set, it invokes INT 23.
- pointing this address to a null function with and IRET disables
  Ctrl-Break aborts
- if the INT 1B code chooses to retain control of the system, it
  must issue an EOI for any interrupt pending on the 8259 and reset
  all I/O devices
- should be terminated via an IRET
- should not be called directly by user application

IRET/IRETD - Interrupt Return

```
        Usage:  IRET
                IRETD  (386+ only)
        Modifies flags: AF CF DF IF PF SF TF ZF

        Returns control to point of interruption by popping IP, CS
        and then the Flags from the stack and continues execution at
        this location.  CPU exception interrupts will return to the
        instruction that cause the exception because the CS:IP
placed
        on the stack during the interrupt is the address of the
offending
        instruction.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
iret	32/44	17+m	22	15	1
iret (prot. mode)	-	31+m	38	15	1
iret (to less privilege)	-	55+m	82	36	1
iret (different task, NT=1)	-	169+m	TS	TS+32	1
iretd	-	-	22/38	15	1
iretd (to less privilege)	-	-	82	36	1
iretd (to VM86 mode)	-	-	60	15	1
iretd (different task, NT=1)	-	-	TS	TS+32	1

- 386 timings are listed as real-mode/protected-mode
  - see Instruction Timing

Instruction Clock Cycle Calculation

Some instructions require additional clock cycles due to a "Next Instruction Component" identified by a "+m" in the instruction clock cycle listings. This is due to the prefetch queue being purge on a control transfers. Below is the general rule for calculating "m":

88/86 not applicable
286 "m" is the number of bytes in the next instruction
386 "m" is the number of components in the next instruction
(the instruction coding (each byte), plus the data and the displacement are all considered components)

8088/8086 Effective Address (EA) Calculation

Description	Clock Cycles
Displacement	6
Base or Index (BX,BP,SI,DI)	5
Displacement+(Base or Index)	9
Base+Index (BP+DI,BX+SI)	7
Base+Index (BP+SI,BX+DI)	8
Base+Index+Displacement (BP+DI,BX+SI)	11
Base+Index+Displacement (BP+SI+disp,BX+DI+disp)	12

- add 4 cycles for word operands at odd addresses
- add 2 cycles for segment override
- 80188/80186 timings differ from those of the 8088/8086/80286

Task State Calculation

"TS" is defined as switching from VM/486 or 80286 TSS to one of the following:

	New Task				
Old Task	486 TSS (VM=0)	486 TSS (VM=1)	386 TSS (VM=0)	386 TSS (VM=1)	286 TSS
386 TSS (VM=0)			309	226	282
386 TSS (VM=1)			314	231	287
386 CPU/286 TSS			307	224	280
486 CPU/286 TSS	199	177			180

Miscellaneous

- all timings are for best case and do not take into account wait states, instruction alignment, the state of the prefetch queue, DMA refresh cycles, cache hits/misses or exception processing.
- to convert clocks to nanoseconds divide one microsecond by the processor speed in MegaHertz:

(1000MHz/(n MHz)) = X nanoseconds

- see 8086 Architecture

EBDA - Extended BIOS Data Area EBDA (PS/2)

Offset	Size	Description
00	word	number of bytes allocated to EBDA in Kbytes
01-21	21bytes	reserved
22	dword	device driver far call pointer
26	byte	pointing device flag (1st byte, see below)
27	byte	pointing device flag (2nd byte, see below)
28-2F	8 bytes	reserved

Pointing Device Flag Byte 1

```
|7|6|5|4|3|2|1|0| Offset 26
 | | | | | `------- index count
 | | | | `-------- reserved (0)
 | | | `---------- error
 | | `------------ acknowledge
 | `------------- resend
 `-------------- command in progress
```

Pointing Device Flag Byte 2

```
|7|6|5|4|3|2|1|0| Offset 27
 | | | | | `------- package size
 | `--------------- reserved
 `---------------- device driver far call flag
```

- EBDA is located in highest memory just under 640K on PS/2
- word at BIOS Data Area 40:0E is segment address of EBDA

BDA - BIOS Data Area - PC Memory Map

```
Address Size       Description

00:00 256dwords Interrupt vector table
30:00 256bytes  Stack area used during post and bootstrap
40:00   word    COM1 port address
40:02   word    COM2 port address
40:04   word    COM3 port address
40:06   word    COM4 port address
40:08   word    LPT1 port address
40:0A   word    LPT2 port address
40:0C   word    LPT3 port address
40:0E   word    LPT4 port address (except PS/2)
                Extended BIOS Data Area segment (PS/2, see EBDA)
40:10  2 bytes  Equipment list flags (see INT 11)

              |7|6|5|4|3|2|1|0| 40:10 (value in INT 11 register AL)
               | | | | | | | `- IPL diskette installed
               | | | | | | `-- math coprocessor
               | | | | |-+--- old PC system board RAM < 256K
               | | | | `-- pointing device installed (PS/2)
               | | | `--- not used on PS/2
               | | `------ initial video mode
               `--------- # of diskette drives, less 1

              |7|6|5|4|3|2|1|0| 40:11  (value in INT 11 register AH)
               | | | | | | | `- 0 if DMA installed
               | | | | `------ number of serial ports
               | | | `------- game adapter
               | | `-------- not used, internal modem (PS/2)
               `----------- number of printer ports

40:12   byte    PCjr: infrared keyboard link error count
40:13   word    Memory size in Kbytes  (see INT 12)
40:15   byte    Reserved
40:16   byte    PS/2 BIOS control flags
40:17   byte    Keyboard flag byte 0 (see KB FLAGS)

              |7|6|5|4|3|2|1|0| keyboard flag byte 0
```

```
          | | | | | | | `--- right shift key depressed
          | | | | | | `---- left shift key depressed
          | | | | | `----- CTRL key depressed
          | | | | `------ ALT key depressed
          | | | `------- scroll-lock is active
          | | `-------- num-lock is active
          | `--------- caps-lock is active
          `---------- insert is active
```

40:18 byte Keyboard flag byte 1 (see KB FLAGS)

```
         |7|6|5|4|3|2|1|0| keyboard flag byte
          | | | | | | | `--- left CTRL key depressed
          | | | | | | `---- left ALT key depressed
          | | | | | `----- system key depressed and held
          | | | | `------ suspend key has been toggled
          | | | `------- scroll lock key is depressed
          | | `-------- num-lock key is depressed
          | `--------- caps-lock key is depressed
          `---------- insert key is depressed
```

40:19 byte Storage for alternate keypad entry
40:1A word Offset from 40:00 to keyboard buffer head
40:1C word Offset from 40:00 to keyboard buffer tail
40:1E 32bytes Keyboard buffer (circular queue buffer)
40:3E byte Drive recalibration status

```
         |7|6|5|4|3|2|1|0| drive recalibration status
          | | | | | | | `-- 1=recalibrate drive 0
          | | | | | | `--- 1=recalibrate drive 1
          | | | | | `---- 1=recalibrate drive 2
          | | | | `----- 1=recalibrate drive 3
          | | `---------- unused
          `----------- 1=working interrupt flag
```

40:3F byte Diskette motor status

```
         |7|6|5|4|3|2|1|0| diskette motor status
          | | | | | | | `-- 1=drive 0 motor on
          | | | | | | `--- 1=drive 1 motor on
          | | | | | `---- 1=drive 2 motor on
          | | | | `----- 1=drive 3 motor on
          | | `---------- unused
          `----------- 1=write operation
```

40:40 byte Motor shutoff counter (decremented by INT 8)

```
        40:41   byte    Status of last diskette operation (see INT
13,1)

                |7|6|5|4|3|2|1|0| status of last diskette operation
                 | | | | | | | `--- invalid diskette command
                 | | | | | | `---- diskette address mark not found
                 | | | | | `----- sector not found
                 | | | | `------ diskette DMA error
                 | | | `------- CRC check / data error
                 | | `-------- diskette controller failure
                 | `--------- seek to track failed
                 `---------- diskette time-out

        40:42  7 bytes  NEC diskette controller status (see FDC)
        40:49   byte    Current video mode  (see VIDEO MODE)
        40:4A   word    Number of screen columns
        40:4C   word    Size of current video regen buffer in bytes
        40:4E   word    Offset of current video page in video regen
buffer
        40:50  8 words  Cursor position of pages 1-8, high order
byte=row
                        low order byte=column; changing this data
isn't
                        reflected immediately on the display
        40:60   byte    Cursor ending (bottom) scan line (don't
modify)
        40:61   byte    Cursor starting (top) scan line (don't
modify)
        40:62   byte    Active display page number
        40:63   word    Base port address for active 6845 CRT
controller
                        3B4h = mono, 3D4h = color
        40:65   byte    6845 CRT mode control register value (port
3x8h)
                        EGA/VGA values emulate those of the MDA/CGA
        40:66   byte    CGA current color palette mask setting
(port 3d9h)
                        EGA and VGA values emulate the CGA
        40:67   dword   CS:IP for 286 return from protected mode
                dword   Temp storage for SS:SP during shutdown
                dword   Day counter on all products after AT
                dword   PS/2 Pointer to reset code with memory
preserved
                5 bytes Cassette tape control (before AT)
        40:6C   dword   Daily timer counter, equal to zero at
midnight;
```

```
                        incremented by INT 8; read/set by INT 1A
        40:70   byte    Clock rollover flag, set when 40:6C exceeds
24hrs
        40:71   byte    BIOS break flag, bit 7 is set if Ctrl-Break
was
                        *ever* hit; set by INT 9
        40:72   word    Soft reset flag via Ctl-Alt-Del or JMP
FFFF:0

                        1234h  Bypass memory tests & CRT
initialization
                        4321h  Preserve memory
                        5678h  System suspend
                        9ABCh  Manufacturer test
                        ABCDh  Convertible POST loop
                        ????h  many other values are used during
POST

        40:74   byte    Status of last hard disk operation (see INT
13,1)
        40:75   byte    Number of hard disks attached
        40:76   byte    XT fixed disk drive control byte
        40:77   byte    Port offset to current fixed disk adapter
        40:78  4 bytes  Time-Out value for LPT1,LPT2,LPT3(,LPT4
except PS/2)
        40:7C  4 bytes  Time-Out value for COM1,COM2,COM3,COM4
        40:80   word    Keyboard buffer start offset (seg=40h,BIOS
10-27-82)
        40:82   word    Keyboard buffer end offset (seg=40h,BIOS 10-
27-82)
        40:84   byte    Rows on the screen (less 1, EGA+)
        40:85   word    Point height of character matrix (EGA+)
                byte    PCjr: character to be repeated if the
typematic
                        repeat key takes effect
        40:86   byte    PCjr: initial delay before repeat key
action begins
        40:87   byte    PCjr: current Fn function key number
                byte    Video mode options (EGA+)

                        |7|6|5|4|3|2|1|0| Video mode options (EGA+)
                         | | | | | | | `-- 1=alphanumeric cursor emulation
enabled
                         | | | | | | `--- 1=video subsystem attached to
monochrome
```

```
                | | | | | `---- reserved
                | | | | `----- 1=video subsystem is inactive
                | | | `------ reserved
                | `--------- video RAM  00-64K  10-192K  01-128K
11-256K
                `---------- video mode number passed to INT 10,
function 0

        40:88   byte    PCjr: third keyboard status byte
                        EGA feature bit switches, emulated on VGA

                |7|6|5|4|3|2|1|0| EGA feature bit switches (EGA+)
                 | | | | | | | `-- EGA SW1 config (1=off)
                 | | | | | | `--- EGA SW2 config (1=off)
                 | | | | | `---- EGA SW3 config (1=off)
                 | | | | `----- EGA SW4 config (1=off)
                 | | | `------ Input FEAT0 (ISR0 bit 5) after
output on FCR0
                 | | `------- Input FEAT0 (ISR0 bit 6) after output
on FCR0
                 | `-------- Input FEAT1 (ISR0 bit 5) after output
on FCR1
                 `--------- Input FEAT1 (ISR0 bit 6) after output
on FCR1

        40:89   byte    Video display data area (MCGA and VGA)

                |7|6|5|4|3|2|1|0| Video display data area (MCGA and
VGA)
                 | | | | | | | `-- 1=VGA is active
                 | | | | | | `--- 1=gray scale is enabled
                 | | | | | `---- 1=using monochrome monitor
                 | | | | `----- 1=default palette loading is
disabled
                 | | | `------ see table below
                 | | `------- reserved
                 | `--------  1=display switching enabled
                 `--------- alphanumeric scan lines (see table
below)
```

Bit7	Bit4	Scan Lines
0	0	350 line mode
0	1	400 line mode
1	0	200 line mode
1	1	reserved

```
        40:8A   byte    Display Combination Code (DCC) table index
(EGA+)
        40:8B   byte    Last diskette data rate selected

                |7|6|5|4|3|2|1|0| last diskette data rate selected
                 | | | | `--------- reserved
                 | | `------------ last floppy drive step rate
selected
                 `-------------- last floppy data rate selected
```

Data Rate		Step Rate	
00	500K bps	00	step rate time of 0C
01	300K bps	01	step rate time of 0D
10	250K bps	10	step rate time of 0A
11	reserved	11	reserved

```
        40:8C   byte    Hard disk status returned by controller
        40:8D   byte    Hard disk error returned by controller
        40:8E   byte    Hard disk interrupt control flag(bit
7=working int)
        40:8F   byte    Combination hard/floppy disk card when bit
0 set
        40:90  4 bytes  Drive 0,1,2,3 media state

                |7|6|5|4|3|2|1|0| drive media state (4 copies)
                 | | | | | `------- drive/media state (see below)
                 | | | | `------- reserved
                 | | | `------- 1=media/drive established
                 | | `------- double stepping required
                 `--------- data rate:  00=500K bps    01=300K bps
                                        10=250K bps    11=reserved
```

Bits	
210	**Drive Media State**
000	360Kb diskette/360Kb drive not established
001	360Kb diskette/1.2Mb drive not established
010	1.2Mb diskette/1.2Mb drive not established
011	360Kb diskette/360Kb drive established
100	360Kb diskette/1.2Mb drive established
101	1.2Mb diskette/1.2Mb drive established
110	Reserved
111	None of the above

```
        40:94   byte    Track currently seeked to on drive 0
        40:95   byte    Track currently seeked to on drive 1
        40:96   byte    Keyboard mode/type
```

```
              |7|6|5|4|3|2|1|0| Keyboard mode/type
               | | | | | | | `--- last code was the E1 hidden code
               | | | | | | `---- last code was the E0 hidden code
               | | | | | `----- right CTRL key depressed
               | | | | `------ right ALT key depressed
               | | | `------- 101/102 enhanced keyboard installed
               | | `-------- force num-lock if Rd ID & KBX
               | `--------- last char was first ID char
               `---------- read ID in process

      40:97   byte    Keyboard LED flags

              |7|6|5|4|3|2|1|0| Keyboard LED flags
               | | | | | | | `--- scroll lock indicator
               | | | | | | `---- num-lock indicator
               | | | | | `----- caps-lock indicator
               | | | | `------ circus system indicator
               | | | `------- ACK received
               | | `-------- re-send received flag
               | `--------- mode indicator update
               `---------- keyboard transmit error flag

      40:98   dword   Pointer to user wait complete flag
      40:9C   dword   User wait Time-Out value in microseconds
      40:A0   byte    RTC wait function flag

              |7|6|5|4|3|2|1|0| INT 15,86 RTC wait function flag
               | | | | | | | `--- 1= wait pending
               | | `-------------- not used
               `---------------- 1=INT 15,86 wait time elapsed

      40:A1   byte    LANA DMA channel flags
      40:A2  2 bytes  Status of LANA 0,1
      40:A4   dword   Saved hard disk interrupt vector
      40:A8   dword   BIOS Video Save/Override Pointer Table
address
                      (see VIDEO TABLES)
      40:AC  8 bytes  Reserved
      40:B4   byte    Keyboard NMI control flags (convertible)
      40:B5   dword   Keyboard break pending flags (convertible)
      40:B9   byte    Port 60 single byte queue (convertible)
      40:BA   byte    Scan code of last key (convertible)
      40:BB   byte    NMI buffer head pointer (convertible)
      40:BC   byte    NMI buffer tail pointer (convertible)
      40:BD  16bytes  NMI scan code buffer (convertible)
      40:CE   word    Day counter (convertible and after)
```

```
        40:F0  16bytes  Intra-Applications Communications Area (IBM
Technical
                        Reference incorrectly locates this at 50:F0-
50:FF)
```

Address	Size	Description (BIOS/DOS Data Area)
50:00	byte	Print screen status byte 00 = PrtSc not active, 01 = PrtSc in progress FF = error
50:01	3 bytes	Used by BASIC
50:04	byte	DOS single diskette mode flag, 0=A:, 1=B:
50:05	10bytes	POST work area
50:0F	byte	BASIC shell flag; set to 2 if current shell
50:10	word	BASICs default DS value (DEF SEG)
50:12	dword	Pointer to BASIC INT 1C interrupt handler
50:16	dword	Pointer to BASIC INT 23 interrupt handler
50:1A	dword	Pointer to BASIC INT 24 disk error handler
50:20	word	DOS dynamic storage
50:22	14bytes	DOS diskette initialization table (INT 1E)
50:30	4bytes	MODE command
70:00		I/O drivers from IO.SYS/IBMBIO.COM

The following map varies in size and locus

07C0:0		Boot code is loaded here at startup (31k mark)
A000:0		EGA/VGA RAM for graphics display mode 0Dh & above
B000:0		MDA RAM, Hercules graphics display RAM
B800:0		CGA display RAM
C000:0		EGA/VGA BIOS ROM (thru C7FF)
C400:0		Video adapter ROM space
C600:0	256bytes	PGA communication area
C800:0	16K	Hard disk adapter BIOS ROM
C800:5		XT Hard disk ROM format, AH=Drive, AL=Interleave
D000:0	32K	Cluster adapter BIOS ROM
D800:0		PCjr conventionalsoftware cartridge address
E000:0	64K	Expansion ROM space (hardwired on AT+)
	128K	PS/2 System ROM (thru F000)

F000:0		System monitor ROM
		PCjr: software cartridge override address
F400:0		System expansion ROMs
F600:0		IBM ROM BASIC (AT)
F800:0		PCjr software cartridge override address
FC00:0		BIOS ROM
FF00:0		System ROM
FFA6:E		ROM graphics character table
FFFF:0		ROM bootstrap code
FFFF:5	8 bytes	ROM date (not applicable for all clones)
FFFF:E	byte	ROM machine id (see MACHINE ID)

INT 11 - BIOS Equipment Determination / BIOS Equipment Flags

```
        no input data

        on return:
        AX contains the following bit flags:

        |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  AX
         | | | | | | | | | | | | | | | `---- IPL diskette installed
         | | | | | | | | | | | | | | `----- math coprocessor
         | | | | | | | | | | | | | `-------- old PC system board RAM
< 256K
         | | | | | | | | | | | | | `----- pointing device installed
(PS/2)
         | | | | | | | | | | | | `------ not used on PS/2
         | | | | | | | | | | | `--------- initial video mode
         | | | | | | | | | `------------ # of diskette drives, less 1
         | | | | | | | | `------------- 0 if DMA installed
         | | | | `------------------ number of serial ports
         | | | `------------------- game adapter installed
         | | `--------------------- unused, internal modem (PS/2)
         `----------------------- number of printer ports


        - bits 3 & 2,  system board RAM if less than 256K
motherboard
            00 - 16K                 01 - 32K
            10 - 16K                 11 - 64K (normal)

        - bits 5 & 4,  initial video mode
            00 - unused              01 - 40x25 color
            10 - 80x25 color         11 - 80x25 monochrome


        - bits 7 & 6,  number of disk drives attached, when bit 0=1
            00 - 1 drive             01 - 2 drives
            10 - 3 drive             11 - 4 drives


        - returns data stored at BIOS Data Area location 40:10
        - some flags are not guaranteed to be correct on all
```

machines
- bit 13 is used on the PCjr to indicate serial printer

INT 12 - Memory Size Determination

```
        no input data


        on return:
        AX  = the number of contiguous 1k memory blocks found at
startup

        - contiguous memory does not include video memory or
extended RAM
```

INT 8 - System timer

no input data

related memory:

40:6C = Daily timer counter (4 bytes)
40:70 = 24 hr overflow flag (1 byte)
40:67 = Day counter on all products after AT
40:40 = Motor shutoff counter - decremented until 0 then
shuts off diskette motor

- INT 1C is invoked as a user interrupt
- the byte at 40:70 is a flag that certain DOS functions use and adjust the date if necessary. Since this is a flag and not a counter it results in DOS (not the RTC) losing days when several midnights pass before a DOS call
- generated 18.2 times per second by the 8253 Programmable Interval Timer (PIT)
- normal INT 8 execution takes approximately 100 microseconds

- see 8253

INT 1C - System Timer Tick (User Routine)

- called by INT 8, approximately 18.206 times per second
  (unless the program modifies the 8253 PIT
- this is a user routine which the BIOS defaults to a simple IRET
- this interrupt vector can be used for TSR popup utilities,
  animated graphics updates and event polling
- since many TSR's don't maintain the integrity of the interrupt
  call chain, relying on this for TSR popup isn't recommended

8253/8254 PIT - Programmable Interval Timer

```
        Port 40h, 8253 Counter 0 Time of Day Clock (normally mode 3)
        Port 41h, 8253 Counter 1 RAM Refresh Counter (normally mode
2)
        Port 42h, 8253 Counter 2 Cassette and Speaker Functions
        Port 43h, 8253 Mode Control Register, data format:

        |7|6|5|4|3|2|1|0|  Mode Control Register
         | | | | | | | `---- 0=16 binary counter, 1=4 decade BCD
counter
         | | | | `--------- counter mode bits
         | | `------------ read/write/latch format bits
         `---------------- counter select bits (also 8254 read back
command)
```

Bits
76 Counter Select Bits
00 select counter 0
01 select counter 1
10 select counter 2
11 read back command (8254 only, illegal on 8253, see below)

Bits
54 Read/Write/Latch Format Bits
00 latch present counter value
01 read/write of MSB only
10 read/write of LSB only
11 read/write LSB, followed by write of MSB

Bits
321 Counter Mode Bits
000 mode 0, interrupt on terminal count; countdown, interrupt, then wait for a new mode or count; loading a new count in the middle of a count stops the countdown
001 mode 1, programmable one-shot; countdown with optional restart; reloading the counter will not affect the countdown until after the following trigger
010 mode 2, rate generator; generate one pulse after

```
'count' CLK
             cycles; output remains high until after the new
countdown has
             begun; reloading the count mid-period does not take
affect
             until after the period
        011  mode 3, square wave rate generator; generate one pulse
after
             'count' CLK cycles; output remains high until 1/2 of
the next
             countdown; it does this by decrementing by 2 until
zero, at
             which time it lowers the output signal, reloads the
counter
             and counts down again until interrupting at 0;
reloading the
             count mid-period does not take affect until after the
period
        100  mode 4, software triggered strobe; countdown with
output high
             until counter zero;  at zero output goes low for one
CLK
             period;  countdown is triggered by loading counter;
reloading
             counter takes effect on next CLK pulse
        101  mode 5, hardware triggered strobe; countdown after
triggering
             with output high until counter zero; at zero output
goes low
             for one CLK period
```

Read Back Command Format (8254 only)

```
        |7|6|5|4|3|2|1|0| Read Back Command (written to Mode
Control Reg)
         | | | | | | | `--- must be zero
         | | | | | | `---- select counter 0
         | | | | | `----- select counter 1
         | | | | `------ select counter 2
         | | | `------- 0 = latch status of selected counters
         | | `-------- 0 = latch count of selected counters
         `------------ 11 = read back command
```

Read Back Command Status (8254 only, read from counter register)

```
        |7|6|5|4|3|2|1|0|  Read Back Command Status
         | | | | | | | `--- 0=16 binary counter, 1=4 decade BCD
counter
         | | | | `-------- counter mode bits (see Mode Control Reg
above)
         | | `------------ read/write/latch format (see Mode Control
Reg)
         | `------------- 1=null count (no count set), 0=count
available
         `-------------- state of OUT pin (1=high, 0=low)
```

- the 8253 is used on the PC & XT, while the 8254 is used on the AT+
- all counters are decrementing and fully independent
- the PIT is tied to 3 clock lines all generating 1.19318 MHz.
- the value of 1.19318MHz is derived from (4.77/4 MHz) and has it's
  roots based on NTSC frequencies
- counters are 16 bit quantities which are decremented and then
  tested against zero. Valid range is (0-65535). To get a value
  of 65536 clocks you must specify 0 as the default count since
  65536 is a 17 bit value.
- reading by latching the count doesn't disturb the countdown but
  reading the port directly does; except when using the 8254 Read
  Back Command
- counter 0 is the time of day interrupt and is generated
  approximately 18.2 times per sec. The value 18.2 is derived from
  the frequency 1.10318/65536 (the normal default count).
- counter 1 is normally set to 18 (dec.) and signals the 8237 to do
  a RAM refresh approximately every 15æs
- counter 2 is normally used to generate tones from the speaker
  but can be used as a regular counter when used in conjunction
  with the 8255
- newly loaded counters don't take effect until after a an output

pulse or input CLK cycle depending on the mode
- the 8253 has a max input clock rate of 2.6MHz, the 8254 has max
input clock rate of 10MHz

Programming considerations:

1. load Mode Control Register
2. let bus settle (jmp $+2)
3. write counter value
4. if counter 0 is modified, an INT 8 handler must be written to
call the original INT 8 handler every 18.2 seconds. When it
does call the original INT 8 handler it must NOT send and EOI
to the 8259 for the timer interrupt, since the original INT 8
handler will send the EOI also.

Example code:

```
countdown  equ  8000h ; approx 36 interrupts per second

   cli
   mov  al,00110110b  ; bit 7,6 = (00) timer counter 0
                      ; bit 5,4 = (11) write LSB then MSB
                      ; bit 3-1 = (011) generate square wave
                      ; bit 0 = (0) binary counter
   out  43h,al        ; prep PIT, counter 0, square wave&init count
   jmp  $+2
   mov  cx,countdown  ; default is 0x0000 (65536) (18.2 per sec)
                      ; interrupts when counter decrements to 0
   mov  al,cl         ; send LSB of timer count
   out  40h,al
   jmp  $+2
   mov  al,ch         ; send MSB of timer count
   out  40h,al
   jmp  $+2
   sti
```

8259 Programmable Interrupt Controller (PIC)

Initialization Command Word 1 at Port 20h and A0h

```
|7|6|5|4|3|2|1|0|  ICW1
 | | | | | | | `---- 1=ICW4 is needed, 0=no ICW4 needed
 | | | | | | `----- 1=single 8259, 0=cascading 8259's
 | | | | | `------ 1=4 byte interrupt vectors, 0=8 byte int
vectors
 | | | | `------- 1=level triggered mode, 0=edge triggered
mode
 | | | `-------- must be 1 for ICW1 (port must also be 20h
or A0h)
 `-------------- must be zero for PC systems
```

Initialization Command Word 2 at Port 21h and A1h

```
|7|6|5|4|3|2|1|0|  ICW2
 | | | | | `--------- 000= on 80x86 systems
 `----------------- A7-A3 of 80x86 interrupt vector
```

Initialization Command Word 3 at Port 21h and A1h

```
|7|6|5|4|3|2|1|0|  ICW3 for Master Device
 | | | | | | | `---- 1=interrupt request 0 has slave, 0=no
slave
 | | | | | | `----- 1=interrupt request 1 has slave, 0=no
slave
 | | | | | `------ 1=interrupt request 2 has slave, 0=no
slave
 | | | | `------- 1=interrupt request 3 has slave, 0=no
slave
 | | | `-------- 1=interrupt request 4 has slave, 0=no slave
 | | `---------- 1=interrupt request 5 has slave, 0=no slave
 | `----------- 1=interrupt request 6 has slave, 0=no slave
 `------------ 1=interrupt request 7 has slave, 0=no slave

|7|6|5|4|3|2|1|0|  ICW3 for Slave Device
 | | | | | `--------- master interrupt request slave is
attached to
 `----------------- must be zero
```

Initialization Command Word 4 at Port 21h and A1h

```
|7|6|5|4|3|2|1|0|  ICW4
 | | | | | | | `---- 1 for 80x86 mode, 0 = MCS 80/85 mode
 | | | | | | `----- 1 = auto EOI, 0=normal EOI
 | | | | `-------- slave/master buffered mode (see below)
 | | | `--------- 1 = special fully nested mode (SFNM),
0=sequential
 `--------------- unused (set to zero)
```

Bits

32	**Buffering Mode**
00	not buffered
01	not buffered
10	buffered mode slave (PC mode)
11	buffered mode master (PC mode)

Operation Control Word 1 / Interrupt Mask Reg. (Ports 21h & A1h)

```
|7|6|5|4|3|2|1|0|  OCW1 - IMR Interrupt Mask Register
 | | | | | | | `---- 0 = service IRQ0, 1 = mask off
 | | | | | | `----- 0 = service IRQ1, 1 = mask off
 | | | | | `------ 0 = service IRQ2, 1 = mask off
 | | | | `------- 0 = service IRQ3, 1 = mask off
 | | | `-------- 0 = service IRQ4, 1 = mask off
 | | `--------- 0 = service IRQ5, 1 = mask off
 | `---------- 0 = service IRQ6, 1 = mask off
 `----------- 0 = service IRQ7, 1 = mask off
```

Operation Control Word 2 / Interrupt Command Reg. (Ports 20h & A0h)

```
|7|6|5|4|3|2|1|0|  OCW2 - ICR Interrupt Command Register
 | | | | | `-------- interrupt request level to act upon
 | | | | `--------- must be 0 for OCW2
 | | | `---------- must be 0 for OCW2
 `--------------- EOI type (see table)
```

Bits

765	**EOI - End Of Interrupt code (PC specific)**
001	non-specific EOI command
010	NOP
011	specific EOI command
100	rotate in automatic EOI mode
101	rotate on non-specific EOI command
110	set priority command (uses bits 2-0)

111 rotate on specific EOI command

Operation Control Word 3 (Ports 20h & A0h)

```
|7|6|5|4|3|2|1|0|  OCW3
 | | | | | | | `--- 1=read IRR on next read, 0=read ISR on next read
 | | | | | | `---- 1=act on value of bit 0, 0=no action if bit 0 set
 | | | | | `----- 1=poll command issued, 0=no poll command issued
 | | | | `------ must be 1 for OCW3
 | | | `------- must be 0 for OCW3
 | | `-------- 1=set special mask, 0=reset special mask
 | `--------- 1=act on value of bit 5, 0=no action if bit 5 set
 `---------- not used (zero)
```

Other Registers

IRR - Interrupt Request Register, maintains a bit vector indicating which IRQ hardware events are awaiting service. Highest level interrupt is reset when the CPU acknowledges interrupt.

ISR - In Service Register, tracks IRQ line currently being serviced. Updated by EOI command.

Hardware Interrupt Sequence of Events:

1. 8259 IRQ signal is raised high by hardware setting the corresponding IRR bits true.

2. PIC evaluates the interrupt requests and signals the CPU where appropriate.

3. CPU acknowledges the INT by pulsing INTA (inverted)

4. INTA signal from CPU is received by the PIC, which then sets the

highest priority ISR bit, and clears the corresponding IRR bit

5. CPU sends a second INTA pulse which causes the PIC to send the interrupt ID byte onto the bus. CPU begins interrupt processing.

6. Interrupts of lower and equal priority are blocked until a Non-Specific EOI (20h) is sent to the command port.

Initialization Procedure

Initialization

1. write ICW1 to port 20h
2. write ICW2 to port 21h
3. if ICW1 bit D1=1 do nothing
   if ICW1 bit D1=0 write ICW3 to port 20h
4. write ICW4 to port 21h
5. OCW's can follow in any order

8259 Programmable Interrupt Controller Notes

- Operation Command Word (OCW), commands that set the 8259 in various interrupt modes. These can be written to the 8259 anytime after initialization.
- The 8259 differentiates between the OCW1, OCW2 and OCW3 by the port address and the value of the data bits D4 and D3. ICW2 through ICW4 are order related and must follow ICW1. ICW1 is identified by an even port number and data bit D4 set to 1.
- PCs operate in fully nested mode, where a Non-Specific EOI resets

the interrupt identified by the highest bit value in the ISR
- 8259s can be chained together where the INT pin (output) of a
slave 8259 can be used as the input to an IRQ line allowing up
to 64 priority vectored interrupts. AT level machines use two
8259's for a total of 16 hardware interrupt levels
- the first 8259 ports are located at 20h and 21h
- the second 8259 ports are located at A0h and A1h
- PC and AT interrupts are Edge Triggered while PS/2's are Level
Triggered
- some ASIC chips designed for Tandy 1000 Systems malfunction if
specific and non-specific EOIs are mixed
- for a more in-depth discussion of the 8259, see Intel's "Micro-
processor and Peripherals Handbook, Volume I"

- see PORTS and INT TABLE

Intel Defined CPU Exception Table (see notes)

Interrupt	Function
0	Divide by zero
1	Single step
2	Non-maskable (NMI)
3	Breakpoint
4	Overflow trap
5	BOUND range exceeded (186,286,386)
6	Invalid opcode (186,286,386)
7	Coprocessor not available (286,386)
8	Double fault exception (286,386)
9	Coprocessor segment overrun (286,386)
A	Invalid task state segment (286,386)
B	Segment not present (286,386)
C	Stack exception (286,386)
D	General protection exception (286,386)
E	Page fault (286,386)
F	Reserved
10	Coprocessor error (286,386)

IBM PC Hardware Interrupt Table (in order of priority)

IRQ#	Interrupt	Function
IRQ0	8	timer (55ms intervals, 18.2 per second)
IRQ1	9	keyboard service required
IRQ2	A	slave 8259 or EGA/VGA vertical retrace
IRQ8	70	real time clock (AT,XT286,PS50+)
IRQ9	71	software redirected to IRQ2 (AT,XT286,PS50+)
IRQ10	72	reserved (AT,XT286,PS50+)
IRQ11	73	reserved (AT,XT286,PS50+)
IRQ12	74	mouse interrupt (PS50+)
IRQ13	75	numeric coprocessor error (AT,XT286,PS50+)
IRQ14	76	fixed disk controller (AT,XT286,PS50+)
IRQ15	77	reserved (AT,XT286,PS50+)

IRQ3	B	COM2 or COM4 service required, (COM3-COM8 on MCA PS/2)
IRQ4	C	COM1 or COM3 service required
IRQ5	D	fixed disk or data request from LPT2
IRQ6	E	floppy disk service required
IRQ7	F	data request from LPT1 (unreliable on IBM mono)

Interrupt Table as Implemented by System BIOS/ DOS

INT #	Locus	Function
0	CPU	divide by zero
1	CPU	single step
2	CPU	non-maskable
3	CPU	breakpoint
4	CPU	overflow trap
5	BIOS	print screen
6	CPU	Invalid opcode (186,286,386)
7	CPU	coprocessor not available (286,386)
8	IRQ0	timer (55ms intervals, 18.21590 per second)
9	IRQ1	keyboard service required (see INT 9)
A	IRQ2	slave 8259 or EGA/VGA vertical retrace
B	IRQ3	COM2 service required (PS/2 MCA COM3-COM8)
C	IRQ4	COM1 service required
D	IRQ5	fixed disk or data request from LPT2
E	IRQ6	floppy disk service required
F	IRQ7	data request from LPT1 (unreliable on IBM mono)
10	BIOS	video (see INT 10)
11	BIOS	Equipment determination (see INT 11)
12	BIOS	memory size (see INT 12)
13	BIOS	disk I/O service (see INT 13)
14	BIOS	serial communications (see INT 14)
15	BIOS	system services, cassette (see INT 15)
16	BIOS	keyboard services (see INT 16)
17	BIOS	parallel printer (see INT 17)
18	BIOS	ROM BASIC loader
19	BIOS	bootstrap loader (unreliable, see INT 19)

1A	BIOS	time of day (see INT 1A)
1B	BIOS	user defined ctrl-break handler (see INT 1B)
1C	BIOS	user defined clock tick handler (see INT 1C)
1D	BIOS	6845 video parameter pointer
1E	BIOS	diskette parameter pointer (base table)
1F	BIOS	graphics character table
20	DOS	general program termination
21	DOS	function request services (see INT 21)
22	DOS	terminate address (see INT 22)
23	DOS	control break termination address (see INT 23)
24	DOS	critical error handler (see INT 24)
25	DOS	absolute disk read (see INT 25)
26	DOS	absolute disk write (see INT 26)
27	DOS	terminate and stay resident (see INT 27)
28	DOS	idle loop, issued by DOS when idle (see INT 28)
29	DOS	fast TTY console I/O (see INT 29)
2A	DOS	critical section and NETBIOS (see INT 2A)
2B	DOS	internal, simple IRET in DOS 2.0-5.0
2C	DOS	internal, simple IRET in DOS 2.0-5.0
2D	DOS	internal, simple IRET in DOS 2.0-5.0
2E	DOS	exec command from base level command interpreter (see INT 2E)
2F	DOS	multiplexer (see INT 2F)
30-31	CPM	far jump vector for CPM (not an interrupt)
31	DPMI	DOS Protected Mode Interface (for DOS extenders)
32		reserved
33		mouse support (see INT 33)
34-3E		Microsoft/Borland floating point emulation
3F		overlay manager
40	BIOS	hard disk
41	BIOS	fixed disk 0 parameters pointer (see INT 13,9)
42	BIOS	relocated video handler (EGA/VGA/PS)
43	BIOS	user font table (EGA/VGA/PS)
44	BIOS	first 128 graphics characters (also Netware)
45	BIOS	reserved for BIOS
46	BIOS	fixed disk 1 parameters ptr (see INT 13,9/ INT 41)
47	BIOS	reserved for BIOS
48	BIOS	PCjr cordless keyboard translation

```
        49      BIOS    PCjr non-keyboard scancode translation table
        4A      BIOS    user alarm  (AT,CONV,PS/2) (see INT 4A)
        4B-4F   BIOS    reserved
        50      BIOS    periodic alarm from timer (PS/2)
        51-58   BIOS    reserved
        59      BIOS    GSS Computer Graphics Interface
        5A      BIOS    cluster adapter BIOS entry point
        5B      BIOS    cluster adapter boot
        5C      NETBIOS NETBIOS interface, TOPS interface
        5D-5F   BIOS    reserved for BIOS
        60-67           reserved for user software interrupts
        67      EMS     LIM/EMS specification (see INT 67)
        68              APPC
        69-6B           reserved by IBM
        6C      DOS     DOS 3.2 real time clock update
                BIOS    system resume vector
        6D-6F           reserved
        70      IRQ8    real time clock  (AT,XT286,PS50+, see INT
15)
        71      IRQ9    software redirected to IRQ2     (AT,XT286,
PS50+)
        72      IRQ10   reserved  (AT,XT286,PS50+)
        73      IRQ11   reserved  (AT,XT286,PS50+)
        74      IRQ12   mouse interrupt  (PS50+)
        75      IRQ13   numeric coprocessor NMI error  (AT,XT286,
PS50+)
        76      IRQ14   fixed disk controller (AT,XT286,PS50+)
        77      IRQ15   reserved  (AT,XT286,PS50+)
        78-79           unused
        80-85           ROM BASIC
        86-F0   DOS     reserved for BASIC interpreter use
        86      NETBIOS NETBIOS relocated INT 18
        E0      CPM     CP/M 86 function calls
        F1-FF           reserved by IBM
        FE-FF           may be destroyed by return from protected
                        mode using VDISK on 286 machines (Apr 86,
DDJ)


        - Intel defined 0 through 20h for use for internal CPU;  IBM
          redefined interrupts 0 through 1Fh for its own use, hence
          the duplicate definitions in the tables
        - all interrupts except the internal CPU exceptions push the
          flags and the CS:IP of the next instruction onto the
stack.
          CPU exception interrupts are similar but push the CS:IP
```

of the
causal instruction. 8086/88 divide exceptions are different,
they return to the instruction following the division
- interrupts are disabled upon entry into any interrupt routine and
should be enabled by the user or by an IRET
- in DOS 3.2+ hardware IRQ interrupts are re-vectored through DOS
to provide standard stack frames

INT 10,FF - Update DESQView/TopView Virtual Screen Regen Buffer

```
        AH = FF
        CX = number of characters changed
        ES:DI = pointer to first character in buffer to change,  ES
is
                set to segment returned by INT 10,FE


        returns nothing


        - the physical screen does not get updated until INT 10,FF
is
          called in TopView
        - it is not necessary to make this call under DESQView
since it
          handles updates automatically
        - calling this function under DESQView will cancel the
automatic
          update mode
```

INT 10,FE - Get DESQView/TopView Virtual Screen Regen Buffer

```
        AH = FE
        ES:DI = set to sentinel value (test for INT 10,FE supported)


        returns:
        ES:DI = address of DESQView/TopView video buffer, DI will
always
                be zero


        - on return ES:DI should be tested against the original
value;
          the value will change if this function is supported
(DESQView
          or TopView loaded), otherwise it will remain unchanged
        - if ES:DI changes this address can be used as the video
screen
          regen buffer
```

INT 13,1A - Format Unit (PS/2 model 50+)

```
        AH = 1Ah
        AL = defect table count
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)
        ES:BX = far pointer to defect table
        CL = modifier bits

                |7|6|5|4|3|2|1|0|  Format Unit Modifier Bits
                 | | | | | | | `---- 1=ignore primary defect map,
0=use map
                 | | | | | | `----- 1=ignore secondary defect map,
0=use map
                 | | | | | `------ 1=update secondary defect map,
0=don't
                 | | | | `------- 1=do extended surface analysis,
0=don't
                 | | | `-------- 1=periodic interrupt status on,
0=off
                 `-------------- reserved

        returns nothing

        - only the disk number is checked for validity
        - see   INT 13,STATUS
```

INT 13,1 - Disk Status

AH = 01

on return:
AL = status:

Status in AL

00	no error
01	bad command passed to driver
02	address mark not found or bad sector
03	diskette write protect error
04	sector not found
05	fixed disk reset failed
06	diskette changed or removed
07	bad fixed disk parameter table
08	DMA overrun
09	DMA access across 64k boundary
0A	bad fixed disk sector flag
0B	bad fixed disk cylinder
0C	unsupported track/invalid media
0D	invalid number of sectors on fixed disk format
0E	fixed disk controlled data address mark detected
0F	fixed disk DMA arbitration level out of range
10	ECC/CRC error on disk read
11	recoverable fixed disk data error, data fixed by ECC
20	controller error (NEC for floppies)
40	seek failure
80	time out, drive not ready
AA	fixed disk drive not ready
BB	fixed disk undefined error
CC	fixed disk write fault on selected drive
E0	fixed disk status error/Error reg = 0
FF	sense operation failed

- codes represent controller status after last disk operation
- returns the status byte located at 40:41 in the BIOS Data Area

INT 14,5 - Extended Communication Port Control (PS/2 & later systems)

```
AH = 05
AL = read or write modem control register
   = 00 read modem control register
   = 01 read modem control register
DX = zero based serial port number (0-1) (0-3 for AT)

BL = modem control register

        |7|6|5|4|3|2|1|0|       BL
         | | | | | | | `---- 1 = activate data terminal
ready
         | | | | | | `----- 1 = activate request to send
         | | | | | `------ OUT1
         | | | | `------- OUT2
         | | | `-------- 0 = normal, 1 = loopback test
         `-------------- reserved, forced to zero

on return:
AH = port status
AL = modem status
BL = UART modem control register

- for status bits see INT 14,STATUS
```

UART - Universal Asynchronous Receiver/ Transmitter

Port	Description
3F8/2F8	Transmit/Receive Buffer (read/write)
	Baud Rate Divisor LSB if bit 7 of LCR is set (read/write)
3F9/2F9	IER - Interrupt Enable Register (read/write)
	Baud Rate Divisor MSB if bit 7 of LCR is set (read/write)
3FA/2FA	IIR - Interrupt Identification Register (read only)
	FCR - 16550 FIFO Control Register (write only)
3FB/2FB	LCR - Line Control Register (read/write)
3FC/2FC	MCR - Modem Control Register (read/write)
3FD/2FD	LSR - Line Status Register (read only)
3FE/2FE	MSR - Modem Status Register (read only)
3FF/2FF	Scratch Pad Register (read/write)

Detailed UART Description

Port 3F8 - Transmit/Receive Buffer (read/write)
Baud Rate Divisor LSB if bit 7 of LCR is set (read/write)

Port 3F9 - Interrupt Enable Register - IER (read/write)
Baud Rate Divisor MSB if bit 7 of LCR is set (read/write)

```
|7|6|5|4|3|2|1|0|  2F9, 3F9: Interrupt Enable Register
 | | | | | | | `---- 1 = enable data available int (and 16550 Timeout)
 | | | | | | `----- 1 = enable THRE interrupt
 | | | | | `------ 1 = enable lines status interrupt
 | | | | `------- 1 = enable modem-status-change interrupt
 `-------------- reserved (zero)
```

- 16550 will interrupt if data exists in the FIFO and isn't read within the time it takes to receive four bytes or if no data is received within the time it takes to receive four bytes.

Baud Rate Divisor Table

Baud Rate	Baud Rate Divisor	Baud Rate	Baud Rate Divisor
50	900h	2400	30h
110	417h	3600	20h
150	300h	4800	18h
300	180h	7200	10h
600	C0h	9600	0Ch
1200	60h	19200	06h
1800	40h	38400	03h
2000	3Ah	115200	01h

- Baud rate divisors can be calculated by taking the oscillating frequency (1,843,200) and dividing by the quantity of the desired baud rate times the UART clocking factor (16). Use the following formula:

divisor = 1843200 / (BaudRate * 16);

Port 3FA - Interrupt Identification Register - IIR (read only)

```
|7|6|5|4|3|2|1|0|  2FA, 3FA Interrupt ID Register
 | | | | | | | `---- 1 = no int. pending, 0=int. pending
```

```
        | | | | | `------- Interrupt Id bits (see below)
        | | | | `-------- 16550  1 = timeout int. pending, 0 for
8250/16450
        | | `------------ reserved (zero)
        `--------------- 16550  set to 1 if FIFO queues are enabled
```

Bits 21	Meaning	Priority	To reset
00	modem-status-change	lowest	read MSR
01	transmit-register-empty	low	read IIR / write THR
10	data-available	high	read rec buffer reg
11	line-status	highest	read LSR

- interrupt pending flag uses reverse logic, 0 = pending, 1 = none
- interrupt will occur if any of the line status bits are set
- THRE bit is set when THRE register is emptied into the TSR

Port 3FA - 16550 FIFO Control Register - FCR (write only)

```
        |7|6|5|4|3|2|1|0|  2FA, 3FA  FIFO Control Register
        | | | | | | | `---- 1 = enable clear XMIT and RCVR FIFO
queues
        | | | | | | `----- 1 = clear RCVR FIFO
        | | | | | `------ 1 = clear XMIT FIFO
        | | | | `------- 1 = change RXRDY & TXRDY pins from mode 0
to mode 1
        | | `---------- reserved (zero)
        `------------- trigger level for RCVR FIFO interrupt
```

Bits 76	RCVR FIFO Trigger Level
00	1 byte
01	4 bytes
10	8 bytes
11	14 bytes

- Bit 0 must be set in order to write to other FCR bits
- Bit 1 when set to 1 the RCVR FIFO is cleared and this bit is reset.
  The receiver shift register is not cleared.
- Bit 2 when set to 1 the XMIT FIFO is cleared and this bit is reset.
  The transmit shift register is not cleared.

Port 3FB - Line Control Register - LCR (read/ write)

```
|7|6|5|4|3|2|1|0|  2FB, 3FB  Line Control Register
 | | | | | | `------ word length select bits (see below)
 | | | | | `------- 0 = 1 stop bit, 1 = 1.5 or 2  (see note)
 | | | | `-------- 0 = no parity, 1 = parity (PEN)
 | | | `--------- 0 = odd parity, 1 = even (EPS)
 | | `---------- 0 = parity disabled, 1 = enabled
 | `----------- 0 = turn break off, 1 = force spacing break state
 `------------ 1 = baud rate divisor (DLAB); 0 = RBR, THR or IER
```

Bits
10 Word length bits
00 = 5 bits per character
01 = 6 bits per character
10 = 7 bits per character
11 = 8 bits per character

- stop bits = 1.5 for 5 bit words or 2 for 6, 7 or 8 bit words
- bit 7 changes the mode of registers 3F8 and 3F9. If set these
  registers become the LSB and MSB of the baud rate divisor.
  Otherwise 3F8 is the Transmit/Receive Buffer Register and 3F9 is
  the Interrupt Enable Register.

Port 3FC - Modem Control Register - MCR (read/

write)

```
       |7|6|5|4|3|2|1|0|  2FC, 3FC  Modem Control Register
        | | | | | | | `---- 1 = activate DTR
        | | | | | | `----- 1 = activate RTS
        | | | | | `------ OUT1
        | | | | `------- OUT2
        | | | `-------- 0 = normal, 1 = loop back test
        `-------------- reserved (zero)
```

- If bit 4 is set, data from the Transmit Shift Register is received in the Receiver Shift Register. The SOUT line is set to logic high, the SIN line and control lines are disconnected. CTS, DSR, RI and CD inputs are disconnected. DTR, RTS, OUT1 and OUT2 are then connected internally.

Port 3FD - Line Status Register - LSR (read only)

```
       |7|6|5|4|3|2|1|0|  2FD, 3FD Line Status Register
        | | | | | | | `---- 1 = data ready
        | | | | | | `----- 1 = overrun error (OE)
        | | | | | `------ 1 = parity error (PE)
        | | | | `------- 1 = framing error (FE)
        | | | `-------- 1 = break interrupt  (BI)
        | | `--------- 1 = transmitter holding register empty (THRE)
        | `---------- 1 = transmitter shift register empty (TSRE)
        `----------- 1 = 16550 PE/FE/Break in FIFO queue, 0 for 8250 & 16450
```

- Bit 0 is set when a byte is placed in the Receiver Buffer Register and cleared when the byte is read by the CPU (or when the CPU clears the FIFO for the 16550). Results in Receive Data Available

Interrupts if enabled.
- Bits 1-4 indicate errors and result in Line Status Interrupts if enabled.
- Bit 1 is set when a second byte is received before the byte in the Receiver Buffer Register is read by the CPU (the 16550 in FIFO mode sets this bit when the queue is full and the byte in the Receiver Shift Register hasn't been moved into the queue). This bit is reset when the CPU reads the LSR
- Bit 2 is set whenever a byte is received that doesn't match the requested parity. Reset upon reading the LSR. (The 16550 maintains parity information with each byte and sets bit 2 only when the byte is at the top of the FIFO queue.)
- Bit 3 is set when a character is received without proper stop bits. Upon detecting a framing error the UART attempts to resynchronize. Reset by reading the LSR. (The 16550 maintains this information with each byte and sets bit 3 only when the byte is at the top of the FIFO queue.)
- Bit 4 is set when a break condition is sensed (when space is detected for longer than 1 fullword). A zero byte is placed in the Receiver Buffer Register (or 16550 FIFO). Reset by reading the LSR. (The 16550 maintains this information with each byte and sets bit 4 only when the byte is at the top of the FIFO queue.)
- Bit 5 is set when the Transmit Holding Register shifts a byte into the Transmit Shift Register (or XMIT FIFO queue is empty for 16550) and is cleared when a byte is written to the THR (or the XMIT FIFO). Results in Transmit Holding Register Empty interrupts

if enabled.
- Bit 6 is set when both the Transmitter Holding Register and the
Transmitter Shift Register are empty. On the 16550, when the XMIT
FIFO and Transmitter Shift Register are empty.
- Bit 7 is 16550 specific and indicates there is a byte in the FIFO
queue that was received with a Parity, Framing or Break error.

Port 3FE - Modem Status Register - MSR (read only)

```
|7|6|5|4|3|2|1|0|  2FE, 3FE Modem Status Register
 | | | | | | | `---- 1 = DCTS  Delta CTS  (CTS changed)
 | | | | | | `----- 1 = DDSR  Delta DSR  (DSR changed)
 | | | | | `------ 1 = RI ring indicator changed
 | | | | `------- 1 = DDCD  Delta Data Carrier Detect (DCD changed)
 | | | `-------- 1 = CTS
 | | `--------- 1 = DSR
 | `---------- 1 = ring indicator (RI)
 `----------- 1 = receive line signal detect
```

- Bits 0-3 are reset when the CPU reads the MSR
- Bit 4 is the Modem Control Register RTS during loopback test
- Bit 5 is the Modem Control Register DTR during loopback test
- Bit 6 is the Modem Control Register OUT1 during loopback test
- Bit 7 is the Modem Control Register OUT2 during loopback test

Port 3FF/2FF - Scratch Pad Register (read/write)

Programming considerations:

- 8250's, 16450's are essentially identical to program
- 16550's is pin and software compatible with the 16450 but has an
  internal FIFO queue that may be enabled/disabled by software
- PCs are capable of 38.4Kb, while AT's are capable of 115.2Kb
- receiver checks only the first stop bit of each character regardless
  of the number of stop bits specified
- Older 8250 and 16450 UARTs may lose THRE interrupt if the THRE and
  Receive Data (RD) or the Line Status (LS) interrupts occur
  simultaneously during a full duplex transmission. RD and LS have
  higher priority than THRE which causes the lower priority interrupt
  to be lost. The following are 3 methods used to avoid this problem:

  1. Disable/re-enable THRE interrupt via the IER after processing
     Receive Data & Line Status interrupts.
  2. While inside the RD and LS interrupt routines check the LSR
     THRE bit and set a flag that a THRE interrupt was waiting.
  3. Poll the LSR THRE bit instead of using the IRR.

- data loss can occur without overrun or framing errors if the
  interrupts are serviced too slowly
- reserved bits are usually set to zero. Code should NOT rely on
  this being the case since future enhancement may use these bits
- see INT TABLE or IRQ for interrupt assignments
- see PORTS for COMx port assignment (3F8,2F8,3E8,2E8,3220...)

RS232 Communication Configuration

IBM PC		IBM AT	
Pin	**25-Pin Signal (DTE)**	**Pin**	**9-Pin Signal (DTE)**
1	Chassis Ground (GND)	1	Carrier Detect (CD)
2	Transmit Data (TD)	2	Receive Data (RD)
3	Receive Data (RD)	3	Transmit Data (TD)
4	Request to Send (RTS)	4	Data Terminal Ready (DTR)
5	Clear to Send (CTS)	5	Signal Ground (SG)
6	Data Set Ready (DSR)	6	Data Set Ready (DSR)
7	Signal Ground (SG)	7	Request to Send (RTS)
8	Carrier Detect (CD)	8	Clear to Send (CTS)
9-19	(not used)	9	Ring Indicator (RI)
20	Data Terminal Ready (DTR)		
22	Ring Indicator (RI)		

There are two general cable configurations used with the RS-232C
Communications Standard:

Data Terminal Equipment (DTE): IBM PC's, printers, plotters, etc
Data Communication Equipment (DCE): modems, multiplexors, etc

DCE to DTE requires all lines run straight through
DTE to DTE usually requires swapping of the following lines

RD and TD		RD and TD
RTS and CTS	or	RTS,CTS and DCD
DTR and DSR		DCD and RTS,CTS

Signal Functions

GND	Ground	protective safety ground
TD	Transmit Data	DTE output data
RD	Receive Data	DTE input data
RTS	Request To Send	DTE output, DTE would like to transmit
CTS	Clear To Send	DTE input, DCE is ready to transmit

	DSR	Data Set Ready	DTE input, DCE is ready to communicate
	SG	Signal Ground	provides a Zero reference voltage
	DCD	Data Carrier Detect	DTE input, data link established, also known as Receive Line Signal Detect (RLSD)
	DTR	Data Terminal Ready	DTE output, device ready
	RI	Ring Indicator	DTE input, announces incoming call

- RTS/CTS is used for half duplex line turn around
- in half duplex DCD is asserted only by the receiving device
- full duplex modems tie CTS & DCD together (no CTS/RTS handshaking)
- most modems require DTR to be present to respond to commands
- maximum voltages are between -15 volts and +15 volts
- binary outputs are between +5 to +15 volts and -5 to -15 volts
- binary inputs are between +3 to +15 volts and -3 to -15 volts
- input voltages between -3 to +3 are undefined while output voltages between -5 and +5 are undefined
- positive voltages indicate ON or SPACE, negative voltages indicate OFF or MARK

INT 14 - BIOS Asynchronous Communications Services

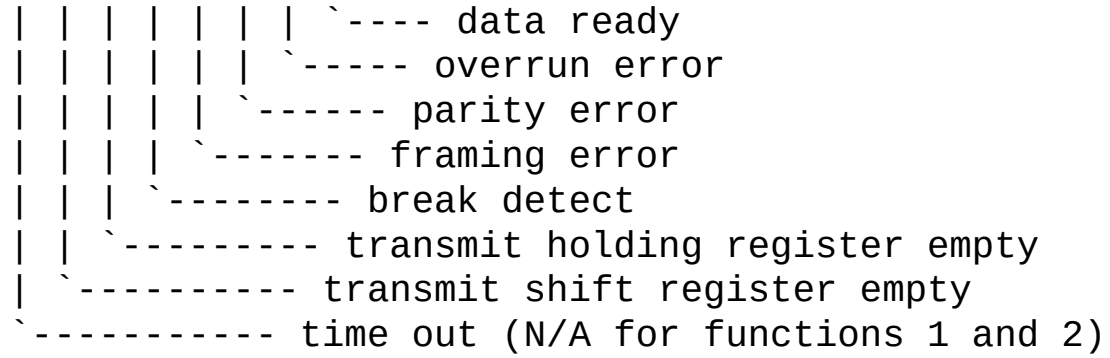
For more information, see the following topics:

INT 14,0 Initialize serial port parameters
INT 14,1 Send character in AL
INT 14,2 Receive character in AL
INT 14,3 Get Serial port status
INT 14,4 Extended initialize (PS/2)
INT 14,5 Extended communication port control (PS/2)

- all functions have:
  AH = function number
  AL = character to send or receive
  DX = zero based RS232 card number
- all registers are preserved except AX
- these functions use hardware flow control
- used by DOS MODE to redirect LPTx output to a serial device
- see individual functions for more data

BIOS Asynchronous Communications Services (Status)

```
|7|6|5|4|3|2|1|0|  AL  modem status
 | | | | | | | `---- clear to send status changed
 | | | | | | `----- data set ready status changed
 | | | | | `------ trailing edge ring indicator
 | | | | `------- receive line signal changed
 | | | `-------- clear to send
 | | `--------- data set ready
 | `---------- ring indicator
 `----------- receive line signal detected

|7|6|5|4|3|2|1|0|  AH  port status
```


| | | | | | `---- data ready
| | | | | `----- overrun error
| | | | `------ parity error
| | | `------- framing error
| | `-------- break detect
| `--------- transmit holding register empty
`---------- transmit shift register empty
`----------- time out (N/A for functions 1 and 2)

INT 14,0 - Initialize Communications Port Parameters

```
AH = 00
AL = parms for initialization (see tables below)
DX = zero based serial port number (0-1) (0-3 for AT)

|7|6|5|4|3|2|1|0|  AL                          Parity (bits 4 &
3)
 | | | | | | `------ word length bits          00 = none
 | | | | | `------- stop bits flag             01 = odd
 | | | `---------- parity bits                  10 = none
 `---------------- baud rate bits                11 = even
```

Word length (bits 1 & 0)

```
   10 = 7 bits
   11 = 8 bits
```

Stop bit count (bit 2)

```
  0 = 1 stop bit
  1 = 2 stop bits
```

Baud rate (bits 7, 6 & 5)

```
000 = 110 baud      100 = 1200 baud
001 = 150 baud      101 = 2400 baud
010 = 300 baud      110 = 4800 baud
011 = 600 baud      111 = 9600 baud (4800 on PCjr)


on return:
AH = port status
AL = modem status
```

- for status bits see INT 14,STATUS

INT 14,1 - Send Character to Communications Port

```
AH = 01
AL = character to send
DX = zero based serial port number (0-1) (0-3 for AT)


on return:
AH = port status  (see INT 14,STATUS)
     bit 7=0 indicates success
     bit 7=1 indicates error, bits 0-6 indicate cause


- INT 14,3 should be used to determine the actual cause
  of the error since the time-out bit of the status register
  is always set during an error on this call
- uses hardware flow control
```

INT 14,3 - Get Serial Port Status

```
AH = 03
DX = zero based serial port number (0-1) (0-3 for AT)


on return:
AH = port status
AL = modem status
```

- for status bits see INT 14,STATUS
- the status check performs a poll of the port and does not perform character I/O

INT 14,2 - Receive Character from Communications Port

```
AH = 02
DX = zero based serial port number (0-1) (0-3 for AT)


on return:
AH = port status  (see INT 14,STATUS)
     bit 7 = 0 if successful
     bit 7 = 1 if call failed
AL = character received if call was success
```

- INT 14,3 should be used to determine the actual cause of the error since the time-out bit of the status register is always set during an error on this call
- uses hardware flow control

INT 14,4 - Serial Port Extended Initialization (PS/2 & later systems)

```
AH = 04
AL = break setting
BH = parity setting
BL = stop bit setting
CH = word length setting
CL = baud rate setting
DX = zero based serial port number (0-3)
```

--- Break Setting ---

```
1 = break
0 = no break
```

----- Parity Setting -----

```
0 = no parity
1 = odd parity
2 = even parity
3 = stick parity odd
4 = stick parity even
```

--- Stop Bit Setting ---

```
0 = one stop bit
1 = 2 stop bits
    1« if register CH=0
    (5 bit word length)
```

-- Word Length Setting --

```
0 = 5 bits
1 = 6 bits
2 = 7 bits
3 = 8 bits
```

---- Baud rate Setting ----

```
0 = 110      5 = 2400
1 = 150      6 = 4800
2 = 300      7 = 9600
3 = 600      8 = 19200
4 = 1200
```

```
on return:
AH = port status
AL = modem status
```

- for status bits see INT 14,STATUS

INT 15,C4 - Programmable Option Select (PS/2)

```
        AH = C4
        AL = 0  get Programmable Option Select adapter register
address
           = 1  enable slot for setup
           = 2  adapter enable


        on return:
        CF = 0 if successful
           = 1 if error
        AH = when CF set, 80h for PC & PCjr, 86h for XT, AT & 30
        BL = slot number (function AL=1)
        DL = POS adapter register address (function AL=0)


        - allows access to PS/2 Programmable Option Select registers
        - available on 80286 and 80386 PS/2's
```

INT 16,12 - Extended Get Keyboard Status (AT+)

```
AH = 12h

on return:
AH = BIOS keyboard flags (BIOS Data Area location 40:18)

        |7|6|5|4|3|2|1|0|  AH
         | | | | | | | `---- left CTRL key depressed
         | | | | | | `----- left ALT key depressed
         | | | | | `------ right CTRL key pressed
         | | | | `------- right ALT key depressed
         | | | `-------- scroll-lock depressed
         | | `--------- num-lock key depressed
         | `---------- caps-lock key depressed
         `----------- system request key depressed

AL = BIOS keyboard flags (BIOS Data Area location 40:17)

        |7|6|5|4|3|2|1|0|  AL
         | | | | | | | `---- right shift key depressed
         | | | | | | `----- left shift key depressed
         | | | | | `------ CTRL key depressed
         | | | | `------- ALT key depressed
         | | | `-------- scroll-lock is active
         | | `--------- num-lock is active
         | `---------- caps-lock is active
         `----------- insert is active


- available on AT and PS/2 machines with extended keyboard support
- similar to INT 16,2

- see   KB FLAGS
```

INT 16,2 - Read Keyboard Flags

```
AH = 02


on return:
AL = BIOS keyboard flags (located in BIOS Data Area 40:17)

        |7|6|5|4|3|2|1|0|  AL or BIOS Data Area 40:17
         | | | | | | | `---- right shift key depressed
         | | | | | | `----- left shift key depressed
         | | | | | `------ CTRL key depressed
         | | | | `------- ALT key depressed
         | | | `-------- scroll-lock is active
         | | `--------- num-lock is active
         | `---------- caps-lock is active
         `----------- insert is active
```

INT 17,2 - Read Printer Port Status

```
AH = 02
DX = printer port to be used (0-2)


on return:
AH = status:

        |7|6|5|4|3|2|1|0|  Printer status bits
         | | | | | | | `---- time out
         | | | | | | `------- unused
         | | | | | `-------- I/O error
         | | | | `---------- selected
         | | | `----------- out of paper
         | | `------------ acknowledge
         | `------------- not busy


- returns status of specified printer port
```

INT 19 - Bootstrap Loader

```
DL = physical drive where boot sector is located


no output


- track 0, sector 1 is loaded into address 0:7C00
  and control is transferred there
- not a preferred method for rebooting by applications.  A
  better method is to set the word at location 40:72 to
1234h
  and jump to location FFFF:0 in ROM
- memory is not cleared when rebooted through this interrupt

- see   WARM BOOT
```

Warm Boot / System Bootstrap

- Ctrl-Alt-Delete results in a warm boot but only after part of the
  POST sequence has executed. See COLD BOOT for more information.
- ROM BIOS bootstrap loader (INT 19) attempts Boot Record load
- Drive A: is searched for a Boot Record, if no Boot Record is found
  the active partition of the first drive is searched. If found, it
  is loaded into memory at 7C00:0 and given control. If still not
  found Cassette BASIC is loaded (IBM only) or a prompt for a system
  diskette is displayed.
- the boot record code checks that IBMBIO.COM and IBMDOS.COM (IO.SYS
  and DOS.SYS for MSDOS) exist and are the first two files. These
  files must be contiguous and the first two directory entries.
- boot record loads IBMBIO.COM
- IBMBIO.COM initialization code loads IBMDOS.COM
  / determines equipment status
  / resets disk system
  / initializes attached devices
  / loads installable device drivers
  / sets DOS relative interrupt vectors
  / relocates IBMDOS.COM downward and begins its execution
- IBMDOS.COM initializes its working tables
  / initializes DOS interrupt vectors for interrupts 20h through 27h
  / builds PSP for COMMAND.COM at lowest possible segment
  / interrupt vectors for INTs 0Fh through 3Fh are initialized
    (for DOS 3.1+)
  / loads COMMAND.COM (or specified command interpreter) using the
    EXEC call.
- AUTOEXEC.BAT is then loaded and executed

- see BOOT RECORD COLD BOOT

POST - Cold Boot / Power On Self Test Activities

- power supply starts Clock Generator (8284) with Power Good signal on BUS
- CPU reset line is pulsed resetting CPU
- DS, ES, and SS are cleared to zero

Cold and Warm Boot both execute the following sequence

- CS:IP are set to FFFF:0000 (address of ROM POST code)
- jump to CS:IP (execute POST, Power On Self test)
- interrupts are disabled
- CPU flags are set, read/write/read test of CPU registers
- checksum test of ROM BIOS
- Initialize DMA (verify/init 8237 timer, begin DMA RAM refresh)
- save reset flag then read/write test the first 32K of memory
- Initialize the Programmable Interrupt Controller (8259) and set 8 major BIOS interrupt vectors (interrupts 10h-17h)
- determine and set configuration information
- initialize/test CRT controller & test video memory (unless 1234h found in reset word)
- test 8259 Programmable Interrupt Controller
- test Programmable Interrupt Timer (8253)
- reset/enable keyboard, verify scan code (AAh), clear keyboard, check for stuck keys, setup interrupt vector lookup table
- hardware interrupt vectors are set
- test for expansion box, test additional RAM
- read/write memory above 32K (unless 1234h found in reset word)
- addresses C800:0 through F400:0 are scanned in 2Kb blocks in search of valid ROM. If found, a far call to byte 3 of the ROM is executed.
- test ROM cassette BASIC (checksum test)
- test for installed diskette drives & FDC recalibration & seek
- test printer and RS-232 ports. store printer port addresses

at 400h and RS-232 port addresses at 408h. store printer time-out values at 478h and Serial time-out values at 47Ch.
- NMI interrupts are enabled
- perform INT 19 (bootstrap loader), pass control to boot record or cassette BASIC if no bootable disk found
- WARM BOOT procedure is now executed

- see DIAGNOSTIC CODES

INT 33 - Mouse Function Calls

For more information see the following topics:

INT 33,0	Mouse Reset/Get Mouse Installed Flag
INT 33,1	Show Mouse Cursor
INT 33,2	Hide Mouse Cursor
INT 33,3	Get Mouse Position and Button Status
INT 33,4	Set Mouse Cursor Position
INT 33,5	Get Mouse Button Press Information
INT 33,6	Get Mouse Button Release Information
INT 33,7	Set Mouse Horizontal Min/Max Position
INT 33,8	Set Mouse Vertical Min/Max Position
INT 33,9	Set Mouse Graphics Cursor
INT 33,A	Set Mouse Text Cursor
INT 33,B	Read Mouse Motion Counters
INT 33,C	Set Mouse User Defined Subroutine and Input Mask
INT 33,D	Mouse Light Pen Emulation On
INT 33,E	Mouse Light Pen Emulation Off
INT 33,F	Set Mouse Mickey Pixel Ratio
INT 33,10	Mouse Conditional OFF
INT 33,13	Set Mouse Double Speed Threshold
INT 33,14	Swap interrupt subroutines
INT 33,15	Get mouse driver state and memory requirements
INT 33,16	Save mouse driver state
INT 33,17	Restore mouse driver state
INT 33,18	Set alternate subroutine call mask and address
INT 33,19	Get user alternate interrupt address
INT 33,1A	Set mouse sensitivity
INT 33,1B	Get mouse sensitivity
INT 33,1C	Set mouse interrupt rate (InPort only)
INT 33,1D	Set mouse CRT page
INT 33,1E	Get mouse CRT page
INT 33,1F	Disable mouse driver
INT 33,20	Enable mouse driver
INT 33,21	Reset mouse software
INT 33,22	Set language for messages
INT 33,23	Get language number
INT 33,24	Get driver version, mouse type & IRQ number

- function is specified in AX
- a mickey is 1/200 inches
- for additional information see your vendor documentation
- function number occupies all of AX rather than AH

Mouse functions can be broken down into the following classes:

Mouse Driver Control / Feedback Functions

INT 33,0 Mouse Reset/Get Mouse Installed Flag
INT 33,15 Get Mouse Driver State and Memory Requirements
INT 33,16 Save Mouse Driver State
INT 33,17 Restore Mouse Driver State
INT 33,1C Set Mouse Interrupt Rate (InPort only)
INT 33,1F Disable Mouse Driver
INT 33,20 Enable Mouse Driver
INT 33,21 Reset Mouse Software
INT 33,24 Get Driver Version, Mouse Type & IRQ Number

Mouse Cursor Control Functions

INT 33,1 Show Mouse Cursor
INT 33,2 Hide Mouse Cursor
INT 33,4 Set Mouse Cursor Position
INT 33,7 Set Mouse Horizontal Min/Max Position
INT 33,8 Set Mouse Vertical Min/Max Position
INT 33,9 Set Mouse Graphics Cursor
INT 33,A Set Mouse Text Cursor
INT 33,F Set Mouse Mickey Pixel Ratio
INT 33,10 Mouse Conditional OFF
INT 33,13 Set Mouse Double Speed Threshold
INT 33,1A Set Mouse Sensitivity
INT 33,1B Get Mouse Sensitivity

Mouse Button and Position Feedback Functions

INT 33,3 Get Mouse Position and Button Status
INT 33,5 Get Mouse Button Press Information
INT 33,6 Get Mouse Button Release Information

INT 33,B Read Mouse Motion Counters

Video Control and Feedback Functions

INT 33,1D Set Mouse CRT Page
INT 33,1E Get Mouse CRT Page

Mouse Interrupt Setup Functions

INT 33,C Set Mouse User Defined Subroutine and Input Mask
INT 33,14 Swap Interrupt Subroutines

Alternate Mouse Interrupt Setup Functions

INT 33,18 Set Alternate Subroutine Call Mask and Address
INT 33,19 Get User Alternate Interrupt Address

Light Pen Emulation Functions

INT 33,D Mouse Light Pen Emulation On
INT 33,E Mouse Light Pen Emulation Off

International Language Support Functions

INT 33,22 Set Language for Messages
INT 33,23 Get Language Number

INT 33,0 - Mouse Reset/Get Mouse Installed Flag

```
AX = 00


on return:
AX = 0000  mouse driver not installed
     FFFF  mouse driver installed
BX = number of buttons


- resets mouse to default driver values:

  .  mouse is positioned to screen center
  .  mouse cursor is reset and hidden
  .  no interrupts are enabled (mask = 0)
  .  double speed threshold set to 64 mickeys per second
  .  horizontal mickey to pixel ratio (8 to 8)
  .  vertical mickey to pixel ratio (16 to 8)
  .  max width and height are set to maximum for video mode
```

INT 33,1F - Disable Mouse Driver

```
        AX = 1F


        on return:
        AX = 001F if successful
             FFFF if error
        ES:BX = previous INT 33 vector


        - restores vectors for INT 10 and INT 71 (8088/86) or INT
74 (286+)
        - INT 33 interrupt vector (software) is not affected
        - use of the vector returned in ES:BX to restore the
previous INT 33
          vector can cause problems since it contains the value of
INT 33
          before the driver was installed; any other hooks into INT
33 set
          after driver installation will not receive service
```

INT 33,24 - Get Driver Version, Mouse Type & IRQ Number

```
AX = 24h


on return:
BH = major version (see below)
BL = minor version (see below)
CH = mouse type:
   = 1  bus mouse
   = 2  serial mouse
   = 3  InPort mouse
   = 4  PS/2  mouse
   = 5  Hewlett Packard mouse
CL = IRQ number:
   = 0  PS/2
   = 2  IRQ 2
   = 5  IRQ 5
   = 7  IRQ 7

- version 6.1 would be represented as BH = 06h, BL = 10h
```

INT 33,3 - Get Mouse Position and Button Status

```
        AX = 03


        on return:
        CX = horizontal (X) position  (0..639)
        DX = vertical (Y) position  (0..199)
        BX = button status:

                |F-8|7|6|5|4|3|2|1|0|  Button Status
                  |  | | | | | | | `---- left button (1 = pressed)
                  |  | | | | | | `----- right button (1 = pressed)
                  `------------------- unused


        - values returned in CX, DX are the same regardless of
video mode
```

INT 33,4 - Set Mouse Cursor Position

```
AX = 4
CX = horizontal position
DX = vertical position


returns nothing


- default cursor position is at the screen center
- the position must be within the range of the current
video mode
- the position may be rounded to fit screen mode resolution
```

INT 33,5 - Get Mouse Button Press Information

```
        AX = 5
        BX = 0  left button
             1  right button


        on return:
        BX = count of button presses (0-32767), set to zero after
call
        CX = horizontal position at last press
        DX = vertical position at last press
        AX = status:

                |F-8|7|6|5|4|3|2|1|0|  Button Status
                  |  | | | | | | | `---- left button (1 = pressed)
                  |  | | | | | | `----- right button (1 = pressed)
                  `------------------- unused
```

INT 33,6 - Get Mouse Button Release Information

```
        AX = 6
        BX = 0  left button
             1  right button


        on return:
        BX = count of button releases (0-32767), set to zero after
call
        CX = horizontal position at last release
        DX = vertical position at last release
        AX = status

                |F-8|7|6|5|4|3|2|1|0|  Button status
                  |  | | | | | | | `---- left button (1 = pressed)
                  |  | | | | | | `----- right button (1 = pressed)
                  `------------------- unused
```

INT 33,7 - Set Mouse Horizontal Min/Max Position

```
AX = 7
CX = minimum horizontal position
DX = maximum horizontal position


returns nothing


- restricts mouse horizontal movement to window
- if min value is greater than max value they are swapped
```

INT 33,8 - Set Mouse Vertical Min/Max Position

```
AX = 8
CX = minimum vertical position
DX = maximum vertical position


returns nothing


- restricts mouse vertical movement to window
- if min value is greater than max value they are swapped
```

INT 33,9 - Set Mouse Graphics Cursor

```
AX = 9
BX = horizontal hot spot (-16 to 16)
CX = vertical hot spot (-16 to 16)
ES:DX = pointer to screen and cursor masks (16 byte bitmap)


returns nothing


- screen mask is AND'ed to screen Cursor Mask is XOR'ed
- bytes 0-7 form the screen mask bitmap
- bytes 8-F form the cursor mask bitmap
```

INT 33,A - Set Mouse Text Cursor

```
AX = 0A
BX = 00  software cursor
     01  hardware cursor
CX = start of screen mask or hardware cursor scan line
DX = end of screen mask or hardware cursor scan line


returns nothing
```

INT 33,B - Read Mouse Motion Counters

```
        AX = 0B


        on return:
        CX = horizontal mickey count (-32768 to 32767)
        DX = vertical mickey count (-32768 to 32767)


        - count values are 1/200 inch intervals (1/200 in. = 1
mickey)
```

INT 33,C - Set Mouse User Defined Subroutine and Input Mask

```
AX = 0C
ES:DX = far pointer to user interrupt
CX = user interrupt mask:

   |F-5|4|3|2|1|0| user interrupt mask in CX
     |  | | | | `--- cursor position changed
     |  | | | `---- left button pressed
     |  | | `----- left button released
     |  | `------ right button pressed
     |  `------- right button released
     `--------- unused

returns nothing


- routine at ES:DX is called if an event occurs and the
  corresponding bit specified in user mask is set
- routine at ES:DX receives parameters in the following
  registers:

  AX = condition mask causing call
  CX = horizontal cursor position
  DX = vertical cursor position
  DI = horizontal counts
  SI = vertical counts
  DS = mouse driver data segment
  BX = button state:

     |F-2|1|0|
       |  | `--- left button (1 = pressed)
       |  `---- right button (1 = pressed)
       `------ unused

- initial call mask and user routine should be restore on
exit
  from user program
- user program may need to set DS to it's own segment
- see   INT 33,14
```

INT 33,14 - Swap Interrupt Subroutines

```
AX = 14h
ES:DX = far pointer to user routine
CX = user interrupt mask:

        |F-8|7|6|5|4|3|2|1|0| user interrupt mask in CX
          |  | | | | | | | `--- cursor position changed
          |  | | | | | | `---- left button pressed
          |  | | | | | `----- left button released
          |  | | | | `------ right button pressed
          |  | | | `------- right button released
          `--------------- unused

on return:
CX = previous user interrupt mask
ES:DX = far pointer to previous user interrupt

- routine at ES:DX is called if an event occurs and the
  corresponding bit specified in user mask is set
- routine at ES:DX receives parameters in the following
  registers:

  AX = condition mask causing call
  CX = horizontal cursor position
  DX = vertical cursor position
  DI = horizontal counts
  SI = vertical counts
  DS = mouse driver data segment
  BX = button state:

     |F-2|1|0|
       |  | `--- left button (1 = pressed)
       |  `---- right button (1 = pressed)
       `------ unused

- initial call mask and user routine should be restore on
exit
  from user program
- user program may need to set DS to it's own segment
- see   INT 33,C
```

INT 33,D - Mouse Light Pen Emulation On

```
AX = 0D


returns nothing


- turns on light pen emulation;  the light pen is considered
  down when both buttons are down;   when both buttons are
  up the pen is considered off screen

- see   INT 33,E
```

INT 33,E - Mouse Light Pen Emulation Off

```
AX = 0E

returns nothing

- disables light pen emulation
- see   INT 33,D
```

INT 33,F - Set Mouse Mickey Pixel Ratio

```
AX = 0F
CX = horizontal ratio (1..32767, default 8)
DX = vertical ratio (1..32767, default 16)

returns nothing

- sets the ratio between physical cursor movement (mickeys) and
  screen coordinate changes
- CX and DX must be unsigned (high bit must be 0)
```

INT 33,10 - Mouse Conditional OFF

```
AX = 10h
CX = upper X screen coordinate
DX = upper Y screen coordinate
SI = lower X screen coordinate
DI = lower Y screen coordinate

returns nothing

- defines screen region for updating in which the mouse is hidden
  if found these coordinates (INT 33,1 must be used to turn cursor
  on again)
- see  INT 33,1
```

INT 33,13 - Set Mouse Double Speed Threshold

```
        AX = 13h
        DX = threshold speed (mickeys per second, default 64)

        returns nothing

        - cursor speed is doubled when the cursor moves across the
screen
          at the threshold speed
```

INT 33,15 - Get Mouse Driver State and Memory Requirements

```
        AX = 15h


        on return
        BX = buffer size need to hold current mouse state


        - used before mouse functions 16h and 17h to determine
memory
          needed to save mouse state before giving up control of
mouse
          to another program
```

INT 33,16 - Save Mouse Driver State

```
AX = 16h
ES:DX = far pointer to mouse state save buffer

returns nothing

- used to save mouse information before relinquishing control
  to another programs mouse handler
- see INT 33,15   INT 33,17
```

INT 33,17 - Restore Mouse Driver State

```
AX = 17h
ES:DX = far pointer to mouse state save buffer


returns nothing


- used to restore mouse information after regaining control
  from another programs mouse handler
- see INT 33,15   INT 33,16
```

INT 33,18 - Set alternate subroutine call mask and address

```
        AX = 18h
        DX = offset to function
        CX = user interrupt mask:

                |F-8|7|6|5|4|3|2|1|0| user interrupt mask in CX
                  |  | | | | | | | `--- alt key pressed during event
                  |  | | | | | | `---- ctrl key pressed during event
                  |  | | | | | `----- shift key pressed during event
                  |  | | | | `------ right button up event
                  |  | | | `------- right button down event
                  |  | | `-------- left button up event
                  |  | `--------- left button down event
                  |  `---------- cursor moved
                  `------------ unused
```

- entire mask is set to zero when INT 33,0 is called
- up to three handlers may be defined with this call
- mask should be cleared before program exit
- when handler is called it receives parameters in the following registers:

```
          AX = condition mask causing call
          CX = horizontal cursor position
          DX = vertical cursor position
          DI = horizontal counts
          SI = vertical counts
          DS = mouse driver data segment
          BX = button state:

             |F-2|1|0|
               |  | `--- left button (1 = pressed)
               |  `---- right button (1 = pressed)
               `------ unused
```

INT 33,19 - Get User Alternate Interrupt Address

```
AX = 19h
CX = user interrupt call mask (see below)


on return:
BX:DX = user interrupt vector
CX = user interrupt call mask or zero if not found

        |F-8|7|6|5|4|3|2|1|0| user interrupt mask in CX
          |  | | | | | | | `--- alt key pressed during event
          |  | | | | | | `---- ctrl key pressed during event
          |  | | | | | `----- shift key pressed during event
          |  | | | | `------ right button up event
          |  | | | `------- right button down event
          |  | | `-------- left button up event
          |  | `--------- left button down event
          |  `---------- cursor moved
          `------------ unused
```

- returns vector to function defined by INT 33,18
- searches the event handlers defined by INT 33,18 for a routine
  with a call mask matching CX

INT 33,1A - Set Mouse Sensitivity

```
AX = 1A
BX = horizontal coordinates per pixel  (ó 100)
CX = vertical coordinates per pixel  (ó 100)
DX = double speed threshold


returns nothing
```

- sets mouse sensitivity by setting the ratio of the mouse coordinates per screen pixel
- provides same results as calls to both INT 33,F and INT 33,13
- these values are not reset by INT 33,0

INT 33,1B - Get Mouse Sensitivity

```
AX = 1B


on return:
BX = horizontal coordinates per pixel  (ó 100)
CX = vertical coordinates per pixel  (ó 100)
DX = double speed threshold


- returns mouse sensitivity information as the number of mouse
  coordinates per screen pixel
```

INT 33,1C - Set Mouse Interrupt Rate (InPort only)

```
        AX = 1C
        BX = rate code
           = 0  no interrupts
           = 1  30 interrupts per second
           = 2  50 interrupts per second
           = 3  100 interrupts per second
           = 4  200 interrupts per second



        - work with the InPort mouse only
        - sets the rate the mouse status is polled by the mouse
driver
        - faster rates provide better resolution but take away CPU
time
        - values in BX > 4 can cause unpredicatable results
```

INT 33,1D - Set Mouse CRT Page

```
AX = 1D
BX = CRT page number


returns nothing


- sets the CRT page which the mouse cursor is displayed
- see VIDEO PAGES
```

Video modes and corresponding number of pages

Mode	Pages	Adapters
00	pages 0-7	(CGA,EGA,MCGA,VGA)
01	pages 0-7	(CGA,EGA,MCGA,VGA)
02	pages 0-3	(CGA)
	pages 0-7	(EGA,MCGA,VGA)
03	pages 0-3	(CGA)
	pages 0-7	(EGA,MCGA,VGA)
07	pages 0-7	(EGA,VGA)
	no pages	(MDA)
0D	pages 0-7	(EGA,VGA)
0E	pages 0-4	(EGA,VGA)
0F	pages 0-1	(EGA,VGA)
10	pages 0-1	(EGA,VGA)

INT 33,1E - Get Mouse CRT Page

```
AX = 1E


on return:
BX = CRT page number cursor is displayed on


- see VIDEO PAGES
```

INT 33,20 - Enable Mouse Driver

```
AX = 20h

returns nothing

- reinstalls the mouse drivers interrupt vectors for INT 10
and
  INT 71 (8088/86) and INT 74 (286/386)
- see INT 33,1F
```

INT 33,21 - Reset Mouse Software

```
AX = 21h

on return:
AX = 0021  mouse driver not installed
     FFFF  mouse driver installed
BX = 2  mouse driver installed

- similar to INT 33,0 but does not reset the mouse hardware or
  display variables
- both AX and BX must be correct for successful reset
- interrupt routines set through INT 33,18 are preserved
- resets the mouse to the following defaults:

  .  mouse is positioned to screen center
  .  mouse cursor is reset and hidden
  .  no interrupts are enabled (mask = 0)
  .  double speed threshold set to 64 mickeys per second
  .  horizontal mickey to pixel ratio (8 to 8)
  .  vertical mickey to pixel ratio (16 to 8)
  .  max width and height are set to maximum for video mode
```

INT 33,22 - Set Language for Messages

```
AX = 22h
BX = language number (with /L switch value):
   = 0  English     n/a
   = 1  French      F
   = 2  Dutch       NL
   = 3  German      D
   = 4  Swedish     S
   = 5  Finnish     SF
   = 6  Spanish     E
   = 7  Portuguese  P
   = 8  Italian     I


returns nothing


- only works with international version of the mouse driver
- see   INT 33,23
```

INT 33,23 - Get Language Number

```
AX = 23h


on return:
BX = language number (with /L switch value):
   = 0  English     n/a
   = 1  French      F
   = 2  Dutch       NL
   = 3  German      D
   = 4  Swedish     S
   = 5  Finnish     SF
   = 6  Spanish     E
   = 7  Portuguese  P
   = 8  Italian     I

- only works with international version of the mouse driver
- returns English (0) if not international version

- see   INT 33,22
```

NEC æPD765 - Floppy Disk Controller - 8272A

PS/2 FDC Diskette Status Register A at 3F0h

```
|7|6|5|4|3|2|1|0|  3F0h  PS/2 Disk Status Register A (read-only)
 | | | | | | | `---- direction
 | | | | | | `----- write protect
 | | | | | `------ index
 | | | | `------- head 1 select
 | | | `-------- track 0
 | | `--------- step
 | `---------- second drive installed
 `----------- interrupt pending
```

PS/2 FDC Diskette Status Register B at 3F1h

```
|7|6|5|4|3|2|1|0|  3F1h  PS/2 Disk Status Register B (read-only)
 | | | | | | | `---- motor enable 0
 | | | | | | `---- motor enable 1
 | | | | | `---- write enable
 | | | | `---- read data (toggles w/positive transition in -RD DATA)
 | | | `---- write data (toggles w/positive transition in WR DATA)
 | | `---- drive select
 `------ reserved
```

FDC Digital Output Register at 3F2h (all systems)

```
|7|6|5|4|3|2|1|0|  port 3F2h  (write only)
 | | | | | | `------ floppy drive select (0=A, 1=B, 2=floppy C, ...)
 | | | | | `------- 1 = FDC enable, 0 = hold FDC at reset
 | | | | `-------- 1 = DMA & I/O interface enabled (reserved PS/2)
 | | | `--------- 1 = turn floppy drive A motor on
 | | `---------- 1 = turn floppy drive B motor on
 | `----------- 1 = turn floppy drive C motor on; (reserved PS/2)
 `------------ 1 = turn floppy drive D motor on; (reserved PS/2)
```

- used to control drive motors, drive selection, and feature enable
- PS/2 only uses bit 0 for floppy drive select; bit 1 is reserved
- PS/2 only uses bits 5 & 4 for motor enable; bits 7&6 are reserved
- all DOR bits are cleared during controller reset

FDC Main Status Register at 3F4h (all systems)

```
|7|6|5|4|3|2|1|0|  port 3F4h  (read only)
 | | | | | | | `---- floppy drive 0 in seek mode/busy
 | | | | | | `----- floppy drive 1 in seek mode/busy
 | | | | | `------ floppy drive 2 in seek mode/busy (reserved PS/2)
 | | | | `------- floppy drive 3 in seek mode/busy (reserved PS/2)
 | | | `-------- FDC read or write command in progress
 | | `--------- FDC is in non-DMA mode
 | `---------- I/O direction;  1 = FDC to CPU; 0 = CPU to FDC
 `----------- data reg ready for I/O to/from CPU (request for master)
```

FDC Command Status Register 0 at 3F5h (all systems)

```
|7|6|5|4|3|2|1|0|  Command Status Register 0 at port 3F5h
 | | | | | | `------ unit selected at interrupt (0=A, 1=B, 2=...)
 | | | | | `------- head number at interrupt (head 0 or 1)
 | | | | `-------- not ready on read/write or SS access to head 1
 | | | `--------- equipment check (see note)
 | | `---------- set to 1 when FDD completes a seek command
 `------------- last command status (see below)
```

Bits

76	**Last Command Status**
00	command terminated successfully
01	command execution started but terminated abnormally
10	invalid command issued
11	command terminated abnormally due to a change in state of the Ready Signal from the FDC (reserved on PS/2)

- equipment check can occur if FDD signals a fault or track zero is
  not found after 77 steps on a recalibrate command
- PS/2 only uses bits 1-0 for drive (values are 01b and 10b)

FDC Command Status Register 1 at 3F5h (all systems)

```
|7|6|5|4|3|2|1|0|  Command Status Register 1 at port 3F5h
 | | | | | | | `---- FDC cannot find ID address mark (see reg 2)
 | | | | | | `----- write protect detected during write
 | | | | | `------ FDC cannot find sector ID
 | | | | `------- unused (always zero)
 | | | `-------- over-run;  FDC not serviced in reasonable time
 | | `--------- data error (CRC) in ID field or data field
 | `---------- unused (always zero)
 `----------- end of cylinder; sector# greater than sectors/track
```

- bit 0 of Status Register 1 and bit 4 of Status Register 2 are
  related and mimic each other

FDC Command Status Register 2 at 3F5h (all systems)

```
|7|6|5|4|3|2|1|0|  Command Status Register 2 at port 3F5h
 | | | | | | | `---- missing address mark in data field
 | | | | | | `----- bad cylinder, ID not found and Cyl Id=FFh
 | | | | | `------ scan command failed, sector not found in cylinder
 | | | | `------- scan command equal condition satisfied
 | | | `-------- wrong cylinder detected
 | | `--------- CRC error detected in sector data
 | `---------- sector with deleted data address mark detected
 `----------- unused (always zero)
```

- bit 0 of Status Register 1 and bit 4 of Status Register 2 are
  related and mimic each other

FDC Command Status Register 3 at 3F5h (FDD status, all systems)

```
        |7|6|5|4|3|2|1|0|  Floppy Disk Drive Status at port 3F5h
         | | | | | | `------ FDD unit selected status (0=A, 1=B,
2=...)
         | | | | | `------- FDD side head select status (0=head 0,
1=head 1)
         | | | | `-------- FDD two sided status signal
         | | | `---------- FDD track zero status signal
         | | `----------- FDD ready status signal
         | `------------ FDD write protect status signal
         `------------- FDD fault status signal
```

PS/2 FDC Digital Input Register at 3F7h

```
        |7|6|5|4|3|2|1|0|  3F7h  PS/2 Digital Input Register (read
only)
         | | | | | | | `--- high density select
         | | `-------------- reserved
         `---------------- diskette change
```

PS/2 FDC Configuration Control Register at 3F7h

```
        |7|6|5|4|3|2|1|0|  3F7h  PS/2 Config. Control Register
(write only)
         | | | | | | `---- DRC1, DRC0  (see below)
         `---------------- reserved

        DRC1 DRC0
          0   0    500000 bit per second mode
          0   1    reserved
          1   0    250000 bit per second mode
          1   1    reserved
```

- Digital Input Register is used to sense the state of the (-diskette change) and the (-high density select) signals
- Configuration Control Register is used to set the transfer rate

FDC Programming Considerations

Three phases of command execution:

1. Command phase; commands are sent from the CPU to the FDC via port 3F5h; bit 6 of the Status Register at 3F4h must be zero
2. Execution phase; FDC executes instruction & generates INT 6
3. Result phase; status and other information is available to CPU; INT 6 sets bit 7 of BIOS Data Area location 40:3E which can be polled for completion status

Example of a read operation:

1. turn disk motor on and set delay time for drive spin up
2. perform seek operation; wait for disk interrupt
3. prepare DMA chip to move data to memory
4. send read command and wait for transfer complete interrupt
5. read status information
6. turn disk motor off

Floppy Diskette Controller Operations (15 commands)

Read Data	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	MT	MF	SK	0	0	1	1	0
command byte 1:	?	?	?	?	?	HD	US1	US0
command byte 2:	cylinder number							
command byte 3:	head number							
command byte 4:	sector number							
command byte 5:	bytes per sector							
command byte 6:	end of track (last sector in track)							
command byte 7:	gap 3 length							
command byte 8:	data length (if cmd byte 5==0)							
result byte 0:	status register 0							
result byte 1:	status register 1							
result byte 2:	status register 2							
result byte 3:	cylinder number							

```
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector

Read Deleted Data  D7  D6  D5  D4  D3  D2  D1  D0

command byte 0:    MT  MF  SK   0   1   1   0   0
command byte 1:     ?   ?   ?   ?   ?  HD  US1 US0
command byte 2:    cylinder number
command byte 3:    head number
command byte 4:    sector number
command byte 5:    bytes per sector
command byte 6:    end of track (last sector in track)
command byte 7:    gap 3 length
command byte 8:    data length (if cmd byte 5==0)
result byte 0:     status register 0
result byte 1:     status register 1
result byte 2:     status register 2
result byte 3:     cylinder number
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector

Write Data         D7  D6  D5  D4  D3  D2  D1  D0

command byte 0:    MT  MF   0   0   0   1   0   1
command byte 1:     ?   ?   ?   ?   ?  HD  US1 US0
command byte 2:    cylinder number
command byte 3:    head number
command byte 4:    sector number
command byte 5:    bytes per sector
command byte 6:    end of track (last sector in track)
command byte 7:    gap 3 length
command byte 8:    data length (if cmd byte 5==0)
result byte 0:     status register 0
result byte 1:     status register 1
result byte 2:     status register 2
result byte 3:     cylinder number
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector

Write Deleted Data D7  D6  D5  D4  D3  D2  D1  D0

command byte 0:    MT  MF   0   0   1   0   0   1
command byte 1:     ?   ?   ?   ?   ?  HD  US1 US0
```

```
command byte 2:    cylinder number
command byte 3:    head number
command byte 4:    sector number
command byte 5:    bytes per sector
command byte 6:    end of track (last sector in track)
command byte 7:    gap 3 length
command byte 8:    data length (if cmd byte 5==0)
result byte 0:     status register 0
result byte 1:     status register 1
result byte 2:     status register 2
result byte 3:     cylinder number
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector
```

Read a Track (Diagnostic)	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	0	MF	SK	0	0	0	1	0
command byte 1:	?	?	?	?	?	HD	US1	US0

```
command byte 2:    cylinder number
command byte 3:    head number
command byte 4:    sector number
command byte 5:    bytes per sector
command byte 6:    end of track (last sector in track)
command byte 7:    gap 3 length
command byte 8:    data length (if cmd byte 5==0)
result byte 0:     status register 0
result byte 1:     status register 1
result byte 2:     status register 2
result byte 3:     cylinder number
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector
```

Read ID	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	0	MF	0	0	1	0	1	0
command byte 1:	?	?	?	?	?	HD	US1	US0

```
result byte 0:     status register 0
result byte 1:     status register 1
result byte 2:     status register 2
result byte 3:     cylinder number
result byte 4:     head number
result byte 5:     sector number
result byte 6:     bytes per sector
```

Format a Track (Write Sector IDs)	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	0	MF	0	0	1	1	0	1
command byte 1:	?	?	?	?	?	HD	US1	US0
command byte 2:	bytes per sector							
command byte 3:	sectors per track							
command byte 4:	gap 3 length							
command byte 5:	filler pattern to write in each byte							
result byte 0:	status register 0							
result byte 1:	status register 1							
result byte 2:	status register 2							
result byte 3:	cylinder number							
result byte 4:	head number							
result byte 5:	sector number							
result byte 6:	bytes per sector							

Scan Equal	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	MT	MF	SK	1	0	0	0	1
command byte 1:	?	?	?	?	?	HD	US1	US0
command byte 2:	cylinder number							
command byte 3:	head number							
command byte 4:	sector number							
command byte 5:	bytes per sector							
command byte 6:	end of track (last sector in track)							
command byte 7:	gap 3 length							
command byte 8:	scan test (1=scan contiguous, 2=scan alternate)							
result byte 0:	status register 0							
result byte 1:	status register 1							
result byte 2:	status register 2							
result byte 3:	cylinder number							
result byte 4:	head number							
result byte 5:	sector number							
result byte 6:	bytes per sector							

Scan Low or Equal	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
command byte 0:	MT	MF	SK	1	1	0	0	1
command byte 1:	?	?	?	?	?	HD	US1	US0
command byte 2:	cylinder number							
command byte 3:	head number							
command byte 4:	sector number							
command byte 5:	bytes per sector							
command byte 6:	end of track (last sector in track)							
command byte 7:	gap 3 length							

command byte 8: scan test (1=scan contiguous, 2=scan alternate)
result byte 0: status register 0
result byte 1: status register 1
result byte 2: status register 2
result byte 3: cylinder number
result byte 4: head number
result byte 5: sector number
result byte 6: bytes per sector

Scan High or Equal **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0:	MT	MF	SK	1	1	1	0	1
command byte 1:	?	?	?	?	?	HD	US1	US0

command byte 2: cylinder number
command byte 3: head number
command byte 4: sector number
command byte 5: bytes per sector
command byte 6: end of track (last sector in track)
command byte 7: gap 3 length
command byte 8: scan test (1=scan contiguous, 2=scan alternate)
result byte 0: status register 0
result byte 1: status register 1
result byte 2: status register 2
result byte 3: cylinder number
result byte 4: head number
result byte 5: sector number
result byte 6: bytes per sector

Recalibrate **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0:	0	0	0	0	0	1	1	1
command byte 1:	?	?	?	?	?	0	US1	US0

returns nothing

Sense Interrupt Status **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0:	0	0	0	0	1	0	0	0

result byte 0: status register 0
result byte 1: present cylinder number

Specify Step & Head Load **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0:	0	0	0	0	0	0	1	1

command byte 1: step rate time | head unload time

command byte 2: ------head load time------ ND
returns nothing

Sense Drive Status **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0: 0 0 0 0 0 1 0 0
command byte 1: ? ? ? ? ? HD US1 US0
result byte 0: status register 3

Seek **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0: 0 0 0 0 1 1 1 1
command byte 1: ? ? ? ? ? HD US1 US0
command byte 2: new cylinder number
returns nothing

æPD765 Version **D7 D6 D5 D4 D3 D2 D1 D0**

command byte 0: ? ? ? 1 0 0 0 0
result byte 0: status register 0
90h = æPD765B; 80h = æPD765A or æPD765A-2

Invalid Command

result byte 0: status register 0 (value of 80h)

Key to Abbreviations

HD = Head Number Selected
MT = Multi-Track
MF = MFM mode
ND = Non-DMA mode
SK = SKip Deleted-data address mark
US0 = drive select bit 0
US1 = drive select bit 1

Head Load Time = 2 to 254ms in 2ms increments
Head Unload Time = 16 to 240ms in 16ms increments
Step Rate Time = 1 to 16ms in 1ms increments

- PS/2 systems use the 8272A diskette controller which is software and port compatible with the NEC æPD765
- accessed through ports 3F0h-3F7h; NEC æPD765 is accessed through ports 3F2h, 3F4h and 3F5h; the 8272A uses ports 3F0h,

3F1h,
  3F2h, 3F4h, 3F5h and 3F7h
- data, command and status registers are all accessed through
  port 3F5h a register stack with one address presented to the bus
- bit 7 of BIOS Data Area byte 40:3E can be polled to determine
  if a disk operation has completed; this bit is set by the
  interrupt handler when the operation has completed; it should
  be reset before continuing on with the next FDC operation

- see BIOS Data Area INT TABLE INT 13

DIAGS IBM PC Diagnostic Error Codes

Code	Description (Diagnostic Error Codes)
1x	undetermined problem errors
2x	power supply errors
61	battery error (PS/2)
62	configuration changed but no adaptors changed or CRC error
63	memory size during POST does not match CMOS RAM
65	card Id's in CMOS do not match system
1xx	**System Board Errors**
101	system board failed
102	BIOS ROM checksum error (PC, XT), Timer (AT)
103	BASIC ROM checksum error (PC, XT), Timer interrupt (AT)
104	interrupt controller (PC, XT), Protected mode (AT)
105	timer (PC,XT), Last 8042 command not accepted (AT)
106	converting logic test failure
107	adapter card or math coprocessor (NMI)
108	timer bus test
109	DMA test error
110	system board memory error (PS/2)
111	adapter memory
112	system unit adapter failure
114	system unit and adapter card unit failure (PS/1)
121	unexpected hardware interrupt
131	cassette wrap test failed
151	real-time clock, CMOS RAM or battery
152	real-time clock
161	system options error, battery failure
162	CMOS RAM configuration error
163	CMOS time and date not set
164	system memory configuration is incorrect
165	adapter added/removed
199	user indicated configuration not correct
2xx	**RAM Errors**
201	memory test error
202	memory address error (Line error 0 - 15)
203	memory address error (Line error 16 - 23)
216	motherboard memory

3xx Keyboard Errors

301 keyboard did not respond to software reset or a stuck (number preceding 301 is scan code for stuck key)
302 user indicated keyboard error or AT system unit is locked
303 keyboard or system board error
304 keyboard or system board error; CMOS does not match system
341 replace keyboard
342 replace keyboard interface cable
343 replace enhancement card or cable
365 keyboard (replace keyboard)
366 interface cable (replace cable)
367 enhancement card or cable (replace)

4xx Monochrome Monitor Errors

401 monochrome memory test, horizontal sync frequency test, or video test failed
408 user indicated display attributes failure
416 user indicated character set failure
424 user indicated 80x25 mode failure
432 parallel port test failed (monochrome adapter)

5xx Color Monitor Errors

501 CGA memory test failed, horizontal sync frequency test, or video test failed
508 user indicated display attribute failure
516 user indicated character set failure
524 user indicated 80x25 mode failure
532 user indicated 40x25 mode failure
540 user indicated 320x200 graphics mode failure
548 user indicated 640x200 graphics mode failure
556 light pen test
564 user indicated screen paging test

6xx Diskette Drive/Adapter Errors

601 diskette power on diagnostics test failed
602 diskette test failed (boot record not valid)
603 diskette size error
606 diskette verify function failed
607 write protected diskette
608 bad command diskette status returned
610 diskette initialization failed
611 time-out - diskette status returned

612 bad NEC FDC - diskette status returned
613 bad DMA - diskette status returned
614 DMA boundary error
621 bad seek - diskette status returned
622 bad CRC - diskette status returned
623 record not found - diskette status returned
624 bad address mark - diskette status returned
625 bad NEC seek - diskette status returned
626 diskette data compare error
627 diskette change line error
628 diskette removed

7xx 8087 Math Coprocessor
701 coprocessor test failure

9xx Parallel Printer Adapter Errors
901 parallel printer adapter test failed

10xx Alternate Parallel Printer Adapter
1001 alternate printer adapter test failure

11xx Asynchronous Communications Adapter Errors
1101 asynchronous communications adapter test failed
1110 modem status register not clear
1111 ring-indicate
1112 trailing edge ring-indicate
1113 receive and delta receive line signal detect
1114 receive line signal detect
1115 delta receive line signal detect
1116 line control register: all bits cannot be set
1117 line control register: all bits cannot be reset
1118 transmit holding and/or shift register stuck on
1119 data ready stuck on
1120 interrupt enable register: all bits cannot be set
1121 interrupt enable register: all bits cannot be reset
1122 interrupt pending stuck on
1123 interrupt ID register stuck on
1124 modem control register: all bits cannot be set
1125 modem control register: all bits cannot be reset
1126 modem status register: all bits cannot be set
1127 modem status register: all bits cannot be reset
1128 interrupt ID
1129 cannot force overrun error
1130 no modem status interrupt
1131 invalid interrupt pending
1132 no data ready

1133 no data available interrupt
1134 no transmit holding interrupt
1135 no interrupts
1136 no received line status interrupt
1137 no receive data available
1138 transmit holding register not empty
1139 no modem status interrupt
1140 transmit holding register not empty
1141 no interrupts
1142 no IRQ4 interrupt
1143 no IRQ3 interrupt
1144 no data transferred
1145 maximum baud rate
1146 minimum baud rate
1148 time-out error
1149 invalid data returned
1150 modem status register error
1151 no DSR and delta DSR
1152 no DSR
1153 no delta DSR
1154 modem status register not clear
1155 no CTS and delta CTS
1156 no CTS
1157 no delta CTS

12xx Alternate Asynchronous Communications Adapter
1200-1257 same as 1100-1157

13xx Game Control Adapter Errors
1301 game control adapter test failed
1302 joystick test failed
1380 audio card and joystick (PS/1)

14xx Printer Errors
1401 printer test failed
1404 matrix printer failed

15xx SDLC Communications Adapter Errors
1501 adapter test failure
1510 8255 port b failure
1511 8255 port a failure
1512 8255 port c failure
1513 8253 timer 1 did not reach terminal count
1514 8253 timer 1 stuck on
1515 8253 timer 0 did not reach terminal count
1516 8253 timer 0 stuck on

1517 8253 timer 2 did not reach terminal count
1518 8253 timer 2 stuck on
1519 8273 port b error
1520 8273 port a error
1521 8273 command/read time-out
1522 interrupt level 4 failure
1523 ring Indicate stuck on
1524 receive clock stuck on
1525 transmit clock stuck on
1526 test indicate stuck on
1527 ring indicate not on
1528 receive clock not on
1529 transmit clock not on
1530 test indicate not on
1531 DSR not on
1532 CD not on
1533 CTS not on
1534 DSR stuck on
1535 CD stuck on
1536 CTS stuck on
1537 level 3 interrupt failure
1538 receive interrupt results error
1539 wrap data compare error
1540 DMA channel 1 error
1541 DMA channel 1 error
1542 error in 8273 error checking or status reporting
1547 stray interrupt level 4
1548 stray interrupt level 3
1549 interrupt presentation sequence time-out

16xx Display Emulation Errors (327x, 5520, 525x)

17xx Fixed Disk Errors
1701 post error
1702 adapter error
1703 drive error (seek)
1704 adapter or drive error
1705 no record found
1706 write fault error
1707 track 0 error
1708 head select error
1709 defective error check
1710 read buffer overrun
1711 bad address mark
1712 undetermined error
1713 data compare error

1714 drive not ready
1780 disk 0 failure
1781 disk 1 failure
1782 disk controller failure
1790 fixed disk 0 error
1791 fixed disk 1 error

18xx I/O Expansion Unit Errors
1801 I/O expansion unit POST error
1810 enable/disable failure
1811 extender card wrap test failed (disabled)
1812 high order address lines failure (disabled)
1813 wait state failure (disabled)
1814 enable/disable could not be set on
1815 wait state failure (enabled)
1816 extender card wrap test failed (enabled)
1817 high order address lines failure (enabled)
1818 disable not functioning
1819 wait request switch not set correctly
1820 receiver card wrap test failure
1821 receiver high order address lines failure

19xx 3270 PC Attachment Card Errors

20xx Bisync Communications Adapter Errors
2010 8255 port a failure
2011 8255 port b failure
2012 8255 port c failure
2013 8253 timer 1 did not reach terminal count
2014 8253 timer 1 stuck on
2016 8253 timer 2 didn't reach terminal count or timer 2 stuck on
2017 8251 DSR failed to come on
2018 8251 CTS not sensed
2019 8251 DSR stuck on
2020 8251 CTS stuck on
2021 8251 hardware reset failed
2022 8251 software reset failed
2023 8251 software error reset failed
2024 8251 transmit ready did not come on
2025 8251 receive ready did not come on
2026 8251 could not force "overrun" error status
2027 interrupt failure, no timer interrupt
2028 transmit interrupt, card or planar failure
2029 transmit interrupt, card failure
2030 receive interrupt, card or planar failure

2031 receive interrupt, card failure
2033 ring indicate stuck on
2034 receive clock stuck on
2035 transmit clock stuck on
2036 test indicate stuck on
2037 ring indicate stuck on
2038 receive clock not on
2039 transmit clock not on
2040 test indicate not on
2041 DSR not on
2042 CD not on
2043 CTS not on
2044 DSR stuck on
2045 CD stuck on
2046 CTS stuck on
2047 unexpected transmit interrupt
2048 unexpected receive interrupt
2049 transmit data did not equal receive data
2050 8251 detected overrun error
2051 lost DSR during data wrap
2052 receive time-out during data wrap

21xx Alternate Bisync Communications Adapter Errors

2100-2152 same as 2000-2052

22xx Cluster Adapter Errors

24xx EGA Errors

2401 adapter test failure
2408 user-indicated display attributes
2416 user-indicated character set
2424 user-indicated 80x25 mode
2432 user-indicated 40x25 mode
2440 user-indicated 320x200 graphics mode
2448 user-indicated 640x200 graphics mode
2456 light pen test
2464 user-indicated screen paging test

2501 EGA Display Error

28xx 3278/79 Emulator Adapter (PC, XT)

29xx Color Matrix Printer Errors

30xx Local Area Network Adapter

31xx Alternate Local Area Network Adapter

33xx Compact Printer Errors

36xx IEEE 488 Adapter

37xx Reserved for Future Use

38xx Data Acquisition Adapter

39xx PGA Display and/or Adapter

48xx Internal Modem

49xx Alternate Internal Modem

71xx Voice Communications Adapter

73xx 3.5 Inch Diskette Drive

7301 diskette drive/adapter test failure
7306 diskette change line error
7307 write-protected diskette
7308 bad command
7310 track zero error
7311 timeout
7312 bad NEC7313 bad DMA
7314 DMA boundary error
7315 bad index
7316 speed error
7321 bad seek
7322 bad CRC
7323 record not found
7324 bad address mark
7325 bad NEC seek

86xx PS/1 Mouse Error

- all error codes for the diagnostic and advanced diagnostic packages for the PC, XT and AT are represented with the device number followed by two digits other than 00. The device number plus 00 represents successful completion of the test.

CMOS RTC - Real Time Clock and Memory (ports 70h & 71h)

Reg# **Description**

```
 00  RTC seconds
 01  RTC seconds alarm
 02  RTC minutes
 03  RTC minutes alarm
 04  RTC hours
 05  RTC hours alarm
 06  RTC day of week
 07  RTC day of month
 08  RTC month
 09  RTC year
 0A  RTC Status register A:

    |7|6|5|4|3|2|1|0|  RTC Status Register A
     | | | | `---------- rate selection Bits for divider
output
     | | | |             frequency (set to 0110 = 1.024kHz,
976.562æs)
     | `--------------- 22 stage divider, time base being
used;
     |                    (initialized to 010 = 32.768kHz)
     `---------------- 1=time update in progress, 0=time/date
available

 0B  RTC Status register B:

    |7|6|5|4|3|2|1|0|  RTC Status Register B
     | | | | | | | `---- 1=enable daylight savings,
0=disable (default)
     | | | | | | `----- 1=24 hour mode, 0=12 hour mode (24
default)
     | | | | | `------ 1=time/date in binary, 0=BCD (BCD
default)
     | | | | `------- 1=enable square wave frequency,
0=disable
     | | | `-------- 1=enable update ended interrupt,
0=disable
     | | `--------- 1=enable alarm interrupt, 0=disable
```

```
            | `---------- 1=enable periodic interrupt, 0=disable
            `----------- 1=disable clock update, 0=update count
normally
```

0C RTC Status register C (read only):

```
           |7|6|5|4|3|2|1|0|  RTC Status Register C  (read only)
            | | | | `---------- reserved (set to 0)
            | | | `----------- update ended interrupt enabled
            | | `----------- alarm interrupt enabled
            | `----------- periodic interrupt enabled
            `----------- IRQF flag
```

0D RTC Status register D (read only):

```
           |7|6-0|  RTC Status Register D  (read only)
            |  `----- reserved (set to 0)
            `------ 1=CMOS RAM has power, 0=CMOS RAM has lost power
```

0E Diagnostic status byte:

```
           |7|6|5|4|3|2|1|0|  Diagnostic Status Byte
            | | | | | | `------ reserved
            | | | | | `------- 1=time is invalid, 0=ok (POST
validity check)
            | | | | `-------- 1=fixed disk 0 failed
initialization, 0=ok
            | | | `--------- 1=memory size doesn't match config
info, 0=ok
            | | `---------- 1=invalid config info found, 0=ok (see
below)
            | `----------- 1=config record checksum is bad, 0=ok
            `------------ 1=RTC lost power, 0=power state stable
```

0F Shutdown status byte:

```
              0  soft reset or unexpected shutdown
              1  shut down after memory size determination
              2  shut down after memory test
              3  shut down with memory error
              4  shut down with boot loader request
              5  JMP DWORD request with INT init
              6  protected mode test 7 passed
              7  protected mode test 7 failed
              8  protected mode test1 failed
              9  block move shutdown request
```

A JMP DWORD request without INT init

10 Diskette drive type for A: and B:

```
|7|6|5|4|3|2|1|0|  Diskette drive type for A: and B:
 | | | | `---------- second diskette type
 `----------------- first diskette type

   0000  no drive installed
   0001  DSDD 48 TPI drive
   0010  DSQD 96 TPI drive
   other values are reserved
```

11 Reserved
12 Fixed disk drive type for drive 0 and drive 1

```
|7|6|5|4|3|2|1|0|  Diskette drive type for A: and B:
 | | | | `---------- second hard disk drive code (0000=no disk)
 `----------------- first hard disk drive code (0000=no disk)
```

13 Reserved
14 Equipment byte

```
|7|6|5|4|3|2|1|0|  Equipment byte
 | | | | | | | `---- 1=diskette drives installed, 0=none
 | | | | | | `----- 1=math coprocessor installed, 0=none
 | | | | `-------- unused
 | | `--------- primary display
 `------------ number of diskette drives installed
```

Bits 54	Primary Display	Bits 76	Number of Drives
00	reserved	00	1 diskette drive
01	40 column color	01	2 diskette drives
10	80 column color	10	reserved
11	monochrome	11	reserved

15 LSB of system base memory in 1k blocks
16 MSB of system base memory in 1k blocks
17 LSB of total extended memory in 1k blocks
18 MSB of total extended memory in 1k blocks
19 Drive C extension byte (reserved AT)
1A Drive D extension byte (reserved AT)
1B 13 bytes reserved

```
         2E  CMOS checksum of bytes 10h-20h (MSB)
         2F  CMOS checksum of bytes 10h-20h (LSB)
         30  LSB of extended memory size found above 1 megabyte
during POST
         31  MSB of extended memory size found above 1 megabyte
during POST
         32  Date century byte in BCD ( BIOS interface to read and
set)
         33  Information flags (set during power-on)

            |7|6|5-0|  Information Flags
             | |  `----- reserved
             | `------- initial setup message flag
             `-------- 1=IBM 128k expansion installed, 0=none

         34  12 bytes reserved
```

Programming Considerations:

Write CMOS address to read or write to port 70h
Read/write port 71h to get/set data

- the information here is only applicable to AT and PS/2 systems
- INT 1A is used to read/set the Time of Day and Alarm. To use the alarm, INT 4A must be a valid interrupt service routine.
- configuration settings are maintained using the Motorola MC146818 Real Time Clock. Each of this chips 64 memory registers is used for storage (0-3F).
- Bit 5 of the diagnostic (0Eh) status byte is set during a power on test. This Bit is set if no floppy disks are found or the display doesn't match the system display switch setting.
- all addresses sent to port 70h have Bits 7&6 clear since Bit 7 of port 70h is used to enable/disable NMI. Setting this Bit 7 enables NMI, clearing this Bit disables NMI.
- when masking the NMI through using port 70H, port 71H should be

read immediately after or the RTC may be left in an unknown state.
This wont affect the PS/2 watchdog timer or system channel timeout.

- see INT 1A

INT 1A,80 - Setup Sound Multiplexer (PCjr only)

```
AL = 00 - source is 8253 channel 2
   = 01 - source is cassette input
   = 02 - source is I/O channel "AUDIO IN"
   = 03 - source is sound generator chip
```

8042 - Keyboard Controller (AT,PS/2)

8042 Status Register (port 64h read)

```
|7|6|5|4|3|2|1|0|  8042 Status Register
 | | | | | | | `---- output register (60h) has data for
system
 | | | | | | `----- input register (60h/64h) has data for
8042
 | | | | | `------ system flag (set to 0 after power on
reset)
 | | | | `------- data in input register is command (1) or
data (0)
 | | | `-------- 1=keyboard enabled, 0=keyboard disabled
(via switch)
 | | `--------- 1=transmit timeout (data transmit not
complete)
 | `---------- 1=receive timeout (data transmit not
complete)
 `----------- 1=even parity rec'd, 0=odd parity rec'd
(should be odd)
```

Port	Mode	Description
64h	read	8042 status register. Can be read at any time. See table above for more information.
64h	write	8042 command register. Writing this port sets Bit 3 of the status register to 1 and the byte is treated as a controller command. Devices attached to the 8042 should be disabled before issuing commands that return data since data in the output register will be overwritten.
60h	read	8042 output register (should only be read if Bit 0 of status port is set to 1)
60h	write	8042 data register. Data should only be written if

Bit 1 of the status register is zero (register is empty).
When this port is written Bit 3 of the status register
is set to zero and the byte is treated as a data. The
8042 uses this byte if it's expecting data for a previous
command, otherwise the data is written directly to the
keyboard. See KEYBOARD COMMANDS for information on
programming the actual keyboard hardware.

8042 Commands Related to PC Systems (Port 64h)

Command	Description
20	Read 8042 Command Byte: current 8042 command byte is placed in port 60h.
60	Write 8042 Command Byte: next data byte written to port 60h is placed in 8042 command register. Format:

```
|7|6|5|4|3|2|1|0|  8042 Command Byte
 | | | | | | | `---- 1=enable output register full interrupt
 | | | | | | `----- should be 0
 | | | | | `------ 1=set status register system, 0=clear
 | | | | `------- 1=override keyboard inhibit, 0=allow inhibit
 | | | `-------- disable keyboard I/O by driving clock line low
 | | `--------- disable auxiliary device, drives clock line low
 | `---------- IBM scancode translation 0=AT, 1=PC/XT
 `----------- reserved, should be 0
```

A4 Password Installed Test: returned data can be read
from port 60h; FA=password installed, F1=no password
A5 Load Security: bytes written to port 60h will be read
until a null (0) is found.
A6 Enable Security: works only if a password is already
loaded
A7 Disable Auxiliary Interface: sets Bit 5 of command
register
stopping auxiliary I/O by driving the clock line low
A8 Enable Auxiliary Interface: clears Bit 5 of command
register
A9 Auxiliary Interface Test: clock and data lines are
tested;
results placed at port 60h are listed below:

00 no error
01 keyboard clock line is stuck low
02 keyboard clock line is stuck high
03 keyboard data line is stuck low
04 keyboard data line is stuck high

AA Self Test: diagnostic result placed at port 60h,
55h=OK
AB Keyboard Interface Test: clock and data lines are
tested;
results placed at port 60h are listed above with
command A9
AC Diagnostic Dump: sends 16 bytes of 8042's RAM,
current input
port state, current output port state and 8042
program status
word to port 60h in scan-code format.
AD Disable Keyboard Interface: sets Bit 4 of command
register
stopping keyboard I/O by driving the clock line low
AE Enable Keyboard Interface: clears Bit 4 of command
register
enabling keyboard interface.
C0 Read Input Port: data is read from its input port
(which is
inaccessible to the data bus) and written to output
register
at port 60h; output register should be empty before
call.

|7|6|5|4|3-0| 8042 Input Port

```
               | | | | `---- undefined
               | | | `----- 1=enable 2nd 256K of motherboard RAM,
0=disable
               | | `------ 1=manufacturing jumper not installed,
0=installed
               | `------- 1=primary display is MDA, 0=primary
display is CGA
               `-------- 1=keyboard not inhibited, 0=keyboard
inhibited
```

C1 Poll Input Port Low Bits: Bits 0-3 of port 1 placed in status Bits 4-7

C2 Poll Input Port High Bits: Bits 4-7 of port 1 placed in status Bits 4-7

D0 Read Output Port: data is read from 8042 output port (which is inaccessible to the data bus) and placed in output register; the output register should be empty. (see command D1 below)

D1 Write Output Port: next byte written to port 60h is placed in the 8042 output port (which is inaccessible to the data bus)

```
  |7|6|5|4|3|2|1|0|  8042 Output Port
   | | | | | | | `---- system reset line
   | | | | | | `----- gate A20
   | | | | `-------- undefined
   | | | `--------- output buffer full
   | | `---------- input buffer empty
   | `----------- keyboard clock (output)
   `------------ keyboard data (output)
```

D2 Write Keyboard Output Register: on PS/2 systems the next data byte written to port 60h input register is written to port 60h output register as if initiated by a device; invokes interrupt if enabled

D3 Write Auxiliary Output Register: on PS/2 systems the next data byte written to port 60h input register is written to port 60h

output register as if initiated by a device; invokes interrupt
if enabled

D4 Write Auxiliary Device: on PS/2 systems the next data byte written to input register a port at 60h is sent to the auxiliary device

E0 Read Test Inputs: 8042 reads its T0 and T1 inputs; data is placed in output register; Bit 0 is T0, Bit 1 is T1:

```
|1|0|  Test Input Port Bits
 | `---- keyboard clock
 `----- keyboard data
```

Fx Pulse Output Port: Bits 0-3 of the 8042 output port can be pulsed low for 6 æs; Bits 0-3 of command indicate which Bits should be pulsed; 0=pulse, 1=don't pulse; pulsing Bit 0 results in CPU reset since it is connected to system reset line.

- PC systems previous to the AT use the 8255 PPI as a keyboard controller and use the keyboard's internal 8048.
- the keyboard's internal controller buffers up to 16 bytes of make/break code information. This is common among all PC systems and shouldn't be confused with the (32 byte) keyboard buffer maintained by the BIOS.
- see KEYBOARD COMMANDS for information on programming the keyboards internal microprocessor

Keyboard Commands & Responses

Commands System Issues to Keyboard (via 8042 port 60h)

ED Set/Reset Mode Indicators, keyboard responds with ACK then waits for a following option byte. When the option byte is received the keyboard again ACK's and then sets the LED's accordingly. Scanning is resumed if scanning was enabled. If another command is received instead of the option byte (high bit set on) this command is terminated. Hardware defaults to these indicators turned off.

```
|7-3|2|1|0| Keyboard Status Indicator Option Byte
  |  | | `--- Scroll-Lock indicator  (0=off, 1=on)
  |  | `---- Num-Lock indicator  (0=off, 1=on)
  |  `----- Caps-Lock indicator  (0=off, 1=on)
  `------- reserved (must be zero)
```

EE Diagnostic Echo, keyboard echoes the EE byte back to the system without an acknowledgement.

F0 PS/2 Select/Read Alternate Scan Code Sets, instructs keyboard to use one of the three make/break scan code sets. Keyboard responds by clearing the output buffer/typematic key and then transmits an ACK. The system must follow up by sending an option byte which will again be ACK'ed by the keyboard:

```
  00  return byte indicating scan code set in use
  01  select scan code set 1  (used on PC & XT)
```

```
              02  select scan code set 2
              03  select scan code set 3

        F2  PS/2 Read Keyboard ID, keyboard responds with an ACK
and a two
            byte keyboard ID of 83AB.
        F3  Set Typematic Rate/Delay, keyboard responds with ACK
and waits
            for rate/delay byte.   Upon receipt of the rate/delay
byte the
            keyboard responds with an ACK, then sets the new
typematic
            values and scanning continues if scanning was enabled.

            |7|6|5|4|3|2|1|0|  Typematic Rate/Delay Option Byte
             | | | |-+-+-+-+---- typematic rate indicator (see INT
16,3)
             | | | | | `------- A in period formula (see below)
             | | | `---------- B is period formula (see below)
             | `-------------- typematic delay
             `---------------- always zero

            delay = (rate+1) * 250   (in milliseconds)
            rate = (8+A) * (2**B) * 4.17  (in seconds, ñ 20%)

            Defaults to 10.9 characters per second and a 500ms
delay.  If a
            command byte (byte with high bit set) is received
instead of an
            option byte this command is cancelled.
        F4  Enable Keyboard, cause the keyboard to clear its output
buffer
            and last typematic key and then respond with an ACK.
The
            keyboard then begins scanning.
        F5  Default w/Disable, resets keyboard to power-on
condition by
            clearing the output buffer, resetting typematic rate/
delay,
            resetting last typematic key and setting default key
types.
            The keyboard responds with an ACK and waits for the next
            instruction.
        F6  Set Default, resets to power-on condition by clearing
the output
            buffer, resetting typematic rate/delay and last
```

typematic key
and sets default key types. The keyboard responds with an ACK
and continues scanning.

F7 PS/2 Set All Keys to Typematic, keyboard responds by sending an ACK, clearing its output buffer and setting the key type to Typematic. Scanning continues if scanning was enabled. This command may be sent while using any Scan Code Set but only has effect when Scan Code Set 3 is in use.

F8 PS/2 Set All Keys to Make/Break, keyboard responds by sending an ACK, clearing its output buffer and setting the key type to Make/Break. Scanning continues if scanning was enabled. This command may be sent while using any Scan Code Set but only has effect when Scan Code Set 3 is in use.

F9 PS/2 Set All Keys to Make, keyboard responds by sending an ACK, clearing its output buffer and setting the key type to Make. Scanning continues if scanning was enabled. This command may be sent while using any Scan Code Set but only has effect when Scan Code Set 3 is in use.

FA PS/2 Set All Keys to Typematic Make/Break, keyboard responds by sending an ACK, clearing its output buffer and setting the key type to Typematic Make/Break. Scanning continues if scanning was enabled. This command may be sent while using any Scan Code Set but only has effect when Scan Code Set 3 is in use.

FB PS/2 Set Key Type to Typematic, keyboard responds by sending an ACK, clearing its output buffer and then waiting for the key ID (make code from Scan Code Set 3). The specified key type is then

set to typematic. This command may be sent while using any
Scan Code Set but only has effect when Scan Code Set 3 is in use.
FC PS/2 Set Key Type to Make/Break, keyboard responds by sending an
ACK, clearing its output buffer and then waiting for the key ID
(make code from Scan Code Set 3). The specified key type is then
set to Make/Break. This command may be sent while using any Scan
Code Set but only has effect when Scan Code Set 3 is in use.
FD PS/2 Set Key Type to Make, keyboard responds by sending an ACK,
clearing its output buffer and then waiting for the key ID (make
code from Scan Code Set 3). The specified key type is then set
to Make. This command may be sent while using any Scan Code Set
but only has effect when Scan Code Set 3 is in use.
FE Resend, should be sent when a transmission error is detected
from the keyboard
FF Reset, Keyboard sends ACK and waits for system to receive it
then begins a program reset and Basic Assurance Test (BAT).
Keyboard returns a one byte completion code then sets default
Scan Code Set 2.

Keyboard Responses to System (via 8042 port 60h)

00 Key Detection Error or Overrun Error for Scan Code Set 1,
replaces last key in the keyboard buffer if the buffer is full.

AA BAT Completion Code, keyboard sends this to indicate the keyboard
test was successful.
EE Echo Response, response to the Echo command.
F0 Break Code Prefix in Scan Code Sets 2 and 3.
FA Acknowledge, keyboard sends this whenever a valid command or
data byte is received (except on Echo and Resend commands).
FC BAT Failure Code, keyboard sends this to indicate the keyboard
test failed and stops scanning until a response or reset is sent.
FE Resend, keyboard request resend of data when data sent to it is
invalid or arrives with invalid parity.
FF Key Detection Error or Overrun Error for Scan Code Set 2 or 3,
replaces last key in the keyboard buffer if the buffer is full.
id Keyboard ID Response, keyboard sends a two byte ID after ACK'ing
the Read ID command. The byte stream contains 83AB in LSB, MSB
order. The keyboard then resumes scanning.

- command F7 through FD are NOP's on the AT and are ACK'ed but not
acted upon
- see 8042 MAKE CODES BREAK CODES INT 16,3

INT 16,3 - Set Keyboard Typematic Rate (AT+)

```
        AH = 03
        AL = 00  set typematic rate to default
             01  increase initial delay
             02  slow typematic rate by 1/2
             04  turn off typematic chars
             05  set typematic rate/delay

        BH = repeat delay (AL=5)
             0 = 250ms     2 = 750ms
             1 = 500ms     3 = 1000ms
        BL = typematic rate, one of the following  (AL=5)

             00 - 30.0      01 - 26.7      02 - 24.0      03 - 21.8
             04 - 20.0      05 - 18.5      06 - 17.1      07 - 16.0
             08 - 15.0      09 - 13.3      0A - 12.0      0B - 10.9
             0C - 10.0      0D - 9.2       0E - 8.6       0F - 8.0
             10 - 7.5       11 - 6.7       12 - 6.0       13 - 5.5
             14 - 5.0       15 - 4.6       16 - 4.3       17 - 4.0
             18 - 3.7       19 - 3.3       1A - 3.0       1B - 2.7
             1C - 2.5       1D - 2.3       1E - 2.1       1F - 2.0

        returns nothing

        - if the typematic rate is not within range,no action is
taken
        - available on AT and PS/2 machines with extended keyboard
support
        - see   KEYBOARD COMMANDS
```

INT 9 - Hardware Keyboard Make/Break Codes

Key	Make	Break	Key	Make	Break
Backspace	0E	8E	F1	3B	BB
Caps Lock	3A	BA	F2	3C	BC
Enter	1C	9C	F3	3D	BD
Esc	01	81	F4	3E	BE
Left Alt	38	B8	F7	41	C1
Left Ctrl	1D	9D	F5	3F	BF
Left Shift	2A	AA	F6	40	C0
Num Lock	45	C5	F8	42	C2
Right Shift	36	B6	F9	43	C3
Scroll Lock	46	C6	F10	44	C4
Space	39	B9	F11	57	D7
Sys Req (AT)	54	D4	F12	58	D8
Tab	0F	8F			

Keypad Keys	Make	Break
Keypad 0 (Ins)	52	D2
Keypad 1 (End)	4F	CF
Keypad 2 (Down arrow)	50	D0
Keypad 3 (PgDn)	51	D1
Keypad 4 (Left arrow)	4B	CB
Keypad 5	4C	CC
Keypad 6 (Right arrow)	4D	CD
Keypad 7 (Home)	47	C7
Keypad 8 (Up arrow)	48	C8
Keypad 9 (PgUp)	49	C9
Keypad . (Del)	53	D3
Keypad * (PrtSc)	37	B7
Keypad -	4A	CA
Keypad +	4E	CE

Key	Make	Break	Key	Make	Break
A	1E	9E	N	31	B1
B	30	B0	O	18	98
C	2E	AE	P	19	99
D	20	A0	Q	10	90
E	12	92	R	13	93
F	21	A1	S	1F	9F

G	22	A2	T	14	94
H	23	A3	U	16	96
I	17	97	V	2F	AF
J	24	A4	W	11	91
K	25	A5	X	2D	AD
L	26	A6	Y	15	95
M	32	B2	Z	2C	AC

Key	Make	Break	Key	Make	Break
1	02	82	-	0C	8C
2	03	83	=	0D	8D
3	04	84	[	1A	9A
4	05	85	]	1B	9B
5	06	86	;	27	A7
6	07	87	'	28	A8
7	08	88	`	29	A9
8	09	89	\	2B	AB
9	0A	8A	,	33	B3
0	0B	8B	.	34	B4
			/	35	B5

Enhanced Keyboard Keys (101/102 keys)

Control Keys	Make	Break
Alt-PrtSc (SysReq)	54	D4
Ctrl-PrtSc	E0 37	E0 B7
Enter	E0 1C	E0 9C
PrtSc	E0 2A E0 37	E0 B7 E0 AA
Right Alt	E0 38	E0 B8
Right Ctrl	E0 1D	E0 9D
Shift-PrtSc	E0 37	E0 B7
/	E0 35	E0 B5
Pause	E1 1D 45 E1 9D C5	(not typematic)
Ctrl-Pause (Ctrl-Break)	E0 46 E0 C6	(not typematic)

- Keys marked as "not typematic" generate one stream of bytes without corresponding break scan code bytes (actually the break codes are part of the make code).

Key	Normal Mode or Shift w/Numlock Make	Break	Numlock on Make	Numlock on Break
Del	E0 53	E0 D3	E0 2A E0 53	E0 D3 E0 AA
Down arrow	E0 50	E0 D0	E0 2A E0 50	E0 D0 E0 AA
End	E0 4F	E0 CF	E0 2A E0 4F	E0 CF E0 AA
Home	E0 47	E0 C7	E0 2A E0 47	E0 C7 E0 AA
Ins	E0 52	E0 D2	E0 2A E0 52	E0 D2 E0 AA
Left arrow	E0 4B	E0 CB	E0 2A E0 4B	E0 CB E0 AA
PgDn	E0 51	E0 D1	E0 2A E0 51	E0 D1 E0 AA
PgUp	E0 49	E0 C9	E0 2A E0 49	E0 C9 E0 AA
Right arrow	E0 4D	E0 CD	E0 2A E0 4D	E0 CD E0 AA
Up arrow	E0 48	E0 C8	E0 2A E0 48	E0 C8 E0 AA

Key	Left Shift Pressed Make	Left Shift Pressed Break	Right Shift Pressed Make	Right Shift Pressed Break
Del	E0 AA E0 53	E0 D3 E0 2A	E0 B6 E0 53	E0 D3 E0 36
Down arrow	E0 AA E0 50	E0 D0 E0 2A	E0 B6 E0 50	E0 D0 E0 36
End	E0 AA E0 4F	E0 CF E0 2A	E0 B6 E0 4F	E0 CF E0 36
Home	E0 AA E0 47	E0 C7 E0 2A	E0 B6 E0 47	E0 C7 E0 36
Ins	E0 AA E0 52	E0 D2 E0 2A	E0 B6 E0 52	E0 D2 E0 36
Left arrow	E0 AA E0 4B	E0 CB E0 2A	E0 B6 E0 4B	E0 CB E0 36
PgDn	E0 AA E0 51	E0 D1 E0 2A	E0 B6 E0 51	E0 D1 E0 36
PgUp	E0 AA E0 49	E0 C9 E0 2A	E0 B6 E0 49	E0 C9 E0 36
Right arrow	E0 AA E0 4D	E0 CD E0 2A	E0 B6 E0 4D	E0 CD E0 36
Up arrow	E0 AA E0 48	E0 C8 E0 2A	E0 B6 E0 48	E0 C8 E0 36
/	E0 AA E0 35	E0 B5 E0 2A	E0 B6 E0 35	E0 B5 E0 36

- The PS/2 models have three make/break scan code sets. The first set matches the PC & XT make/break scan code set and is

the one
listed here. Scan code sets are selected by writing the value F0
to the keyboard via the 8042 (port 60h). The following is a brief
description of the scan code sets (see the PS/2 Technical Reference
manuals for more information on scan code sets 2 and 3):

/ set 1, each key has a base scan code. Some keys generate extra scan codes to generate artificial shift states. This is similar to the standard scan code set used on the PC and XT.
/ set 2, each key sends one make scan code and two break scan codes bytes (F0 followed by the make code). This scan code set is available on the IBM AT also.
/ set 3, each key sends one make scan code and two break scan codes bytes (F0 followed by the make code) and no keys are altered by Shift/Alt/Ctrl keys.
/ typematic scan codes are the same as the make scan code

- Some Tandy 1000's do not handle Alt key combinations when multiple
shift keys are pressed. The Alt-Shift-H combination loses the Alt.
- extended keys like (F11, F12) can only be read with systems that
have extended keyboard BIOS support (or INT 9 extensions); to
read these special keys on these systems INT 16,10 must be used

- see SCAN CODES KB FLAGS KEYBOARD COMMANDS

INT 16,10 - Extended Wait for Keypress and Read Character (AT+)

```
AH = 10h


on return:
AH = scan code
AL = ASCII character or zero if special function key


- available on AT and PS/2 machines with extended keyboard
support
- similar to INT 16,0
```

INT 16 - Keyboard Scan Codes

Key	Normal	Shifted	w/Ctrl	w/Alt
A	1E61	1E41	1E01	1E00
B	3062	3042	3002	3000
C	2E63	2E42	2E03	2E00
D	2064	2044	2004	2000
E	1265	1245	1205	1200
F	2166	2146	2106	2100
G	2267	2247	2207	2200
H	2368	2348	2308	2300
I	1769	1749	1709	1700
J	246A	244A	240A	2400
K	256B	254B	250B	2500
L	266C	264C	260C	2600
M	326D	324D	320D	3200
N	316E	314E	310E	3100
O	186F	184F	180F	1800
P	1970	1950	1910	1900
Q	1071	1051	1011	1000
R	1372	1352	1312	1300
S	1F73	1F53	1F13	1F00
T	1474	1454	1414	1400
U	1675	1655	1615	1600
V	2F76	2F56	2F16	2F00
W	1177	1157	1117	1100
X	2D78	2D58	2D18	2D00
Y	1579	1559	1519	1500
Z	2C7A	2C5A	2C1A	2C00

Key	Normal	Shifted	w/Ctrl	w/Alt
1	0231	0221		7800
2	0332	0340	0300	7900
3	0433	0423		7A00
4	0534	0524		7B00
5	0635	0625		7C00
6	0736	075E	071E	7D00
7	0837	0826		7E00
8	0938	092A		7F00
9	0A39	0A28		8000
0	0B30	0B29		8100

Key	Normal	Shifted	w/Ctrl	w/Alt
-	0C2D	0C5F	0C1F	8200
=	0D3D	0D2B		8300
[	1A5B	1A7B	1A1B	1A00
]	1B5D	1B7D	1B1D	1B00
;	273B	273A		2700
'	2827	2822		
`	2960	297E		
\	2B5C	2B7C	2B1C	2600 (same as Alt L)
,	332C	333C		
.	342E	343E		
/	352F	353F		

Key	Normal	Shifted	w/Ctrl	w/Alt
F1	3B00	5400	5E00	6800
F2	3C00	5500	5F00	6900
F3	3D00	5600	6000	6A00
F4	3E00	5700	6100	6B00
F5	3F00	5800	6200	6C00
F6	4000	5900	6300	6D00
F7	4100	5A00	6400	6E00
F8	4200	5B00	6500	6F00
F9	4300	5C00	6600	7000
F10	4400	5D00	6700	7100
F11	8500	8700	8900	8B00
F12	8600	8800	8A00	8C00

Key	Normal	Shifted	w/Ctrl	w/Alt
BackSpace	0E08	0E08	0E7F	0E00
Del	5300	532E	9300	A300
Down Arrow	5000	5032	9100	A000
End	4F00	4F31	7500	9F00
Enter	1C0D	1C0D	1C0A	A600
Esc	011B	011B	011B	0100
Home	4700	4737	7700	9700
Ins	5200	5230	9200	A200
Keypad 5		4C35	8F00	
Keypad *	372A		9600	3700
Keypad -	4A2D	4A2D	8E00	4A00
Keypad +	4E2B	4E2B		4E00
Keypad /	352F	352F	9500	A400

Left Arrow	4B00	4B34	7300	9B00
PgDn	5100	5133	7600	A100
PgUp	4900	4939	8400	9900
PrtSc			7200	
Right Arrow	4D00	4D36	7400	9D00
SpaceBar	3920	3920	3920	3920
Tab	0F09	0F00	9400	A500
Up Arrow	4800	4838	8D00	9800

- Some key combinations are not available on all systems. The PS/2 includes many that aren't available on the PC, XT and AT.
- To retrieve the character from a scan code logical AND the word with 0x00FF.
- see INT 16 MAKE CODES

INT 16,0 - Wait for Keypress and Read Character

```
AH = 00


on return:
AH = keyboard scan code
AL = ASCII character or zero if special function key


- halts program until key with a scancode is pressed
- see  SCAN CODES
```

PSP - DOS Program Segment Prefix Layout

Offset	Size	Description
00	word	machine code INT 20 instruction (CDh 20h)
02	word	top of memory in segment (paragraph) form
04	byte	reserved for DOS, usually 0
05	5bytes	machine code instruction long call to the DOS function dispatcher (obsolete CP/M)
06	word	.COM programs bytes available in segment (CP/M)
0A	dword	INT 22 terminate address; DOS loader jumps to this address upon exit; the EXEC function forces a child process to return to the parent by setting this vector to code within the parent (IP,CS)
0E	dword	INT 23 Ctrl-Break exit address; the original INT 23 vector is NOT restored from this pointer (IP,CS)
12	dword	INT 24 critical error exit address; the original INT 24 vector is NOT restored from this field (IP,CS)
16	word	parent process segment addr (Undoc. DOS 2.x+) COMMAND.COM has a parent id of zero, or its own PSP
18	20bytes	file handle array (Undocumented DOS 2.x+); if handle array element is FF then handle is available. Network redirectors often indicate remotes files by setting these to values between 80-FE.
2C	word	segment address of the environment, or zero (DOS 2.x+)
2E	dword	SS:SP on entry to last INT 21 function (Undoc. 2.x+) Ø
32	word	handle array size (Undocumented DOS 3.x+)

```
        34   dword      handle array pointer (Undocumented DOS 3.x+)
        38   dword      pointer to previous PSP (deflt FFFF:FFFF,
Undoc 3.x+) Ø
        3C  20bytes     unused in DOS before 4.01  Ø
        50   3bytes     DOS function dispatcher CDh 21h CBh (Undoc.
3.x+) Ø
        53   9bytes     unused
        5C  36bytes     default unopened FCB #1 (parts overlayed by
FCB #2)
        6C  20bytes     default unopened FCB #2 (overlays part of
FCB #1)
        80   byte       count of characters in command tail;  all
bytes
                        following command name;  also default DTA
(128 bytes)
        81 127bytes     all characters entered after the program
name followed
                        by a CR byte



        - offset 5 contains a jump address which is 2 bytes too low
for
          PSP's created by the DOS EXEC function in DOS 2.x+  Ø
        - program name and complete path can be found after the
environment
          in DOS versions after 3.0.  See offset 2Ch.

        Ø see Bibliography for reference to "Undocumented DOS"
```

INT 20 - Program Terminate

CS = address of PSP

returns nothing

- INT 20 restores the terminate, Ctrl-Break, and critical error
  exit addresses from values in the PSP, flushes all buffers, frees
  memory and returns to DOS via the termination handler address
- does not close FCBs
- this function is no longer recommended, but can be used by
  version of DOS before 2.0, see INT 21,4C and INT 21,0

INT 23 - Control-Break Exit Address

```
        no input data

        - not an interrupt but a pointer to a routine that is
called when a
          DOS function detects a Ctrl-Break or Ctrl-C has been
pressed
          resulting in a "Break condition"
        - a Break condition is detected if DOS's internal Break
flag is
          found set by INT 1B or the next word in the BIOS keyboard
buffer
          is a scan code representing one of the Break key
combinations.
          If a matching scan code is found, the keyboard buffer
head pointer
          is incremented by 2 (which effectively removes the ^C
keycode), the
          ^C is printed and then INT 23 is called.
        - disabling the abort is easy, but to disable the ^C from
appearing
          you must either not use any of the DOS I/O functions that
check
          for Break or you must trap the Ctrl-C keypress from
within INT 9.
          Another method is to scan the BIOS keyboard buffer and
delete any
          Break key combinations.
        - Ctrl-Break empties the keyboard buffer by resetting the
keyboard
          head and tail pointers then places a 0000h at the queue
head.  It
          then sets an internal "Break" flag.  DOS subfunctions
that check
          for Break see this and then issue INT 23h.  DOS does not
detect
          INT 1B using the keyboard buffer, but uses it's own
internal flag.
        - Alt-Keypad-3 and Ctrl-2 also result in this interrupt
        - Ctrl-C places 2E03h in the BIOS keyboard buffer while
Ctrl-2
          places 0300h and Alt-Keypad-3 places 0003h;  none of
```

these key
          combinations empty the keyboard like Ctrl-Break but all
result
          in a ^C being displayed; note that all three produce
scan codes
          containing '03'
        - since DOS checks only the first word of the keyboard
buffer
          Ctrl-C, Ctrl-2 and Alt-Keypad-3 are only detected if they
are the
          first word in the buffer while Ctrl-Break is detected via
the
          internal flag and takes effect as soon as it's detected.
        - do not execute this interrupt directly
        - see also INT 9

XFCB - Extended DOS File Control Block

Offset	Size	Description
00	byte	flag containing FF if this is an extended FCB
01	5bytes	reserved
06	byte	directory attribute byte (see FILE ATTRIBUTE)
07	byte	drive number (0 for default drive, 1 = A:, 2 = B:)
08	8bytes	filename, left justified with trailing blanks
16	3bytes	filename extension, left justified w/blanks
19	word	current block number relative to file start (0 based)
21	word	logical record size in bytes
23	dword	file size in bytes
27	word	date the file was created or last updated:

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0| 21,20
 | | | | | | | | | | `---------- day 1-31
 | | | | | | `----------------- month 1-12
 `--------------------------- year + 1980
```

29	16bytes	reserved for system use
39	byte	relative record number within current BLOCK
40	4bytes	relative record number from start of file (0 based)

DOS File Attributes

Directory Attribute Flags

```
|7|6|5|4|3|2|1|0| Directory Attribute Flags
 | | | | | | | `--- 1 = read only
 | | | | | | `---- 1 = hidden
 | | | | | `----- 1 = system
 | | | | `------ 1 = volume label  (exclusive)
 | | | `------- 1 = subdirectory
 | | `-------- 1 = archive
 `------------ unused
```

Directory Time Format

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0| Directory Time Format
 | | | | | | | | | | `------------ seconds (2 second
increments)
 | | | | `---------------------- minutes (0-59)
 `------------------------------ hours (0-23)
```

Directory Date Format

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0| Directory Date Format
 | | | | | | | | | | `------------ Day (1-31)
 | | | | | | `------------------ Month (1-12)
 `------------------------------ Year (less 1980)
```

- with respect to the findfirst() and findnext() functions normal
  files are always included along with the requested attributes
  except when the LABEL attribute is requested. It's up to the
  programmer to determine which files are match the requested
  attributes.

TC: int findfirst(const char *path, struct ffblk *ffblk, int attr) MSC unsigned _dos_findfirst (char *path, unsigned attr, struct find_t *fileinfo) - prototype in dir.h, also include dos.h (MS C: dos.h only) - gets disk directory via DOS 0x4E, where attr is: **Turbo C Microsoft C** FA_RDONLY _A_RDONLY FA_HIDDEN _A_HIDDEN FA_SYSTEM _A_SYSTEM FA_LABEL _A_VOLID FA_DIREC _A_SUBDIR FA_ARCH _A_ARCH **struct ffblk (Turbo C) struct find_t (MSC)** { { char ff_reserved[21]; char reserved[21]; char ff_attrib; char attrib; unsigned ff_ftime; unsigned wr_time; unsigned ff_fdate; unsigned wr_date; long ff_fsize; long size; char ff_name[13]; char name [13]; }; }; - attributes can be logically OR'ed - struct ffblk is defined in Turbo C's dir.h - struct find_t is defined in MS C's dos.h - updates DTA, provisions for saving/setting DTA should be made - normal files are always included along with files that match the requested attributes except when the LABEL attribute is requested. It's up to the programmer to determine which actually match the requested attributes. - returns 0 if OK, -1 if no matching files found or on error - see FILE ATTRIBUTES, FINDNEXT, GETDTA and SETDTA

DTA - Disk Transfer Area (partially undocumented)

DTA contains data, of which the first 21 bytes (00-15h) are known
as being "reserved for DOS use on subsequent find next calls"

Offset	Size	Description
00	byte	attribute of search (undocumented)
01	byte	drive used in search (undocumented)
02	11bytes	search name used (undocumented)
0D	word	directory entry number (0 based, DOS 3.x+, undoc.)
0F	word	starting cluster number of current directory zero for root directory (DOS 3.x+, undocumented)
11	word	reserved (undocumented)
13	word	starting cluster number of current directory zero for root directory (DOS 2.x+, undocumented)
15	byte	attribute of matching file
16	word	file time (see FILE ATTRIBUTES)
18	word	file date (see FILE ATTRIBUTES)
1A	word	file size
1E	13bytes	ASCIIZ filename and extension in the form NAME.EXT with blanks stripped

- fields of DTA change dependent upon function call in progress
- the DTA cannot span a 64K segment boundary
- for compatibility with CP/M the default DTA is at offset 80h
  in the PSP; this area is also used for the command tail. To
  avoid collision, set another DTA (INT 21,1A) or preserve the
  command tail before using FCB function calls

- see INT 21,4E INT 21,1A INT 21,2F

INT 21,4E - Find First Matching File

```
AH = 4E
CX = attribute used during search  (see FILE ATTRIBUTES)
DS:DX = pointer to ASCIIZ filespec, including wildcards


on return:
AX = error code if CF set  (see DOS ERROR CODE)
DTA = data returned from call in the format:
```

Offset	Size	Description
00	byte	attribute of search (undocumented)
	byte	drive letter used in search (DOS 3.1-4.x, undocumented)
01	byte	drive letter used in search (undocumented)
	11bytes	search name used (DOS 3.1-4.x, undocumented)
02	11bytes	search name used (undocumented)
0C	byte	attribute of search (DOS 3.1-4.x, undocumented)
0D	word	directory entry number (0 based, undocumented)
0F	word	starting cluster number of current directory; zero
		for root directory (DOS 3.2+, undocumented)
	dword	pointer to DTA (DOS 2.x-3.1, undocumented)
11	word	reserved
13	word	starting cluster number of current directory; zero
		for root directory (DOS 2.x+, undocumented)
15	byte	attribute of matching file
16	word	file time (see FILE ATTRIBUTES)
18	word	file date (see FILE ATTRIBUTES)
1A	word	file size
1E	13bytes	ASCIIZ filename and extension in the form NAME.EXT
		with blanks stripped

```
- returns information on first file matching specifications
- use INT 21,4F to retrieve following file matches
- DOS 2.x cannot find . and .. entries, while DOS 3.x can
```

```
unless
          they represent the root directory
        - character devices return a zero for size, time and date
in DOS 2.x,
          while DOS 3.0 returns a 40h attribute and current time
and date.
        - multiple calls to this function with a character device
will
          result in unpredictable results
        - normal files are always included along with files that
match the
          requested attributes except when the LABEL attribute is
requested.
          DOS 2.x returns all normal files when label is specified
but 3.x
          doesn't.  It's up to the programmer to determine which
actually
          match the requested attributes.
        - bit 8 of CX (file attributes) indicates Novell Netware
shareable
        - see   INT 21,1A
```

DOS Error Codes

Of the following error codes, only error codes 1-12 are returned in AX upon exit from interrupt 21 or 24; The rest are obtained by issuing the "get extended error" function call; see INT 21,59

01 Invalid function number
02 File not found
03 Path not found
04 Too many open files (no handles left)
05 Access denied
06 Invalid handle
07 Memory control blocks destroyed
08 Insufficient memory
09 Invalid memory block address
0A Invalid environment
0B Invalid format
0C Invalid access mode (open mode is invalid)
0D Invalid data
0E Reserved
0F Invalid drive specified
10 Attempt to remove current directory
11 Not same device
12 No more files
13 Attempt to write on a write-protected diskette
14 Unknown unit
15 Drive not ready
16 Unknown command
17 CRC error
18 Bad request structure length
19 Seek error
1A Unknown media type
1B Sector not found
1C Printer out of paper
1D Write fault
1E Read fault
1F General failure
20 Sharing violation
21 Lock violation
22 Invalid disk change
23 FCB unavailable
24 Sharing buffer overflow

```
25  Reserved
26  Unable to complete file operation (DOS 4.x)
27-31 Reserved
32  Network request not supported
33  Remote computer not listening
34  Duplicate name on network
35  Network name not found
36  Network busy
37  Network device no longer exists
38  NetBIOS command limit exceeded
39  Network adapter error
3A  Incorrect network response
3B  Unexpected network error
3C  Incompatible remote adapter
3D  Print queue full
3E  No space for print file
3F  Print file deleted
40  Network name deleted
41  Access denied
42  Network device type incorrect
43  Network name not found
44  Network name limit exceeded
45  NetBIOS session limit exceeded
46  Temporarily paused
47  Network request not accepted
48  Print or disk redirection is paused
49-4F Reserved
50  File already exists
51  Reserved
52  Cannot make directory entry
53  Fail on INT 24
54  Too many redirections
55  Duplicate redirection
56  Invalid password
57  Invalid parameter
58  Network device fault
59  Function not supported by network (DOS 4.x)
5A  Required system component not installed (DOS 4.x)
```

DOS Error Code/Classes

Error Classes

```
        01  Out of resource, out of space, channel, etc
        02  Temporary situation, not an error, ex: file lock
        03  Authorization, permission denied
        04  Internal, system detected internal error
        05  Hardware failure, serious problem related to hardware
        06  System failure, ex: invalid configuration
        07  Application error, inconsistent request
        08  Not found, file/item not found
        09  Bad format, file/item in invalid format
        0A  Locked, file/item interlocked
        0B  Media failure, ECC/CRC error, wrong or bad disk
        0C  Already exists, collision with existing item
        0D  Unknown, classification doesn't exist or is
inappropriate
```

DOS Error Code/Action Codes and Locus

Error Action Codes (in BL)

```
        01  retry, attempt a few more times and re-prompt
        02  delay retry, retry a few more times after a pause
        03  re-enter input, prompt user to re-enter input
        04  abort with cleanup, orderly abort and shutdown
        05  immediate abort, exit immediately without cleanup
        06  ignore error
        07  user intervention, retry after user fixes the problem
```

Error Locus (in CH)

```
        01  unknown
        02  block device
        03  network
        04  serial device
        05  memory
```

INT 21,59 - Get Extended Error Information (DOS 3.0+)

```
AH = 59h
BX = 00 for versions  3.0, 3.1, 3.2


on return:
AX = extended error code (see DOS ERROR CODES)
   = 0 if no error
BH = error class
BL = suggested action
CH = locus


- may be called after any  INT 21  function or from
  INT 24 when an error is returned
- must be called immediately after the error occurs
- registers CX, DX, DI, SI, BP, DS and ES are destroyed
```

INT 21,F8 - Set OEM Int 21 Handler (functions F9-FF, Undocumented)

```
        AH = F8
        DS:DX = pointer to OEM handler for INT 21H calls F9 through
FF
              = FFFF:FFFF to reset to original handlers


        - DOS is set up to allow ONE handler for all 7 function
calls.
          Any call to these handlers will set the CF and AX
          will contain 1 if they are not initialized.   The handling
          routine is passed all registers just as the user set them.
          The handler should exit with an IRET.
```

INT 24 - Critical Error Handler

```
        no input data

        on entry to INT 24:
        AH = bit 7 of register AH is set to one if other than disk
error
           = bit 7 of register AH is set to zero if disk error
             AL = failing drive number
             AH = bits 0-2 indicate affected disk area and read/
write status

                    |5|4|3|2|1|0|  AH
                     | | | | | `---- read = 0, write = 1
                     | | | `------- 00=DOS, 01=FAT, 10=DIR, 11=data
area
                     | | `-------- 0=FAIL not allowed, 1=FAIL allowed
                     | `---------- 0=RETRY not allowed, 1=RETRY allowed
                     `------------ 0=IGNORE not allowed, 1=IGNORE
allowed

        DI = error code in lower byte
        BP:SI = pointer to device header control block where
additional
                information about the error can be found


        on exit:
        AL = action code determining exit behavior
           = 00  ignore error
           = 01  retry
           = 02  terminate through INT 23
           = 03  fail system call (DOS 3.x+)
```

Error Codes in low order byte of DI:

```
          00  write protect error
          01  unknown unit
          02  drive not ready
          03  unknown command
          04  data error (bad CRC)
          05  bad request structure length
```

```
  06  seek error
  07  unknown media type
  08  sector not found
  09  printer out of paper
  0A  write fault
  0B  read fault
  0C  general failure
```

Users Stack at Entry to Error Handler:

```
(top of stack)
    IP          DOS code next instruction pointer
    CS
    FLAGS       DOS's flags
    AX -\
    BX   \
    CX    \
    DX     \
    SI      |- User's registers at entry to INT 21 function
    DI     /
    BP    /
    DS   /
    ES -/
    IP          User code next instruction pointer
    CS
    FLAGS
```

- on entry registers are setup for a retry operation
- user routine must issue an IRET or simulate an IRET
- INT 21,0 through INT 21,C and INT 21,59 can safely be invoked
  from the handler. Other calls may destroy DOS's stack
- handler must preserve register SS,SP,DS,ES,BX,CX,DX
- choosing ignore can show side effects, since it causes DOS
  to continue as if it the call were successful
- if an improper action code is specified in DOS 3.x it is changed:
  if IGNORE is invalidly specified, action is converted to FAIL
  if RETRY is invalidly specified, action is converted to FAIL
  if FAIL is invalidly specified, action is converted to ABORT
- IGNORE requests are converted to FAIL for FAT and DIR disk
  errors and network critical errors
- if the user routine wishes to handle the error instead of

passing
          it to DOS, it must restore the user program registers from the
          stack and remove all but the last 3 words from the stack (FLAGS,
          CS, IP) and issue an IRET
        - do not execute this interrupt directly
        - INDOS flag is cleared on INT 24 (see INT 21,34)

INT 21,0 - Program Terminate

```
AH = 00
CS = PSP segment address


returns nothing
```

- restores the terminate, Ctrl-Break, and critical error exit addresses, flushes all buffers, frees memory and returns to DOS via the termination handler address
- does not close FCBs
- this function is no longer recommended, but can be used by version of DOS before 2.0, see INT 21,4C and INT 20

INT 21,4C - Terminate Process With Return Code

```
        AH = 4C
        AL = return code (for batch files)


        returns nothing


        - approved method of program termination
        - restores the terminate, Ctrl-Break, and critical error
exit
          addresses, flushes all buffers, frees memory and returns
to
          DOS via the termination handler address
        - does not close FCBs
        - this function is not supported in versions of DOS before
2.x,
          so use INT 21,0  or    INT 20  to exit.


        - see also  INT 27   INT 21,31
```

INT 27 - Terminate and Stay Resident

```
DX = offset of last byte in program to remain resident plus
1
CS = segment of PSP


returns nothing


- terminates process without closing open files
- memory beyond the offset in DX is freed


- see   INT 21,31
```

INT 21,31 - Terminate Process and Remain Resident

```
AH = 31h
AL = exit code (returned to batch files)
DX = memory size in paragraphs to reserve


returns nothing
```

- preferred method for Terminate and Stay Resident programs
- terminates process without releasing allocated memory and without closing open files
- attempts allocation of memory specified in DX from memory allocated by DOS at startup. INT 21,48 memory allocation is not affected

- see INT 27

INT 21,48 - Allocate Memory

```
AH = 48h
BX = number of memory paragraphs requested


on return:
AX = segment address of allocated memory block (MCB + 1para)
   = error code if CF set  (see DOS ERROR CODES)
BX = size in paras of the largest block of memory available
     if CF set, and AX = 08 (Not Enough Mem)
CF = 0 if successful
   = 1 if error


- returns segment address of allocated memory block AX:0000
- each allocation requires a 16 byte overhead for the MCB
- returns maximum block size available if error

- see  INT 21,49,  INT 21,4A
```

MCB - DOS Memory Control Block Format

Offset	Size	Description
00	byte	'M' 4Dh member of a MCB chain, (not last) 'Z' 5Ah indicates last entry in MCB chain other values cause "Memory Allocation Failure" on exit
01	word	PSP segment address of MCB owner (Process Id) possible values: 0 = free 8 = Allocated by DOS before first user pgm loaded other = Process Id/PSP segment address of owner
03	word	number of paras related to this MCB (excluding MCB)
05	11bytes	reserved
08	8bytes	ASCII program name, NULL terminated if less than max length (DOS 4.x+)
10	nbytes	first byte of actual allocated memory block

- to find the first MCB in the chain, use INT 21,52
- DOS 3.1+ the first memory block contains the DOS data segment
  ie., installable drivers, buffers, etc
- DOS 4.x the first memory block is divided into subsegments,
  with their own memory control blocks; offset 0000h is the first
- the 'M' and 'Z' are said to represent Mark Zbikowski
- the MCB chain is often referred to as a linked list, but technically isn't

DOS 4.x Initial Data Segment Subsegment Control Blocks:

Offset	Size	Description
00	byte	subsegment type 'D' device driver 'E' device driver appendage 'I' Installable File System driver 'F' FILES= control block storage area (for FILES>5) 'X' FCBS= control block storage area, if present 'C' BUFFERS EMS workspace area if BUFFERS / X is used 'B' BUFFERS= storage area 'L' LASTDRIVE= current directory structure array 'S' STACKS= code/data area, if present (see below)
01	word	paragraph of subsegment start
03	word	subsegment size in paragraphs
05	3bytes	unused
08		types "D" and "I", filename of driver loaded driver

- see INT 21,48 INT 21,49 INT 21,4A

INT 21,52 - Get Pointer to DOS "INVARS" (Undocumented)

```
        AH = 52h

        on return:
        ES:BX = pointer to DOS "invars", a table of pointers used
by DOS.
                Known "invars" fields follow (varies with DOS
version):
```

Offset	Size	Description
-12	word	sharing retry count (DOS 3.1-3.3)
-10	word	sharing retry delay (DOS 3.1-3.3)
-8	dword	pointer to current disk buffer (DOS 3.x)
-4	word	pointer in DOS code segment of unread CON input; 0 indicates no unread input (DOS 3.x)
-2	word	segment of first Memory Control Block (MCB)
00	dword	pointer to first DRIVE PARAMETER TABLE (A:) in chain
04	dword	pointer to DOS System File Table (SFT)
08	dword	pointer to $CLOCK device driver
0C	dword	pointer to CON device driver
10	byte	number of logical drives in system
11	word	maximum bytes/block of any block device
13	dword	pointer to DOS cache buffer header
17	18bytes	NUL device header, first 4 bytes of device header point to the next device in device chain

DOS 3.0 Specific Information

Offset	Size	Description
10	byte	number of block devices
11	word	max sector size used by DOS prior to loading

```
block
                    device drivers;  limits allowable device
sector size
          13   dword  pointer to DOS cache buffer header
          17   dword  pointer to drive information table
          1B   byte   value of LASTDRIVE command in CONFIG.SYS
          1C   dword  pointer to STRING= workspace area
          20   word   size of STRING area (CONFIG.SYS value of
STRING=x)
          22   dword  FCB file table pointer
          26   word   number of protected FCBs
          28  18bytes NUL device header, first 4 bytes of device
header
                      point to the next device in device chain
```

DOS 3.1-3.3 Specific Information

Offset	Size	Description
10	word	max sector size used by DOS prior to loading block device drivers; limits allowable device sector size
12	dword	pointer to DOS cache buffer header
16	dword	pointer to drive information table
1A	dword	FCB file table pointer
1E	word	number of protected FCBs
20	byte	number of block devices
21	byte	value of LASTDRIVE command in CONFIG.SYS
22	18bytes	NUL device header, first 4 bytes of device header point to the next device in device chain
34	byte	number of JOIN'ed drives

DOS 4.x Specific Information

Offset	Size	Description

```
          10   word   max sector size used by DOS prior to loading
block
                      device drivers;  limits allowable device
sector size
          12   dword  pointer to DOS cache buffer header
          16   dword  pointer to drive information table
          1A   dword  FCB file table pointer
          1E   word   number of protected FCBs
          20   byte   number of block devices
          21   byte   value of LASTDRIVE command in CONFIG.SYS
(default 5)
          22  18bytes NUL device header, first 4 bytes of device
header
                      point to the next device in device chain
          34   byte   number of joined drives
          35   word   pointer within IBMDOS code segment to list of
special
                      program names
          37   dword  pointer to resident IFS utility function
          3B   dword  pointer to chain of installable file system
drivers
          3F   word   the x in BUFFERS x,y (rounded up to multiple
of 30
                      if EMS is used)
          41   word   the y in BUFFERS x,y
          43   byte   boot drive (1=A:)
          44   byte   ???
          45   byte   extended memory size in K bytes


        - see Bibliography references for Bernd Schemmer &
"Undocumented DOS"
        - see   INDOS  INT 21,5D
```

DPT/DPB - Drive Parameter Table / Disk Parameter Block

Offset	Size	Description
00	byte	drive (0 = A, 1 = B)
01	byte	unit within device, usually equals drive (ramdisk=0)
02	word	bytes per sector
04	byte	sectors per cluster minus 1
05	byte	sectors per cluster (times to shift left or x2)
06	word	number of sectors before FAT (boot sectors)
08	byte	number of FAT copies
09	word	number of root directory entries
0B	word	number of first data sector
0D	word	total number of clusters plus 1
0F	byte	number of sectors used by first FAT
		The following fields are DOS version dependant
10	word	number of first sector in root directory
12	dword	far pointer to current disk device header
16	byte	media descriptor byte (see MEDIA DESCRIPTOR)
17	byte	zero if disk accessed, (default=FF, must rebuild DPB)
18	dword	far pointer to next drive parameter table; offset is set to FFFFh if last block in chain
1C	word	current directory cluster number; 0=root
1E	64bytes	ASCIIZ current working directory

Fields differing in DOS 3.x

Offset	Size	Description
1C	word	starting cluster for free space search
1E	64bytes	number of free clusters; FFFFh = unknown

Fields differing in DOS 4.x

Offset	Size	Description
11	word	first sector of root directory
13	dword	far pointer to current disk device header
17	byte	media descriptor byte (see MEDIA DESCRIPTOR)
18	byte	zero if disk accessed, (default=FF, must rebuild DPB)
19	dword	pointer to next drive parameter table; offset is set to FFFFh if last block in chain
1D	word	starting cluster for free space search
1F	word	number of free clusters, FFFFh = unknown

- INT 21,32 which is used to read this data resets accessed byte at
  offset 17h or 18h (depending on DOS version)
- dword at offset 12h & 13h aren't supported in OS/2 compatability box
- sector references are DOS logical sectors
- see Disk Base Table BPB INT 21,1F INT 21,32

FAT - File Allocation Table

12 Bit	Meaning	16 Bit
000	free space	0000
FF1-FF7	bad track marking	FFF1-FFF7
FF8-FFE	may be used to mark end of a file chain	FFF8-FFFE
FFF	standard marker for end of a file chain	FFFF

- the FAT is implemented as an array containing a linked list for each file; the files directory entry has a pointer to the first cluster which contains the cluster number of the next cluster in the chain until the pointer contained is FFFh (12 bit FAT) and FFFFh (16 bit FAT) marking end of file
- DOS maintains two copies of the FAT, but does not use the second copy for anything other than a mirror image of the first; CHKDSK doesn't even read the second FAT
- disks with FF1h clusters and above use 16 bit FAT tables, disk with less use 12 bit FAT tables
- DOS 4.x did not change the size of the cluster number as some suggest, but instead increased the size of the sector number
- bytes 0 of the FAT contains the Media Descriptor Byte

Calculating 12 bit FAT Entries

1. Get starting cluster from directory entry.

2. Multiply the cluster number just used by 1.5

3. The whole part of the product is the offset into the FAT,

of the entry that maps to the cluster in the directory. This entry contains the number of the next cluster.

4. Move the word at the calculated FAT into a register.

5. If the last cluster used was an even number, keep the low order 12 bits of the register, otherwise, keep the high order 12 bits.

6. If the resultant 12 bits are (0FF8h-0FFFh) no more clusters are in the file. Otherwise, the next 12 bits contain the cluster number of the next cluster in the file.

Calculating 16 Bit FAT Entries

1. Get the starting cluster of the file from the directory.

2. Multiply the cluster number found by 2.

3. Load the word at the calculated FAT offset into a register.

4. If the 16 bits are (0FFF8h-0FFFFh) no more clusters are in the file. Otherwise, the 16 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, similar to that used by DEBUG, int 25h and 26h):

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

- see MEDIA DESCRIPTOR

Media Descriptor Byte

Media Descriptor Byte Layout:

```
|7|6|5|4|3|2|1|0| IBM Media Descriptor Byte
 | | | | | | | `--- 1 = 2 sided, 0 = not 2 sided
 | | | | | | `---- 1 = 8 sector, 0 = not 8 sector
 | | | | | `----- 1 = removable, 0 = not removable
 `--------------- must be set to 1
```

F8 Hard disk
Double sided 18 sector diskette PS/2 1.44 Mb. DSQD

F9 5¬ Double sided, High Density diskette (15 sector)
3« Double Sided, Double High Density diskette (9 sector)

FA Ram disk (not all Ramdisks use this)

FC 5¬ Single Sided, Double Density diskette (9 sector)
8 inch Double Sided, Single Density diskette

FD 5¬ Double Sided, Double Density (9 sector)
8 inch Double Sided Single Density (26 sector) (IBM 3740 format) 128 bytes per sector, soft sector, 4 sectors cluster, 4 reserved sectors, 2 FATs, 68 directory entries, 77*26*2 sectors.

FE 5¬ Single Sided, Double Density diskette (8 sector)
8 inch Single Sided, Single Density diskette (26 sector) (IBM 3740 format) 128 bytes per sector, soft sector, 4 sectors per cluster, 1 reserved sector, 2 FATs. 68 directory entries, 77*26 sectors.
8 inch Double Sided, Double Density (8 sector), 1024 bytes per sector, soft sector, 1 sector per cluster, 1 reserved

sector, 2 FATs, 192 directory entries, 77*8*2 sectors. To distinguish from 8 inch SS/SD attempt read of side 2.

FF 5¬ Double Sided, Double Density diskette (8 sector)

- see also INT 21,1B INT 21,1C FAT

INT 21,1B - Get Allocation Table Information

```
        AH = 1B


        on return:
        AL = sectors per cluster
        CX = bytes per sector
        DX = clusters on disk
        DS:BX = pointer to Media Descriptor Byte found in FAT


        - retrieves information on capacity and format of default
drive
        - DS:BX can be used to determine if drive is RAMDISK or
removable
        - see  INT 21,1C
```

INT 21,1C - Get Allocation Table Info for Specified Drive

```
AH = 1C
DL = drive number (0 for default, 1 = A:, Z: = 26)


on return:
AL = sectors per cluster
CX = bytes per sector
DX = clusters on disk
DS:BX = pointer to Media Descriptor Byte found in FAT


- retrieves information on capacity and format of specified
drive
- DS:BX can be used to determine if drive is RAMDISK or
removable
- see  INT 21,1B
```

INT 21,32 - Get Pointer to Drive Parameter Table (Undocumented)

```
        AH = 32h
        DL = drive (0=default, 1=A:, 2=B:, 3=C:, ...)


        on return:
        AL =  FF if the drive number in DL was invalid
        DS:BX = address of drive parameter table (DPT)


        - available since DOS 2.0
        - used by DOS commands CHKDSK and RECOVER
        - forces a media check, which clears byte DS:[BX+17h] or DS:
[BX+18h]
          in the Drive Parameter Table
        - actually accesses the disk, causing a critical error if a
disk
          error occurs
        - can be used to determine if a drive is SUBST'ed by
comparing
          DS:[BX+1] and DS:[BX] for a match.  If not equal, then the
          drive is possibly SUBST'ed (though not guaranteed, this
may also
          indicate a Bernoulli box)
        - can be used to determine if a drive is a RAM disk; if the
disk is
          NOT removable and (DS:[BX+1] == 0), then the disk is a
RAM disk
          (see IOCTL,0 bit number 0Bh to determine if the disk is
removable)
```

INT 21,44,0 / IOCTL,0 - Get Device Information

```
AH = 44h
AL = 00
BX = handle (must be an opened device)


on return
AX = error code if CF set  (see DOS ERROR CODES)
DX = device information  (see tables below)


|F|E|D|C|B|A-8|7|6|5-0|  DX  Block Device Information
 | | | | |  |  | |  `----- drive number (0=A:,1=B:)
 | | | | |  |  | `------- 0 = file has been written
 | | | | |  |  `--------- 0 = disk file; 1 = character device
 | | | | |  `------------ reserved, must be zero
 | | | | `-------------- 1 = media not removable
 | | | `---------------- 1 = network device (DOS 3.x+)
 | | `------------------ 1 = reserved
 | `-------------------- 1 = don't update file time or date (DOS
4.x+)
 `---------------------- 1 = file is remote (DOS 3.x+)

|F|E|D|C|B|A-8|7|6|5|4|3|2|1|0|  DX  Character Device
Information
 | | | | |  |  | | | | | | `---- 1 = standard input device
 | | | | |  |  | | | | | `---- 1 = standard output device
 | | | | |  |  | | | | `---- 1 = NUL device
 | | | | |  |  | | | `---- 1 = clock device
 | | | | |  |  | | `---- uses DOS INT 29 for fast
character output
 | | | | |  |  | `---- 1 = binary mode, 0 = translated
 | | | | |  |  `---- 0 = end of file on input
 | | | | |  `------- 1 = character device, 0 if disk file
 | | | | `---------- reserved
 | | | `------------ 1 = media not removable
 | | `------------- 1 = network device (DOS 3.x+)
 | `--------------- reserved
 `----------------- 1 = supports IOCTL, via functions 2 & 3
```

- BIT 7 of register DX can be used to detect if STDIN/STDOUT is
  redirected to/from disk; if a call to this function has DX BIT 7
  set it's not redirected from/to disk; if it's clear then it is
  redirected to/from disk
- BIT B of register DX can be used to determine if a drive is
  removable.

INT 29 - DOS Fast Character I/O (Undocumented 2.x+)

```
AL = character to write


returns nothing ???
```

- used by DOS device drivers to output data to the screen if bit 4 of the device attribute word is set
- similar to INT 21,2 and INT 21,9 but much faster
- simply calls the BIOS INT 10,E in DOS versions 2.x and 3.x
- advances the cursor after character output

INT 21,2F - Get Disk Transfer Address (DTA)

```
AH = 2F


on return:
ES:BX = pointer to current DTA


- returns the DTA address
- the default DTA is a 128 byte block at address 80h in the
  Program Segment Prefix (PSP).  This area also contains the
  command tail at program startup it must be saved or the
DTA
  must be relocated before use to preserve the command tail

- see   INT 21,1A
```

INT 21,1A - Set Disk Transfer Address (DTA)

```
AH = 1A
DS:DX = pointer to disk transfer address (DTA)


returns nothing


- specifies the disk transfer address to DOS
- DTA cannot overlap 64K segment boundary
- offset 80h in the PSP is a 128 byte default DTA supplied
  by DOS upon program load
- use of the DTA provided by DOS will result in the loss
  of the program command tail which also occupies the 128
  bytes starting at offset 80h of the PSP
- see   INT 21,2F
```

INT 21,9 - Print String

```
AH = 09
DS:DX = pointer to string ending in "$"


returns nothing


- outputs character string to STDOUT up to "$"
- backspace is treated as non-destructive
- if Ctrl-Break is detected, INT 23 is executed
```

INT 10,E - Write Text in Teletype Mode

```
AH = 0E
AL = ASCII character to write
BH = page number (text modes)
BL = foreground pixel color (graphics modes)


returns nothing


- cursor advances after write
- characters BEL (7), BS (8), LF (A), and CR (D) are
  treated as control codes
- for some older BIOS (10/19/81), the BH register must point
  to the currently displayed page
- on CGA adapters this function can disable the video
signal while
  performing the output which causes flitter.
```

DBT - Disk Base Table (BIOS INT 13)

Offset	Size	Description
00	byte	specify byte 1; step-rate time, head unload time
01	byte	specify byte 2; head load time, DMA mode
02	byte	timer ticks to wait before disk motor shutoff
03	byte	bytes per sector code: 0 - 128 bytes 2 - 512 bytes 1 - 256 bytes 3 - 1024 bytes
04	byte	sectors per track (last sector number)
05	byte	inter-block gap length/gap between sectors
06	byte	data length, if sector length not specified
07	byte	gap length between sectors for format
08	byte	fill byte for formatted sectors
09	byte	head settle time in milliseconds
0A	byte	motor startup time in eighths of a second

BPB - BIOS Parameter Block

Offset	Size	Description
00	word	sector size in bytes
02	byte	sectors per cluster (allocation unit size)
03	word	number of reserved sectors
05	byte	number of FATs on disk
06	word	number of root directory entries (directory size)
08	word	number of total sectors; if partition > 32Mb then set to zero and dword at 15h contains the actual count
0A	byte	media descriptor byte (see MEDIA DESCRIPTOR)
0B	word	sectors per FAT
		Additional/different fields for DOS 3.0+
0D	word	sectors per track
0F	word	number of heads
11	word	number of hidden sectors
15	11bytes	reserved
		Additional/different fields for DOS 4.0+
15	dword	number of total sectors if offset 8 is zero
19	6bytes	reserved
1F	word	number of cylinders
21	byte	device type
22	word	device attributes

- located in the boot sector at offset 0Bh

- see BOOT SECTOR

Boot Sector (since DOS 2.0)

Offset	Size	Description
00	3bytes	jump to executable code
03	8bytes	OEM name and version
0B	word	bytes per sector
0D	byte	sectors per cluster (allocation unit size)
0E	word	number of reserved sectors (starting at 0)
10	byte	number of FAT's on disk
11	word	number of root directory entries (directory size)
13	word	number of total sectors (0 if partition > 32Mb)
15	byte	media descriptor byte (see MEDIA DESCRIPTOR)
16	word	sectors per FAT
18	word	sectors per track (DOS 3.0+)
1A	word	number of heads (DOS 3.0+)
1C	word	number of hidden sectors (DOS 3.0+)
20	dword	(DOS 4+) number of sectors if offset 13 was 0
24	byte	(DOS 4+) physical drive number
25	byte	(DOS 4+) reserved
26	byte	(DOS 4+) signature byte (29h)
27	dword	(DOS 4+) volume serial number
2B	11bytes	(DOS 4+) volume label
36	8bytes	(DOS 4+) reserved

- implementation format not guaranteed in all OEM DOS releases
- BIOS expects a boot sector of 512 bytes
- DOS 3.2 began reading BIOS Parameter Block (BPB) information from the boot sector, previous versions used only the media byte in FAT
- DOS 4.x added offsets 20-3Dh and offset 20h determines the number of sectors if offset 13h is zero
- hard disks have a master boot record and partition boot records; the master boot record and Disk Partition Table (DPT)

share the
same sector

Disk Partition Table (Fixed disk boot record)

Offset	Represents: (see format below)
01BE	Partition 1 data table (16 bytes)
01CE	Partition 2 data table (16 bytes)
01DE	Partition 3 data table (16 bytes)
01EE	Partition 4 data table (16 bytes)
01FE	Signature (hex 55 AA, 2 bytes)

Offset from beginning of partition data shown above:

Offset	Size	Description
00	byte	boot indicator
01	byte	beginning sector head number
02	byte	beginning sector (2 high bits of cylinder #)
03	byte	beginning cylinder# (low order bits of cylinder #)
04	byte	system indicator
05	byte	ending sector head number
06	byte	ending sector (2 high bits of cylinder #)
07	byte	ending cylinder# (low order bits of cylinder #)
08	dword	number of sectors preceding the partition
0B	dword	number of sectors in the partition

Boot indicator (BYTE)

00 - non-bootable partition
80 - bootable partition (one partition only)

System Indicator (BYTE)

00 - unknown operating system
01 - DOS with 12 bit FAT, 16 bit sector number
02 - XENIX
04 - DOS with 16 bit FAT, 16 bit sector number
05 - DOS Extended partition (DOS 3.3+)
06 - DOS 4.0 (Compaq 3.31), 32 bit sector number
51 - Ontrack extended partition
64 - Novell

```
75 - PCIX
DB - CP/M
FF - BBT
```

Signature

Hex 55AA marks the end of valid boot sector. This is also required in each of the partition boot records.

Sector/Cylinder

2 bytes are combined to a word similar to INT 13:

```
|7|6|5|4|3|2|1|0| 1st byte  (sector)
 | | `------------ Sector offset within cylinder
 `---------------- High order bits of cylinder #

|7|6|5|4|3|2|1|0| 2nd byte  (cylinder)
 `----------------- Low order bits of cylinder #
```

- all partitions begin on sector 1 head 0, except the first partition which follows the disk's master boot record and begins in sector 2
- some of this information may vary with some variants of DOS 3.2 and DOS 3.3 that use their own sectoring scheme for large disks
- see INT 21,32 Disk Partition Table

INT 21,1F - Get Pointer to Current Drive Parameter Table

(Undocumented)

```
AH = 1F
DL = drive number (0=default, 1=A, ...)


on return:
AL = 00  DS:BX is pointer to drive parameter table  (DPT)
     FF  drive does not exist
DS:BX = pointer to drive parameter table (DPT) if AL=0


- the format of the DPT varies between DOS versions
- calls INT 21,32 with DL=00 for DOS version 2.x+
```

System File Table (Undocumented)

DOS 2.x System File Table Format

Offset	Size	Description
00	dword	pointer to next system file table; offset of ffffh indicates last table in chain
04	word	number of file descriptors in table
06	nbytes	file descriptor table of the format (40 bytes each):

Offset	Size	Description
00	byte	count of handles referring to this file or zero if file is no longer open
01	byte	open mode
02	byte	file attribute
03	byte	drive (1=A, 2=B:, ..., 0=char device)
04	11bytes	filename in FCB format
0F	word	unknown
11	word	unknown
13	dword	file size
17	word	file date
19	word	file time
1B	byte	device attribute
1C	word	starting cluster of file (block device)
	dword	pointer to device driver (if char device)
1E	word	if block device, relative cluster within file of the last cluster read; zero if file has never been read or written
20	word	absolute cluster number of current cluster
22	word	unknown
24	dword	current file position

DOS 3.x System File Table and FCB Table Format

Offset	Size	Description
00	dword	pointer to next system file table; offset of ffffh indicates last table in chain
04	word	number of file descriptors in table
06	nbytes	file descriptor table of the format (53 bytes each):

Offset	Size	Description
00	word	count of handles referring to this file or zero if file is no longer open
02	word	open mode, bit 15 set if file id opened via FCB
04	byte	file attribute
05	word	device info word
07	dword	if char device pointer to device driver header if block device pointer to DOS Device Control Block
0B	word	starting cluster of file
0D	word	file time
0F	word	file date
11	dword	file size
15	dword	current file position
19	word	if block device, relative cluster within file of the last cluster read
1B	word	absolute cluster number of last cluster read zero if file has never been read or written
1D	word	sector number containing the directory entry
1F	byte	number of dir entry within sector
20	11bytes	filename in FCB format
2B	dword	SHARE pointer to previous SFT sharing same file
2F	word	SHARE number of network machine opening file
31	word	PSP segment of file owner
33	word	offset within SHARE code segment of sharing record

0000h = none

DOS 4.0+ System File Table and FCB Table

Offset	Size	Description
00	dword	pointer to next system file table; offset of ffffh indicates last table in chain
04	word	number of file descriptors in table
06	nbytes	file descriptor table of the format (59 bytes each):

Offset	Size	Description
00	word	count of handles referring to this file or zero if file is no longer open
02	word	open mode, bit 15 set if file id opened via FCB
04	byte	file attribute
05	word	device info word
07	dword	if char device pointer to device driver header if block device pointer to DOS Device Control Block or REDIR data
0B	word	starting cluster of file
0D	word	file time
0F	word	file date
11	dword	file size
15	dword	current file position
19	dword	if network redirector, pointer to REDIRIFS record
	word	if local block dev, relative cluster within file of the last cluster read
1B	dword	if local, sector number of directory entry
1F	byte	if local, number of directory entry within sector
20	11bytes	filename in FCB format
2B	dword	SHARE pointer to previous SFT sharing same file

```
        2F   word       SHARE number of network machine opening file
        31   word       PSP segment of file owner
        33   word       offset within SHARE code seg of sharing
record
                        0000h = none
        35   word       absolute cluster number of last cluster read
                        zero if file has never been read or written
        37   dword      pointer to file IFS driver, NULL if native
DOS
```

- these structures are allocated by DOS during the scanning of the CONFIG.SYS file
- INT 21,67 can be used to allow an application to have more than 20 file handles; the total number of handles is limited still to the value of FILES=n in CONFIG.SYS
- the structure of these fields is undocumented; I do not recommend relying on any of this information, instead use it as an aid in debugging
- see Bibliography references to Bernd Schemmer and "Data Structures Used in IBM PC Compatibles and the PS/2".
- see INT 21,52

INT 21,67 - Set Handle Count (DOS 3.3+)

```
        AH = 67h
        BX = new maximum open handles allowed

        on return
        CF = 0 if successful
           = 1 if error
        AX = error code if CF is set  (see DOS ERROR CODES)


        - gives program control of the number of files
simultaneously open
        - if CX is less than the current number of open files the
change
          will take effect when the number of open handles falls
below
          the new limit
        - this function allows the application to use more than 20
files,
          up to the FILES=N limit
        - earlier copies of IBM DOS 3.3 sometimes incorrectly
allocates
          memory (up to 64K) if an even number of handles is
requested
        - only the first 20 files handles are copied to a child
process
          regardless of the max number of files
        - it is possible to allocate more than 255 file handles but
it is
          difficult to use more than 255

        - see  SFT   FILE HANDLES
```

DOS Default/Predefined File Handles

```
 0 - Standard Input Device - can be redirected (STDIN)
 1 - Standard Output Device - can be redirected (STDOUT)
 2 - Standard Error Device - can be redirected (STDERR)
 3 - Standard Auxiliary Device (STDAUX)
 4 - Standard Printer Device (STDPRN)
```

See the following INT 21h Handle related function calls:

INT 21,3C Create file using handle
INT 21,3D Open file using handle
INT 21,3E Close file using handle
INT 21,3F Read file or device using handle
INT 21,40 Write file or device using handle
INT 21,41 Delete file
INT 21,42 Move file pointer using handle
INT 21,43 Change file mode
INT 21,45 Duplicate file handle
INT 21,46 Force duplicate file handle
INT 21,56 Rename file
INT 21,57 Get/set file date and time using handle
INT 21,5A Create temporary file (3.x+)
INT 21,5B Create new file (3.x+)
INT 21,67 Set handle count (3.3+)
INT 21,68 Flush buffer (3.3+)

- STDIN, STDOUT and STDERR can be redirected
- maximum number of files available to all DOS processes at one time is defined by the FILES=N statement of CONFIG.SYS
- maximum number of files available to an application is N-3 where N is derived from the CONFIG.SYS FILES=N statement minus the count of handles used by other processes
- INT 21,67 can be used in DOS 3.3+ to increase the number of file handles for an application to greater than 20; the max

handle
count is still limited by the value of FILES= in CONFIG.SYS file
- An application can increase the maximum allowed file handles in
DOS 3.0-3.2 by copying the open file table located through offset
34h in the PSP to another location. All unopened slots must be
initialized to FF and the values at PSP offsets 32h and 34h must
be updated to reflect the new values.
- see SFT INT 21 INT 21,52 INT 21,67

INT 21,3C - Create File Using Handle

```
AH = 3C
CX = file attribute  (see FILE ATTRIBUTES)
DS:DX = pointer to ASCIIZ path name


on return:
CF = 0 if successful
   = 1 if error
AX = files handle if successful
   = error code if failure  (see DOS ERROR CODES)


- if file already exists, it is truncated to zero bytes on
opening
```

INT 21,3D - Open File Using Handle

```
AH = 3D
AL = open access mode
     00  read only
     01  write only
     02  read/write
DS:DX = pointer to an ASCIIZ file name

on return:
AX = file handle if CF not set
   = error code if CF set  (see DOS ERROR CODES)
```

Access modes in AL:

```
|7|6|5|4|3|2|1|0|  AL
 | | | | | `-------- read/write/update access mode
 | | | | `--------- reserved, always 0
 | | | `---------- sharing mode (see below) (DOS 3.1+)
 | `--------------- 1 = private, 0 = inheritable (DOS 3.1+)
```

Sharing mode bits (DOS 3.1+):
654

```
000  compatibility mode (exclusive)
001  deny others read/write access
010  deny others write access
011  deny others read access
100  full access permitted to all
```

Access mode bits:
210

```
000  read access
001  write access
010  read/write access
```

- will open normal, hidden and system files
- file pointer is placed at beginning of file

INT 21,3E - Close File Using Handle

```
AH = 3E
BX = file handle to close


on return:
AX = error code if CF set  (see DOS ERROR CODES)


- if file is opened for update, file time and date stamp
  as well as file size are updated in the directory
- handle is freed
```

INT 21,3F - Read From File or Device Using Handle

```
        AH = 3F
        BX = file handle
        CX = number of bytes to read
        DS:DX = pointer to read buffer


        on return:
        AX = number of bytes read is CF not set
           = error code if CF set  (see DOS ERROR CODES)


        - read specified number of bytes from file into buffer DS:DX
        - when AX is not equal to CX then a partial read occurred
due
          to end of file
        - if AX is zero, no data was read, and EOF occurred before
read
```

INT 21,40 - Write To File or Device Using Handle

```
        AH = 40h
        BX = file handle
        CX = number of bytes to write, a zero value truncates/
extends
             the file to the current file position
        DS:DX = pointer to write buffer


        on return:
        AX = number of bytes written if CF not set
           = error code if CF set  (see DOS ERROR CODES)


        - if AX is not equal to CX on return, a partial write
occurred
        - this function can be used to truncate a file to the
current
          file position by writing zero bytes
```

INT 21,41 - Delete File

```
        AH = 41h
        DS:DX = pointer to an ASCIIZ filename


        on return:
        AX = error code if CF set  (see DOS ERROR CODES)


        - marks first byte of file directory entry with E5 to
indicate
          the file has been deleted.  The rest of the directory
entry
          stays intact until reused.   FAT pointers are returned to
DOS
        - documented as not accepting wildcards in filename but
actually
          does in several DOS versions
```

INT 21,42 - Move File Pointer Using Handle

```
AH = 42h
AL = origin of move:
     00 = beginning of file plus offset  (SEEK_SET)
     01 = current location plus offset  (SEEK_CUR)
     02 = end of file plus offset  (SEEK_END)
BX = file handle
CX = high order word of number of bytes to move
DX = low order word of number of bytes to move


on return:
AX = error code if CF set  (see DOS ERROR CODES)
DX:AX = new pointer location if CF not set


- seeks to specified location in file
```

INT 21,43 - Get/Set File Attributes

```
AH = 43h
AL = 00 to get attribute
   = 01 to set attribute
DS:DX = pointer to an ASCIIZ path name
CX = attribute to set

|5|4|3|2|1|0|  CX  valid file attributes
 | | | | | `---- 1 = read only
 | | | | `----- 1 = hidden
 | | | `------ 1 = system
 | `--------- not used for this call
 `---------- 1 = archive


on return:
AX = error code if CF set  (see DOS ERROR CODES)
CX = the attribute if AL was 00

- see   DIRECTORY
```

DIRECTORY - DOS Directory Structure

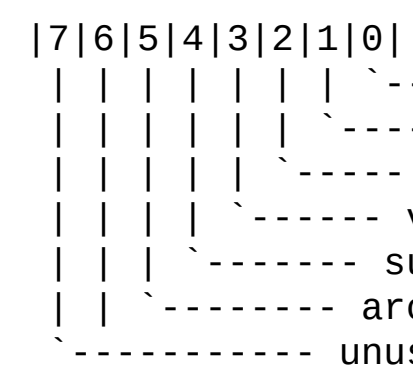
Byte **Description**

00 Filename status:
00 = Filename never used
05 = First character of filename is E5
E5 = File has been erased
2E = This is a subdirectory entry

00-07 Filename, left justified

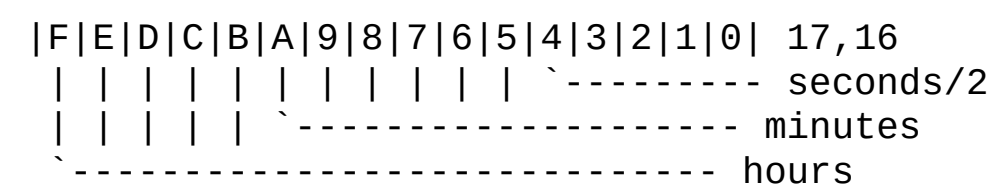
08-0A Filename extension, left justified

0B File's attribute:

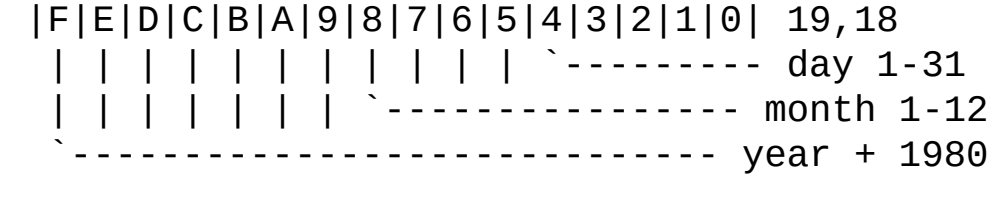


0C-15 Reserved by DOS

16-17 Time the file was created or last updated:



18-19 Date the file was created or last updated:



1A-1B Starting cluster number of the first file cluster

1C-1F File size in bytes (low order first)

INT 21,45 - Duplicate File Handle

```
AH = 45h
BX = file handle


on return:
AX = new file handle if CF not set
   = error code if CF set  (see DOS ERROR CODES)


- gets another file handle for the same file
- both file handles move in unison
- often used to flush file data and update a file directory
  entry without closing the initial file
```

INT 21,46 - Force Duplicate File Handle

```
AH = 46h
BX = existing file handle
CX = second file handle


on return:
AX = error code if CF set  (see DOS ERROR CODES)


- if file handle in CX is currently open, current file
  identified by CX is closed and the handle in BX is
  dupped and placed in CX
- after duping file handles move together through file
- similar to INT 21,45
```

INT 21,56 - Rename File

```
AH = 56h
DS:DX = pointer to old ASCIIZ path/filename
ES:DI = pointer to new ASCIIZ path/filename


on return:
AX = error code if CF set  (see DOS ERROR CODES)
```

- supports full pathnames and allows renaming files across directories and in DOS 3.x allows renaming subdirectories
- does not support use of wildcards unless invoked from via INT 21,5D in which case error code 12h is returned
- unpredictable result may occur if an opened file is renamed
- see Bibliography reference to "Undocumented DOS"

INT 21,5D - Critical Error Information (Undocumented, DOS 3.x+)

See second reference below for more information about this interrupt
information presented here is incomplete

AH = 5D
AL = 00 server function call (DOS 3.1+)
= 01 commit all files (DOS 3.1+)
= 02 SHARE: close file by name (DOS 3.1+)
= 03 SHARE: close all files for given computer (DOS 3.1+)
= 04 SHARE: close all files for given process (DOS 3.1+)
= 05 SHARE: get open file list entry (DOS 3.1+)
= 06 get address of DOS swappable area into DS:SI (DOS 3.0+)
= 07 get network redirected printer mode (DOS 3.1+)
= 08 set network redirected printer mode (DOS 3.1+)
= 09 flush network redirected printer output (DOS 3.1+)
= 0A set extended error information (DOS 3.1+)
= 0B get DOS swappable data areas (DOS 4.x+)
DS:DX = pointer to 18 byte DOS Parameter List (DPL, if AL=00)
= pointer to 9 byte data block of the form (AL=0A):

Offset	Size	Description
00	word	extended error code to set
02	dword	pointer to driver address to set
06	byte	action code to set
07	byte	class code to set
08	byte	locus code to set

on return:
DS:SI = (if AL was 6) address of critical flag of the form:

Offset	Size	Description
00	word	extended error code
02	byte	action code

```
              03   byte   class code
              04   byte   pointer to driver address
```

- function 0 copies 18 bytes from DS:SI to the DOS internal register-save area; this data will be placed in the registers when DOS returns to the caller thereby circumventing the DOS register save logic
- may be used by a TSR to prevent accidental changing of an error code and causing problems in the foreground process
- see bibliography references for "Undocumented DOS" and "DOS Programmers Reference"

- see INT 21,59

Bibliography of Published Sources Referenced in HelpPC

The following list contains information on reference materials used to verify and supply the information found in HelpPC. They are listed chronologically in the order I used them in HelpPC.

1. Powell, David. "IBM PC-DOS Programmer's Quick Reference Summary". (Unpublished paper.)

2. Powell, David. "IBM BIOS Programmer's Quick Reference Summary". (Unpublished paper.)

3. Norton, Peter. "Programmer's Guide to the IBM PC". Redmond, Washington: Microsoft Press, 1985.

4. Duncan, Ray. "Advanced MS-DOS". Redmond, Washington: Microsoft Press, 1986.

5. IBM Corporation. "Disk Operating System Version 3.10: Technical Reference". Boca Raton, Florida: International Business Machines Corporation, 1986.

6. Davies, Russ. "COMPUTE!'s Mapping the IBM PC and PCjr". Greensboro, North Carolina: COMPUTE! Publications, Inc., 1986.

7. Brenner, Robert C. "IBM PC Troubleshooting & Repair Guide". Indianapolis, Indiana: Howard W Sams & Company, 1985.

8. Borland International, Inc. "Turbo C Reference Guide". Scotts Valley, California: Borland International, Inc., 1987.

9. Scanlon, Leo J. "8086/88 Assembly Language Programming".

Bowie, Maryland: Robert J. Brady Co., 1984.

10. IBM Corporation. "Technical Reference: PC/XT". Boca Raton, Florida: International Business Machines Corporation, 1983.

11. Wilton, Richard. "Programmer's Guide to PC & PS/2 Video Systems". Redmond, Washington: Microsoft Press, 1987.

12. Dettemann, Terry R. "DOS Programmers Reference". Carmel, Indiana: Que Corporation, 1988.

13. Hogan, Thom. "The Programmer's PC Sourcebook". Redmond, Washington: Microsoft Press, 1988.

14. Intel Corporation. "Microprocessor and Peripheral Handbook". 2 vols. Mountain View, California: Intel Corporation, 1989.

15. Wyatt, Allen L. Sr. "Assembly Language Quick Reference". Carmel, Indiana: Que Corporation, 1989.

16. IBM Corporation. "Technical Reference: PS/2 Model 30 Technical Reference". Boca Raton, Florida: International Business Machines Corporation, 1987.

17. IBM Corporation. "Technical Reference: PS/2 Model 50 and 60 Technical Reference". Boca Raton, Florida: International Business Machines Corporation, 1987.

18. Norton, Peter, and Richard Wilton. "Programmer's Guide to the IBM PC & PS/2". Redmond, Washington: Microsoft Press, 1988.

19. Duncan, Ray, and Susan Lammers, eds. "The MS-DOS Encyclopedia". Redmond, Washington: Microsoft Press, 1988.

20. IBM Corporation. "Technical Reference: PCjr". Boca

Raton,
Florida: International Business Machines Corporation, 1983.

21. IBM Corporation, "Technical Reference: PC/AT". Boca Raton,
Florida: International Business Machines Corporation, 1984.

22. Bailey, Sharon. "Periscope Manual". Atlanta, Georgia: The Periscope Company, 1990.

23. Microsoft Corporation. "Microsoft Mouse Programmer's Reference". Redmond, Washington: Microsoft Press, 1989.

24. Schemmer, Bernd. Letter to author. 1 July 1990.

25. Microsoft Corporation. "Microsoft Macro Assembler 5.0, Programmer's Guide". Redmond, Washington: Microsoft Corporation, 1987.

26. Parke, William C. "Data Structures Used in IBM PC Compatibles
and the PS/2". (Unpublished paper).

27. NEC Electronics, Inc. "Intelligent Peripheral Devices (IPD)
Data Book". Mountain View, California: NEC Electronics, Inc., 1989.

28. Seiko Epson Corporation. "Epson LX-800 User's Manual". Nagano,
Japan: Seiko Epson Corporation, 1987.

29. Hewlett-Packard Company. "Hewlett-Packard LaserJet Family
Technical Reference Manual". Boise, Idaho: Hewlett-Packard
Company, 1986.

30. "System BIOS for IBM PC/XT/AT Computers and Compatibles".
Phoenix Technical Reference Series. Reading, Massachusetts:
Addison-Wesley, 1990.

31. Schulman, Andrew. "Undocumented DOS". Reading, Massachusetts: Addison-Wesley, 1990.

INT 21,57 - Get/Set File Date and Time Using Handle

```
AH = 57h
AL = 00  get date and time
   = 01  set date and time
   = 02  ??? (DOS 4.0+ undocumented)
   = 03  ??? (DOS 4.0+ undocumented)
   = 04  ??? (DOS 4.0+ undocumented)
BX = file handle
CX = time to set (if setting)
DX = date to set (if setting)
ES:DI = pointer to buffer to contain results


on return:
AX = error code if CF set  (see  DOS ERROR CODES)
CX = file time (if reading, see below)
DX = file date (if reading, see below)
```

Time encoding:

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  Time in CX
 | | | | | | | | | | | `------------  two second incr (0-29)
 | | | | | `-----------------------  minutes 0-59)
 `-------------------------------  hours (0-29)
```

Date Encoding

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  Date in DX
 | | | | | | | | | | | `------------  day (1-31)
 | | | | | | | `-------------------  month (1-12)
 `-------------------------------  year - 1980
```

INT 21,5A - Create Temporary File (DOS 3.0+)

```
AH = 5A
CX = attribute
DS:DX = pointer to ASCIIZ path ending in '\'


on return:
AX = handle if call CF clear
   = error code if CF set  (see DOS ERROR CODES)
DS:DX = pointer to updated ASCIIZ filespec


- creates files with random names with any combination of
  the following attributes: normal, system and hidden
- ending backslash is required
```

INT 21,5B - Create File (DOS 3.0+)

```
AH = 5B
CX = attribute
DS:DX = pointer to ASCIIZ path/filename


on return:
AX = handle if CF not set
   = error code if CF set  (see DOS ERROR CODES)


- standard method of opening files
- returns a file handle of a file opened with specified
  attributes (combinations of normal, system and hidden)
```

INT 21,68 - Flush Buffer Using Handle (DOS 3.3+)

```
        AH = 68h
        BX = file handle


        on return
        AX = error code if CF set  (see DOS ERROR CODES)


        - flushes DOS buffers to disk, does not update directory
entry
```

INT 21,34 - Get Address to DOS Critical Flag

INDOS (Undocumented DOS 2.0+)

```
        AH = 34h

        on return:
        ES:BX = address of a byte indicating whether a DOS call is
                in progress.  No DOS calls should be made if set.


        - AKA the INDOS flag, this critical section flag may be
checked from
          within an interrupt handler before requesting a DOS
service. It is
          a semaphore that is non-zero when DOS is busy, and zero
otherwise.
        - though this flag indicates whether a DOS interrupt is
active, it
          should not be used alone to determine DOS is safe for re-
entry;
          Here's the standard rule for safe DOS entry: if INT 28 is
active
          or this flag and the critical error flag are clear then
it is safe
          to call DOS
        - this interrupt should be used only during TSR
initialization;
          the returned pointer should be used thereafter
        - this flag is cleared after a critical error (INT 24)
        - ES:BX-1 points to the critical error flag for DOS 3.x+
          ES:BX+1 points to the critical error flag for DOS 2.x
          ES:BX-1AA points to the critical error flag for COMPAQ
DOS 3.0
          INT 21,5D in DOS 3.x+ can be used to locate the critical
          error flag
```

INT 28 - DOS Idle Loop / Scheduler (Undocumented)

- issued by DOS during keyboard poll loop
- indicates DOS may be carefully re-entered by TSR
- used by TSR programs to popup and make DOS disk I/O calls
- supported by OS/2 compatibility box
- default behavior is to simply perform an IRET
- any DOS functions above 0Ch may be called while in this handler
- INT 21,3F and INT 21,40 may not use a handle that refers to
  CON while in this handler
- see also INDOS

INT 21,C - Clear Keyboard Buffer and Invoke Keyboard Function

```
AH = 0C
AL = 01, 06, 07, 08 or 0A   (INT 21 input functions)


on return:
see return values from INT 21,AL where AL is 1, 6, 7, 8 or A


- main function is to clear the input buffer and call INT
21h with
  the specified function (in AL)
- see INT 21,1,  INT 21,6,  INT 21,7,  INT 21,8 & INT 21,A
```

INT 21,6C - Extended Open/Create (DOS 4.x+)

```
        AH = 6C
        AL = 00
        BX = open mode (see below)
        CX = file attribute (see FILE ATTRIBUTES, bits 15-8 are
zero)
        DX = function control indicator (see below)
        DS:SI = ASCIIZ filespec


        returns:
        AX = handle if CF clear
           = error code if CF set  (see DOS ERROR CODES)
        CX = 01  file opened
           = 02  file created and opened
           = 03  file truncated and opened (replaced)
        CF = 0 if successful
           = 1 if error
```

Open Mode in BX:

```
        |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  BX  (Open Mode)
         | | | | | | | | | | | | `--------- read/write access mode
         | | | | | | | | | | | `--------- reserve (must be 0)
         | | | | | | | | | `------------ sharing mode
         | | | | | | | | `------------ 1 = private, 0 = inheritable
         | | | `--------------------- reserved
         | | `---------------------- 0 = call INT 24, 1 = ignore INT
24
         | `---------------------- 0 = normal write, 1 = flush each
write
         `---------------------- reserved
```

Read/Write Access bits:

210

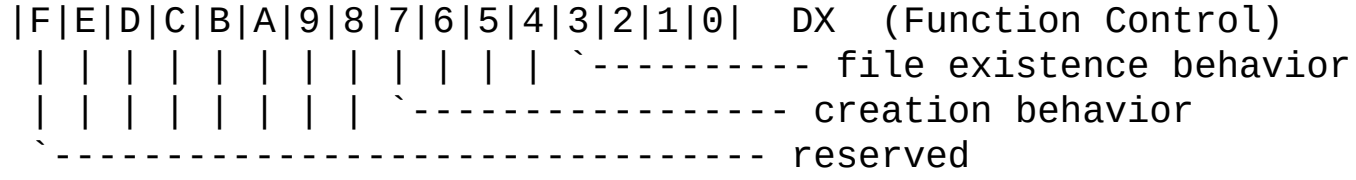
```
000  read access
001  write access
010  read/write access
```

Sharing Mode bits:

654

```
000  compatibility mode
001  read/write access
010  write access denied
011  read access denied
100  full access
```

Function Control Indicator in DX:



File Existence Behavior bits:
3210
0000 if file exist, generate error, take no action
0001 if file exists, open file
0010 if file exists, truncate and open

Creation Behavior bits:
7654
0000 if file does not exist, generate error, take no action
0001 if file does not exist, create

INT 21,7 - Direct Console Input Without Echo

```
AH = 07


on return:
AL = character from STDIN


- waits for keyboard input until keystroke is ready
- character is not echoed to STDOUT
- returns 0 for extended keystroke, then function must be
  called again to return scan code
- ignores Ctrl-Break and Ctrl-PrtSc
- see   INT 21,1
```

INT 21,8 - Console Input Without Echo

```
AH = 08


on return:
AL = character from STDIN


- returns 0 for extended keystroke, then function must be
  called again to return scan code
- waits for character from STDIN and returns data in AL
- if Ctrl-Break is detected, INT 23 is executed
```

INT 21,A - Buffered Keyboard Input

```
        AH = 0A
        DS:DX = pointer to input buffer of the format:

        | max | count |  BUFFER (N bytes)
           |      |          `------ input buffer
           |      `------------ number of characters returned (byte)
           `-------------- maximum number of characters to read
(byte)


        returns nothing

        - since strings can be pre-loaded, it is recommended that
the
          default string be terminated with a CR
        - N bytes of data are read from STDIN into buffer+2
        - max buffer size is 255, minimum buffer size is 1 byte
        - chars up to and including a CR are placed into the buffer
          beginning at byte 2;  Byte 1 returns the number of chars
          placed into the buffer  (extended codes take 2 characters)
        - DOS editing keys are active during this call
        - INT 23 is called if Ctrl-Break or Ctrl-C detected
```

INT 21,49 - Free Allocated Memory

```
        AH = 49h
        ES = segment of the block to be returned (MCB + 1para)


        on return:
        AX = error code if CF set  (see DOS ERROR CODES)


        - releases memory and MCB allocated by INT 21,48
        - may cause unpredictable results is memory wasn't
allocated using
          INT 21,48 or if memory wasn't allocated by the current
process
        - checks for valid MCB id, but does NOT check for process
ownership
        - care must be taken when freeing the memory of another
process, to
          assure the segment isn't in use by a TSR or ISR
        - this function is unreliable in a TSR once resident, since
          COMMAND.COM and many other .COM files take all available
memory
          when they load
        - see  INT 21,4A
```

INT 21,4A - Modify Allocated Memory Block (SETBLOCK)

```
AH = 4A
BX = new requested block size in paragraphs
ES = segment of the block (MCB + 1 para)


on return:
AX = error code if CF set  (see DOS ERROR CODES)
BX = maximum block size possible, if CF set and AX = 8
```

- modifies memory blocks allocated by INT 21,48
- can be used by programs to shrink or increase the size of allocated memory
- PC-DOS version 2.1 and DOS 3.x will actually allocate the largest available block if CF is set. BX will equal the size allocated.
- see also INT 21,49

INT 21,4F - Find Next Matching File

```
AH = 4F
DS:DX = unchanged from previous function 4E


on return:
AX = error code if CF set  (see DOS ERROR CODES)


- INT 21,4E should be called to find the first file and this
  function should be called to find all other matching files
- normal files are always included along with files that
match
  the requested attributes except when the LABEL attribute
is
  requested.  It's up to the programmer to determine which
  actually match the requested attributes.
- see  INT 21,1A  DTA  FILE ATTRIBUTES
```

TC: int findnext(struct ffblk *ffblk) MSC: unsigned _dos_findnext(struct find_t *fileinfo) - prototype in dir.h, also include (MSC: dos.h only) - findnext updates DTA - returns 0 if OK, -1 if no matching files found or on error - gets next disk directory via DOS 0x4F, after calling findfirst() - struct ffblk is defined in dir.h for Turbo C and dos.h for MSC **struct ffblk (Turbo C) struct find_t (MSC)** { { char ff_reserved[21]; char reserved[21]; char ff_attrib; char attrib; unsigned ff_ftime; unsigned wr_time; unsigned ff_fdate; unsigned wr_date; long ff_fsize; long size; char ff_name[13]; char name[13]; }; }; - normal files are always included along with files that match the requested attributes except when the LABEL attribute is requested. It's up to the programmer to determine which actually match the requested attributes. - see FILE ATTRIBUTES, findfirst(), getdta(), setdta()

TC: char far *getdta(void)

- prototype in dos.h

- returns current setting of the DTA as a far pointer
- see INT 21,2F

TC: void setdta(char far *dta)

- prototype in dos.h

- changes current setting of the DTA as given by DTA
- no corresponding function in MS C

INT 22 - Program Terminate

```
no input data
```

- not an interrupt, but a vector to the terminate address
- copied to PSP offset 0Ah during program load
- do not execute this interrupt directly

INT 21,4B - EXEC/Load and Execute Program

```
        AH = 4B
        AL = 00  to load and execute program
           = 01  (Undocumented)  create program segment prefix and
load
                 program, but don't execute.  The CS:IP and SS:SP
of the
                 program is placed in parameter block.  Used by
debuggers
           = 03  load program only
           = 04  called by MSC spawn() when P_NOWAIT is specified
        DS:DX = pointer to an ASCIIZ filename
        ES:BX = pointer to a parameter block


        on return:
        AX = error code if CF set  (see DOS ERROR CODES)
        ES:BX = when AL=1, pointer to parameter block similar to:
```

Offset	Size	Description
00	word	when AL=1, segment of env. or zero if using parents env.
	word	when AL=3, segment of load point for overlay
02	dword	when AL=1, pointer to cmd line at PSP 80h
	word	when AL=3, relocation factor for EXE overlay
06	dword	pointer to default FCB passed at PSP 5Ch
0A	dword	pointer to default FCB passes at PSP 6Ch
0E	dword	value of program SS:SP
12	dword	value of program CS:IP

- allows execution of an external program as well as overlay management from within an application
- all registers except CS and IP are destroyed
- SS and SP should be preserved in code segment before call since a bug in DOS version 2.x destroys these
- return code can be retrieved if child process exits via INT 21,4C
- calling process must assure presence of enough unallocated memory
- subfunction 4 returns with an error under DOS 4.x+

- calls INT 21,55

- see also INT 21,26

INT 21,55 - Create New PSP (Undocumented)

```
        AH = 55h
        DX = New PSP segment address
        SI = memory size value to place in PSP offset 02h (DOS 3+)

        returns nothing

        - the following should be observed when using this function

          o  Allocate memory for the PSP and program code  (INT
21,48)
          o  Duplicate the PSP   (INT 21,55)
          o  Load program code into allocated segment above the new
PSP
          o  Save Current PSP
          o  Set PSP to the PSP just created  (INT 21,50)
          o  Jump to start of code if .COM otherwise handle
relocation,
             setup stack and registers

        - similar to INT 21,26 except the PSP is setup by DOS
        - increments reference count in SFT for all inherited files
        - files flagged as not inheritable are marked as closed in
the
          new PSP
        - invoked by INT 21,4B

        - see   UNDOC   INT 21,4B   INT 21,26
```

INT 21,50 - Set Current Process ID (Undocumented DOS 2.x)

```
AH = 50h
BX = process ID number (process PSP segment address)

returns nothing

- the process ID number is actually the segment address of a
  program's PSP.  This is useful for TSR's to access their own
  file handle table inside their PSP.
- this function cannot be called while in an INT 28 handler in
  DOS 2.x unless the critical error flag is set or stack corruption
  will occur
- safe in DOS 3.x INT 28 handlers since they use a different stack
  by default
- available in OS/2 compatibility box
- see  INT 21,51   INT 21,62
```

INT 21,51 - Get Current Process ID (Undocumented DOS 2.x)

```
        AH = 51h

        on return:
        BX = process ID
```

- The process ID number is actually the segment address of program's PSP. This in conjunction with INT 21,50 is useful for TSR's to access their own file handle table in their respective PSP.
- this function cannot be called while in an INT 28 handler in DOS 2.x unless the critical error flag is set or stack corruption will occur
- INT 21,62 is highly recommended for DOS 3.x due to a possible bug when activated from a TSR. DOS may switch to the wrong internal stack which may cause a problems with TSR's if called during an INT 28.
- see INT 21,62 (Get PSP segment) for DOS 3.x applications

INT 21,62 - Get PSP address (DOS 3.x)

```
AH = 62h


on return:
BX = segment address of current process



- this is useful for TSR's to access their own file handles
- for DOS 2.x use INT 21,51

- see also  INT 21,50
```

INT 21,26 - Create New Program Segment Prefix

```
AH = 26h
DX = segment address of new PSP

returns nothing

- allocates memory for a PSP and copies current PSP there
- used before DOS 2.x to spawn a child process
- the application is responsible for allocating any memory
  necessary for the child process
- INT 21,4B (EXEC) is now the recommended method for
starting
  a child process, so this function should be avoided

- see also INT 21,55
```

DOS Undocumented Interrupts, Functions and Structures

INT 15,20	PRINT.COM Critical Region Flag (DOS 3.x+)
INT 21,1F	Get pointer to current drive parameter table
INT 21,32	Get pointer to drive parameter table
INT 21,34	Get address to DOS critical flag (INDOS flag)
INT 21,37	Get/set switch char
INT 21,4B	EXEC/Load and execute (Function 1 undocumented)
INT 21,50	Set current process id (DOS 2.x)
INT 21,51	Get current process id (DOS 2.x)
INT 21,52	Get pointer to DOS "INVARS"
INT 21,53	Generate drive parameter table
INT 21,55	Create new PSP
INT 21,58	Get/set memory allocation strategy (DOS 3.x+)
INT 21,5D	Critical error information (DOS 3.x+)
INT 21,60	Get fully qualified file name (DOS 3.x+)
INT 21,64	Set device driver look ahead (DOS 3.3+)
INT 21,69	Get/set disk serial number (DOS 4.0+)
INT 21,F8	Set INT 21 OEM handler
INT 28	DOS idle loop/scheduler
INT 29	Fast character output
INT 2E	Execute command using base level COMMAND.COM
BCB	Batch Control Block
DTA	Disk Transfer Area (partially undocumented)
MCB	Memory Control Block
PSP	Program Segment Prefix Layout (partially undocumented)
SFT	System File Table

- the above items are undocumented with respect to IBM/Microsoft
  and should be not be used unless one fully understands the repercussions
- several functions are version dependant and OEM specific

INT 15,20 - PRINT.COM Critical Region Flag (undocumented DOS 3.x+)

```
AH = 20h
AL = 00  disable user critical region flag
   = 01  enable user critical region flag
ES:BX = pointer to user critical region flag byte (AH = 1)


returns nothing


- provides PRINT.COM with the address of a byte to
increment on
  entry to DOS functions
- see bibliography reference for "Undocumented DOS"
```

INT 21,37 - Get/Set Switch Character (Undocumented, DOS 2.0+)

```
        AH = 37h
        AL = 0  get switch character into DL;  some systems return
"-"
           = 1  set switch character to value in DL
           = 2  read device prefix flag into DL;  returns DL = 0
indicating
                devices must be accessed using /DEV/device. A non-
zero value
                indicates devices may be accessed without prefix
           = 3  set device prefix flag, device names must begin
with \DEV\.
        DL = new switch character (AL=1)
           = 00  \DEV\ must preceed device names (AL=3)
           = 01  \DEV\ is not neccesary in device names (AL=3)

        on return:
        AL = FF  illegal subfunction code specified
        DL = current switch character (AL=0)
           = device availability (AL=2, always FF with DOS 4.x+)


        - subfunctions 0 and 1 were formerly available as a CONFIG.
SYS
          command in DOS versions before 3.x;   also supportedin
the OS/2
          compatibility box
        - subfunctions 2 and 3 were available in DOS 2.x only;
they have
          no effect in DOS 4.x+
        - /DEV/ prefix is valid in DOS 2.x by default, setting the
flag makes
          it mandatory.  DOS internal commands like DIR, TYPE and
DEL do not
          recognize filenames the are identical to device names
regardless of
          the setting of the AVAILDEV flag.
```

INT 21,53 - Generate Drive Parameter Table (Undocumented)

```
        AH = 53h
        DS:SI = address of BIOS Parameter Block (BPB)
        ES:BP = pointer to buffer to hold first Drive Parameter
Table (DPT)


        returns:
        ES:BP = pointer to buffer containing the first Drive
Parameter Table
                in chain


        - available since DOS 2.0+
        - translates BPB  into a DOS Disk Parameter Table
        - see  INT 21,32  INT 21,52
```

INT 21,58 - Get/Set Memory Allocation Strategy

(Undocumented, DOS 3.x)

```
AH = 58h
AL = 00  get strategy code
   = 01  set strategy code
BX = strategy code (when AL = 01)
   = 00  first fit  (default)
   = 01  best fit
   = 02  last fit

on return:
AX = strategy code if CF clear
   = error if CF set, see  DOS ERROR CODES


- defaults to first fit, first block large enough is chosen
- in first fit, DOS searches the MCB chain from low addresses to
  high for the first block large enough to fill the request.  For
  best fit, DOS searches all memory blocks for the closest fit.
  In last fit, DOS starts at high addresses and works downward
  and uses the high part of the chosen block
- any strategy value greater than 2 defaults to 2, but the last value
  set, even if incorrect, is the value returned by the get strategy
```

INT 21,60 - Get Fully Qualified File Name (Undocumented 3.x+)

```
        AH = 60h
        DS:SI = pointer to ASCIIZ string containing unqualified
filename
        ES:DI = pointer to 128 byte buffer to contain fully
qualified filename


        on return:
        ES:DI = address of fully qualified filename string
        AH = error code if CF set


        - undocumented, available in DOS 3.x+
        - does not check for file existence, simple provides a fully
          qualified filename for the given filespec
        - this function ignores ASSIGN'ed, JOIN'ed, SUBST'ed and
network
          redirections and returns the true path name
        - returned string is in upper case and has forward slashes
replaced
          by back slashes;  filename always begins with "d:" or "\
\" format
        - AL register is destroyed
        - supported by OS/2 1.1 compatibility box
```

INT 21,64 - Set Device Driver Look Ahead (Undocumented)

```
        AH = 64h
        AL = value for switch
             00   causes DOS to perform look ahead to the device
driver
                  before execution of INT 21,1, INT 21,8 and INT 21,
A
             other - no look ahead is performed


        returns ???


        - DOS internal; should not be called by user program
        - switch defaults to zero
        - undocumented; since DOS 3.3
        - some other subfunctions are know to have existed since
DOS 3.2 but
          details on their functionality are incomplete
```

INT 21,69 - Get/Set Disk Serial Number (Undocumented DOS 4+)

```
        AH = 69h
        AL = 00 get serial number
           = 01 set serial number
        BL = drive (0 = default, 1=A:, 2 = B:, ... )
        DS:DX = buffer to contain extended BIOS Parameter Block
(BPB, AL=0)
              = disk information to set of the form:

          Offset  Size          Description
            00    word    info level (zero)
            02    dword   disk serial number (binary)
            06  11bytes   volume label or "NO NAME    " if not
present
            11   8bytes   filesystem type string "FAT12   " or
"FAT16  "

        returns:
        CF = set on error
           AX = error code if CF set
              = 01  network drive not supported
              = 05  no extended BPB found on disk
        CF = clear if successful and value in AX is destroyed
           AL = 00  buffer filled from Extended BPB on disk
              = 01  disk Extended BPB updated from buffer
        DS:DX = pointer to returned data, when AL = 0


        - copies data from Extended BIOS Parameter Block to buffer
        - no critical error is generated on error
        - does not work on network drives (returns 01)
        - returned buffer (after offset 2) is similar to offsets 27
through
          3D of the Extended BPB found in the boot record

        - see  INT 21,59
```

INT 2E - Execute Command Using Base Level Command Interpreter

(Undocumented DOS 2.0+)

```
        DS:SI = pointer to command string to be executed

        returns nothing ???

        - causes the base level COMMAND.COM to execute a command as
if it
          were typed from the keyboard
        - the transient portion of COMMAND.COM must be loaded if
not resident
        - first byte of the string contains the string length,
inclusive
        - command string must be terminated with a CR
        - all registers including SS & SP are destroyed;  SS & SP
should be
          preserved in your own code segment
        - can be used to modify parents environment with SET command
        - incompatible under some software multitasking systems
        - before returning to the parent program this interrupt will
          attempt to execute the "current" batch file (if the
parent or
          any ancestors were invoked from a batch file.   This can
cause
          remaining lines in a batch file to execute before the
parent
          program finishes and fragment memory
```

BCB - Batch Control Block (undocumented)

DOS 2.x thru DOS 3.2 BCB Format

Offset	Size	Description
00	byte	unknown
01	word	if non-zero; segment of control block for active FOR
03	byte	type of batch command 0 - normal batch command 1 - FOR-loop active
04	dword	offset of next command to execute in batch file
07	word	offset of variable %0 (batch file name)
09	9 words	offset of %N batch file parameters, 0FFFFh indicates parameter is null
1C	nbytes	null terminated path and filename of the current batch file immediately followed by command line parameters. Each parameter %0-%9 plus a CR is appended and resulting string is null terminated.

DOS 3.3 BCB Format

Offset	Size	Description
00	byte	unknown
01	byte	global echo switch, if exec'd by batch CALL-command 1 - turn ECHO ON on return to calling batch file

```
                            0 - turn ECHO OFF on return to calling
batch file
        02   word       batch file BCD segment if executed via CALL
                            if zero; batch file was called from
command line
                            if non-zero; batch file executed via
CALL-command
        04   word       if non-zero; segment of control block for
active FOR
        06   byte       type of batch command
                            0 - normal batch command
                            1 - FOR-loop active
        07   dword      offset of next command to execute in batch
file
        0B   word       offset of variable %0 (batch file name)
        0D  9words      offset of %N batch file parameters, 0FFFFh
indicates
                        parameter is null
        1F  nbytes      null terminated path and filename of the
current
                        batch file immediately followed by command
line
                        parameters.  Each parameter %0-%9 plus a CR
is
                        appended and resulting string is null
terminated.


        - BCB length is variable and depends on the size and count
of the
          parameters and fully qualified batch file name
        - the MCB for a BCB has a process Id of the transient
portion of
          the latest COMMAND.COM
        - offsets displayed are relative to the BCB segment
        - SHIFT command changes the offsets of the parameters in
the table
          at the offsets 0B0h thru 1Ch
        - BCB  of DOS 3.3 is the  same as earlier versions except 3
bytes
          were added after offset 0
        - to find a BCB, locate the first block in the MCB chain
belonging
          to COMMAND.COM (the second allocated block always belongs
to
          COMMAND.COM).  Then scan the MCB chain for a 64 byte
```

```
block with
          the same owner ID as COMMAND.COM).  This will be the BCB.
```

6845 - Motorola CRT Controller

```
        3B0-3BB Monochrome Monitor Adapter
        3D0-3DC Color Graphics Adapter (mapped similarly)

        3B0/3D0  port address decodes to 3B4/3D4
        3B1/3D1  port address decodes to 3B5/3D5
        3B2/3D2  port address decodes to 3B4/3D4
        3B3/3D3  port address decodes to 3B5/3D5
        3B4/3D4  6845 index register, selects which register [0-11h]
                 is to be accessed through port 3B5/3D5
        3B5/3D5  6845 data register [0-11h] selected by port
3B4/3D4,
                 registers 0C-0F may be read.  If a read occurs
                 without the adapter installed, FFh is returned.
        3B6/3D6  port address decodes to 3B4/3D4
        3B7/3D7  port address decodes to 3B5/3D5
        3B8/3D8  6845 mode control register
        3B9/3D9  color select register on color adapter
        3BA/3DA  status register (read only)
        3BB/3DB  light pen strobe reset
            3DC  preset light pen latch
            3DF  CRT/CPU page register (PCjr only)
```

Registers: Accessed through ports 3B5 & 3D5	**VALID VALUES**		
	MONO	**CO40**	**CO80**
GRPH			
00 - Horiz. total characters	61	38	71
38			
01 - Horiz. displayed characters per line	50	28	50
28			
02 - Horiz. synch position	52	2D	5A
2D			
03 - Horiz. synch width in characters	0F	0A	0A
0A			
04 - Vert. total lines	19	1F	1F
7F			
05 - Vert. total adjust (scan lines)	06	06	06
06			
06 - Vert. displayed rows	19	19	19
64			
07 - Vert. synch position (character rows)	19	1C	1C
70			

```
        08 - Interlace mode                          02   02   02
02
        09 - Maximum scan line address               0D   07   07
01
        0A - Cursor start (scan line)                0B   06   06
06
        0B - Cursor end (scan line)                  0C   07   07
07
        0C - Start address (MSB)                     00   00   00
00
        0D - Start address (LSB)                     00   00   00
00
        0E - Cursor address (MSB) (read/write)       00   --   --
--
        0F - Cursor address (LSB) (read/write)       00   --   --
--
        10 - Light pen (MSB)   (read only)           --   --   --
--
        11 - Light pen (LSB)   (read only)           --   --   --
--

        - Registers 00-0D are write only, registers 0E-0F are read/
write and
          registers 10-11 are read only
        - Cursor address is calculated with using the following
(row*80)+col
```

6845 - Port 3B8 (Monochrome)

```
        |7|6|5|4|3|2|1|0|  3B8 CRT Control Port
         | | | | | | | `---- 1 = 80x25 text
         | | | | | | `------- unused
         | | | | `-------- 1 = enable video signal
         | | | `---------- unused
         | | `----------- 1 = blinking on
         `-------------- unused
```

6845 - Port 3D8 (Color)

```
        |7|6|5|4|3|2|1|0|  3D8 Mode Select Register
```

```
        | | | | | | | `---- 1 = 80x25 text, 0 = 40x25 text
        | | | | | | `----- 1 = 320x200 graphics, 0 = text
        | | | | | `------ 1 = B/W, 0 = color
        | | | | `------- 1 = enable video signal
        | | | `-------- 1 = 640x200 B/W graphics
        | | `--------- 1 = blink, 0 = no blink
        `------------ unused
```

6845 - Port 3D9 Color Text Modes

```
       |7|6|5|4|3|2|1|0|  3D9 Color Select Register (3B9 not used)
        | | | | | `-------- screen/border RGB
        | | | | `--------- select intensity setting
        | | | `---------- background intensity
        `--------------- unused
```

6845 - Port 3D9 Color Graphics Modes

```
       |7|6|5|4|3|2|1|0|  3D9 Color Select Register (3B9 not used)
        | | | | | `-------- RGB for background
        | | | | `--------- intensity
        | | | `---------- unused
        | | `----------- 1 = palette 1, 0=palette 0 (see below)
        `-------------- unused

         Palette 0 = green, red, brown
         Palette 1 = cyan, magenta, white
```

6845 - Port 3DA Status Register

```
       |7|6|5|4|3|2|1|0|  3DA Status Register
        | | | | | | | `---- 1 = display enable, RAM access is OK
        | | | | | | `----- 1 = light pen trigger set
        | | | | | `------ 0 = light pen on, 1 = light pen off
        | | | | `------- 1 = vertical retrace, RAM access OK for
next 1.25ms
        `-------------- unused
```

INT 25 - Absolute Disk Read

```
AL = logical drive number (0=A:, 1=B:, 2=C:, ...)
CX = number of sectors to read
   = -1 if DOS 4.x (control block pointer is in DS:BX)
DX = starting logical sector number (see below for DOS 4.x+)
DS:BX = pointer to data buffer
      = pointer to control block (DOS 4.x+, see below)

on return:
AH = error code if CF set:
     01  bad command
     02  bad address mark
     03  write protect
     04  sector not found
     08  DMA failure
     10  data error (bad CRC)
     20  controller failed
     40  seek failed
     80  attachment failed to respond
AL = BIOS error code if CF set
     00  write protect error
     01  unknown unit
     02  drive not ready
     03  unknown command
     04  data error (bad CRC)
     05  bad request structure length
     06  seek error
     07  unknown media type
     08  sector not found
     0A  write fault
     0B  read fault
     0C  general failure
```

Control Block Format (DOS 4.x):

Offset	Size	Description
00	dword	starting sector
04	word	number of sectors to read

```
          06   dword  pointer to buffer


        - reads disk sectors into buffer at DS:BX or DS:[BX+6]
        - after calling this interrupt the flags register remains
on the
          stack and must be popped manually
        - sectors are logical sectors starting at the beginning of a
          logical disk; each DOS partition on a drive unit is
considered
          one logical drive and has it's own logical sector numbers
with
          track 0 starting at the first track in the partition
        - this function uses logical drives, and is susceptible to
ASSIGN
        - physical sector numbers can be converted to and from DOS
sector
          numbers with the following formulas:

          dos_sector = (sector - 1) + (head * sectors_per_track) +
                       (track * sectors_per_track * num_heads)

          physical_sector = 1 + (dos_sector  MOD  sectors_per_track)
          physical_head = (dos_sector / sectors_per_track)  MOD
num_heads
          physical_track = dos_sector / (sectors_per_track *
num_heads)

        - see   INT 13,STATUS
```

INT 26 - Absolute Disk Write

```
AL = logical drive number (0=A:, 1=B:, 2=C:, ...)
CX = number of sectors to write
   = -1 if DOS 4.x (control block pointer is in DS:BX)
DX = starting logical sector number (see below for DOS 4.x+)
DS:BX = pointer to data buffer
      = pointer to control block (DOS 4.x+, see below)

on return:
AH = error code if CF set:
     01  bad command
     02  bad address mark
     03  write protect
     04  sector not found
     08  DMA failure
     10  data error (bad CRC)
     20  controller failed
     40  seek failed
     80  attachment failed to respond
AL = BIOS error code if CF set
     00  write protect error
     01  unknown unit
     02  drive not ready
     03  unknown command
     04  data error (bad CRC)
     05  bad request structure length
     06  seek error
     07  unknown media type
     08  sector not found
     0A  write fault
     0B  read fault
     0C  general failure
```

Control Block Format (DOS 4.x):

Offset	Size	Description
00	dword	starting sector
04	word	number of sectors to write

```
          06   dword  pointer to buffer


        - writes disk sectors from buffer at DS:BX or DS:[BX+6]
        - after calling this interrupt the flags register remains
on the
          stack and must be popped manually
        - sectors are logical sectors starting at the beginning of a
          logical disk; each DOS partition on a drive unit is
considered
          one logical drive and has it's own logical sector numbers
with
          track 0 starting at the first track in the partition
        - this function uses logical drives, and is susceptible to
ASSIGN
        - physical sector numbers can be converted to and from DOS
sector
          numbers with the following formulas:

          dos_sector = (sector - 1) + (head * sectors_per_track) +
                       (track * sectors_per_track * num_heads)

          physical_sector = 1 + (dos_sector  MOD  sectors_per_track)
          physical_head = (dos_sector / sectors_per_track)  MOD
num_heads
          physical_track = dos_sector / (sectors_per_track *
num_heads)

        - see   INT 13,STATUS
```

INT 2A,87 - Critical Section

```
AH = 87h
AL = 00  start critical section
   = 01  end critical section


returns nothing


- specifies the start or end of a critical section
- called by PRINT.COM
- see  Bibliography reference to "Undocumented DOS"
```

INT 2F,80 - PRINT.COM Time Slice Allotment (Undocumented DOS 3.1+)

```
AL = 80h
AH = 00


returns nothing


- DOS PRINT.COM is given a time slice
```

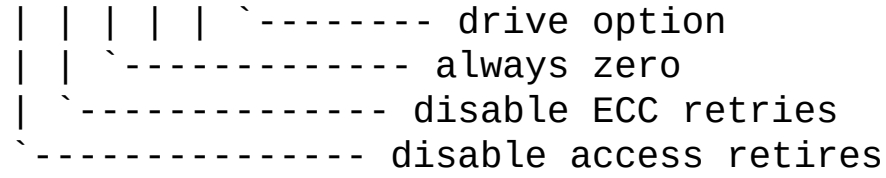
INT 13,9 - Initialize Fixed Disk Table (XT & newer)

```
AH = 09
DL = fixed disk number  (80h=drive 0, 81h=drive 1)


on return:
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error


- sets specified fixed disk table to the default values
from ROM
- many good programming references indicate this function
is only
  available on the AT, PS/2 and later systems, but all hard
disk
  systems since the XT have this function available
- INT 41h vector is pointer to table for drive 0
- INT 46h vector is pointer to table for drive 1
```

Table definitions located by interrupt vectors 41H & 46H

Offset	Size	Description
00	word	maximum number of cylinders
02	byte	maximum number of heads
03	word	starting reduced write current cylinder
05	word	starting write pre-comp cylinder
07	byte	maximum ECC data burst length
08	byte	control byte:

```
|7|6|5|4|3|2|1|0|  Control byte
```


| | | | | `-------- drive option
| | `------------- always zero
| `-------------- disable ECC retries
`--------------- disable access retires

INT 4A - RTC Alarm Handler Vector

- not a true interrupt, but a pointer to an alarm handler routine
- called by INT 70 and enabled by INT 1A,6
- routine must exit via IRET

INT 70 - Real Time Clock Interrupt (XT 286,AT, PS/2)

- called 1024 times per second for periodic and alarm functions
- decrements a DWORD counter by 976æ sec (1/1024)
- when DWORD reaches zero, bit 7 of designated wait flag is set
- if alarm was enabled by INT 1A,6, INT 4A is called when counter reaches zero to activate alarm handler
- not available in model 30 PS/2

- see INT 15,83 INT 15,86

INT 1A,6 - Set Real Time Clock Alarm (XT 286, AT,PS/2)

```
AH = 06
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD


on return:
CF = 1 if alarm already set or clock inoperable


- alarm setting is not relative like some sources claim, but the
  actually clock time the interrupt should occur
- settings values must be in BCD
- when the alarm time is reached INT 4A is executed
- INT 4A vector should be replaced with address of the alarm
  handling interrupt routine before setting the alarm
- INT 1A,7 should be called to disable the RTC alarm
```

INT 1A,7 - Disable Real Time Clock Alarm (XT, AT,PS/2)

```
AH = 07

returns nothing

- should be called previous to setting or resetting the
  RTC alarm with INT 1A,6
```

INT 15,83 - Event Wait

```
        AH = 83h
        AL = 0  set interval
           = 1  cancel interval (PS/2)
        CX,DX = number of microseconds to wait (976 æs resolution)
        ES:BX = pointer to memory flag (bit 7 is set
                when interval expires)
                (granularity is 976 microseconds)


        on return:
        CF = 1 if function already busy
        AH = 80h for PC
           = 86h for XT & AT


        - available on the AT (BIOS after 1/10/84) and PS/2's
except 30
        - designed for use in simple multitasking
```

INT 15,86 - Elapsed Time Wait (AT and PS/2)

```
AH = 86h
CX,DX = number of microseconds to wait (976 æs resolution)


on return:
CF = set if error (PC,PCjr,XT)
   = set if wait in progress
   = clear if successful wait
AH = 80h for PC and PCjr
   = 86h for XT


- AT and PS/2 only for system timing
- not designed for user application usage
```

INT 67,60 - LIM EMS Get Physical Window Array

```
AH = 60h
ES:DI = pointer to physical window array


on return:
AH = 00  success
   = error status  (see EMS STATUS)
```

INT 67,40 - Get EMM Status (LIM EMS 3.0+)

```
        AH = 40h

        on return:
        AH = status

             00  success
             80  internal software error
             81  hardware malfunction
             83  unallocated or invalid handle
             84  undefined function requested
             85  no handles available
             86  error in save or restore of mapping context
             87  more pages requested than physically exist
             88  more pages requested than currently available
             89  zero pages requested
             8A  invalid logical page number
             8B  illegal physical page number
             8C  context stack out of memory
             8D  context save failed, handle already has context
stack
             8E  context restore failed, no context stack for handle
             8F  invalid subfunction
             90  undefined attribute type
             91  feature not supported
             92  success, portion of the source region was
overwritten
             93  length of source or destination region exceeds
length of
                 region allocated to either source or destination
handle
             94  conventional and expanded memory regions overlap
             95  offset within logical page exceeds size of logical
page
             96  region length exceeds 1M
             97  source and destination regions have same handle
and overlap
             98  memory source or destination type undefined
             9A  specified DMA register set not supported ???
                 specified alternate map register set not supported
             9B  all DMA register sets currently allocated ???
                 all alternate map register sets currently allocated
```

```
            9C  alternate DMA sets not supported ???
                alternate map register sets not supported
            9D  undefined or unallocated DMA register set ???
                undefined or unallocated alternate map register set
            9E  dedicated DMA channels not supported
            9F  specified dedicated DMA channel not supported
            A1  duplicate handle name
            A2  attempted to wrap around 1Mb conventional address
space
            A3  contents of partial page map corrupted or count of
                mappable segments exceeds total mappable segments
            A4  operating system denied access


        - registers not listed are preserved
        - should only be used after establishing that the EMS driver
          is present
```

INT 10,0 - Set Video Mode

```
        AH = 00
        AL = 00  40x25 B/W text (CGA,EGA,MCGA,VGA)
           = 01  40x25 16 color text (CGA,EGA,MCGA,VGA)
           = 02  80x25 16 shades of gray text (CGA,EGA,MCGA,VGA)
           = 03  80x25 16 color text (CGA,EGA,MCGA,VGA)
           = 04  320x200 4 color graphics (CGA,EGA,MCGA,VGA)
           = 05  320x200 4 color graphics (CGA,EGA,MCGA,VGA)
           = 06  640x200 B/W graphics (CGA,EGA,MCGA,VGA)
           = 07  80x25 Monochrome text (MDA,HERC,EGA,VGA)
           = 08  160x200 16 color graphics (PCjr)
           = 09  320x200 16 color graphics (PCjr)
           = 0A  640x200 4 color graphics (PCjr)
           = 0B  Reserved (EGA BIOS function 11)
           = 0C  Reserved (EGA BIOS function 11)
           = 0D  320x200 16 color graphics (EGA,VGA)
           = 0E  640x200 16 color graphics (EGA,VGA)
           = 0F  640x350 Monochrome graphics (EGA,VGA)
           = 10  640x350 16 color graphics (EGA or VGA with 128K)
                 640x350 4 color graphics (64K EGA)
           = 11  640x480 B/W graphics (MCGA,VGA)
           = 12  640x480 16 color graphics (VGA)
           = 13  320x200 256 color graphics (MCGA,VGA)
           = 8x  EGA, MCGA or VGA ignore bit 7, see below
           = 9x  EGA, MCGA or VGA ignore bit 7, see below


        - if AL bit 7=1, prevents EGA,MCGA & VGA from clearing
display
        - function updates byte at 40:49;  bit 7 of byte 40:87
          (EGA/VGA Display Data Area) is set to the value of AL bit
7
```

Video Information Tables

BIOS Data Area Fields

Address	Size	Description
40:49	byte	Current video mode (see VIDEO MODE)
40:4A	word	Number of screen columns
40:4C	word	Size of video regen buffer in bytes
40:4E	word	Starting address in video regen buffer (offset)
40:50	8 words	Cursor position of pages 1-8, high order byte=row, low order byte=column
40:60	byte	Ending (bottom) scan line for cursor
40:61	byte	Starting (top) scan line for cursor
40:62	byte	Active display page number
40:63	word	Base port address for active 6845 CRT controller 3B4h = mono, 3D4h = color
40:65	byte	6845 CRT mode control register value (port 3x8h) EGA/VGA values emulate those of the MDA/CGA
40:66	byte	CGA current color palette setting (port 3d9h) EGA and VGA values emulate the CGA
40:84	byte	Rows on the screen (less 1, EGA+)
40:85	word	Point height of character matrix (EGA+)
40:87	byte	Video mode options (EGA+)

```
                |7|6|5|4|3|2|1|0| Video mode options (EGA+)
                 | | | | | | | `-- 1=alphanumeric cursor emulation
enabled
                 | | | | | | `--- 1=video subsystem attached to
monochrome
                 | | | | | `---- reserved
                 | | | | `----- 1=video subsystem is inactive
                 | | | `------ reserved
                 | `---------- video RAM  00-64K  10-192K  01-128K
11-256K
                 `----------- video mode number passed to INT 10,
function 0
```

40:88	byte	EGA feature bit switches, emulated on VGA

```
                |7|6|5|4|3|2|1|0| EGA feature bit switches (EGA+)
                 | | | | | | | `-- EGA SW1 config (1=off)
                 | | | | | | `--- EGA SW2 config (1=off)
                 | | | | | `---- EGA SW3 config (1=off)
                 | | | | `----- EGA SW4 config (1=off)
                 | | | `------ Input FEAT0 (ISR0 bit 5) after
output on FCR0
                 | | `------- Input FEAT0 (ISR0 bit 6) after output
on FCR0
                 | `-------- Input FEAT1 (ISR0 bit 5) after output
on FCR1
                 `--------- Input FEAT1 (ISR0 bit 6) after output
on FCR1

        40:89   byte    Video display data area (MCGA and VGA)

                |7|6|5|4|3|2|1|0| Video display data area (MCGA and
VGA)
                 | | | | | | | `-- 1=VGA is active
                 | | | | | | `--- 1=gray scale is enabled
                 | | | | | `---- 1=using monochrome monitor
                 | | | | `----- 1=default palette loading is
disabled
                 | | | `------ see table below
                 | | `------- reserved
                 | `--------  1=display switching enabled
                 `--------- alphanumeric scan lines (see table
below)

                Bit7   Bit4     Scan Line information
                 0      0       350 line mode
                 0      1       400 line mode
                 1      0       200 line mode
                 1      1       reserved

        40:8A   byte    Display Combination Code (DCC) table index
(EGA+)
        40:A8  dword    BIOS Video Save/Override Pointer Table
address
```

EGA / VGA Related Tables

Video Save/Override Pointer Table (pointer at 40:A8):

```
00  dword  Video Parameter Table pointer
04  dword  Dynamic Parameter Save Area pointer (EGA, VGA)
08  dword  Alphanumeric Character Set Override pointer
0C  dword  Graphics Character Set Override pointer
10  dword  Secondary Save Pointer Table pointer (VGA)
14  dword  reserved, set to 0000:0000
18  dword  reserved, set to 0000:0000
```

Video Parameter Table

```
00   byte    number of displayed character columns
01   byte    number of displayed screen rows minus 1
02   byte    character matrix height in points
03   word    video buffer size in bytes
05   dword   contents of sequencer registers 1-4
09   byte    misc. output register values
0A  25bytes  contents of CRTC registers 0-18h
23  20bytes  contents of attribute controller regs 0-13h
37  9 bytes  contents of graphics controller regs 0-8
```

Dynamic Parameter Save Area

```
00  16bytes   contents of graphics controller pallette regs
10    byte    contents of graphics controller overscan reg
11  239bytes  reserved
```

Alphanumeric Character Set Override

```
00   byte    length of each character definition in bytes
01   byte    character generator RAM bank
02   word    count of characters defined
04   word    first character code in table
06   dword   pointer to character font definition table
0A   byte    number of character rows displayed
0B   nbytes  array of applicable video modes
0B+n byte    FFh end of mode list marker
```

Graphics Character Set Override

```
00   byte    count of displayed character rows
01   word    length of each character definition in bytes
03   dword   pointer to character font definition table
07   nbytes  array of applicable video modes
07+n byte    FFh end of mode list marker
```

Secondary Save Pointer Table

```
00  word   length of table in bytes
02  dword  pointer to display combination code table
06  dword  pointer to secondary alphanumeric char set
override
0A  dword  pointer to user palette profile table  (VGA)
0E  dword  reserved
12  dword  reserved
16  dword  reserved
```

Display Combination Code Table

```
00   byte   number of table entries
01   byte   DCC table version number
02   byte   maximum display type code
03   byte   reserved
04  n words array valid display combinations:

0,0 entry  0  no display
0,1 entry  1  MDPA
0,2 entry  2  CGA
2,1 entry  3  MDPA + CGA
0,4 entry  4  EGA
4,1 entry  5  EGA + MDPA
0,5 entry  6  MEGA
2,5 entry  7  MEGA + CGA
0,6 entry  8  PGC
1,6 entry  9  PGC + MDPA
5,6 entry 10  PGC + MEGA
0,8 entry 11  CVGA
1,8 entry 12  CVGA + MDPA
0,7 entry 13  MVGA
2,7 entry 14  MVGA + CGA
2,6 entry 15  MVGA + PGC
```

Secondary Alpha Mode Auxillary Character Generator Table

```
00   byte    bytes per character
01   byte    block to load
02   byte    reserved
03   dword   font table pointer
07   nbytes  array of mode values for this font
07+n byte    FFh end of mode list marker
```

Palette Profile Table (VGA only)

```
00   byte    1 - enable underlining in all alphanumeric
modes
             0 - enable underlining in monochrome alpha
modes
            -1 - disable underlining in all alpha modes
01   byte   reserved
02   word   reserved
04   word   count of attribute controller regs in table
06   word   first attribute controller register number
08   dword  pointer to attribute controller reg table
0C   word   count of video DAC color registers in table
0E   word   first video DAC color register number
10   dword  video DAC color register table pointer
14   nbytes array of applicable video modes for this font
14+n byte   FFh end of video mode list marker
```

- see also BIOS Data Area
- see INT 10,1B for Dynamic Video State Table and Video Static
  Functionality Table

INT 10,1B - Video BIOS Functionality and State Information (MCGA/VGA)

```
AH = 1B
BX = implementation type (must be zero)
ES:DI = pointer to 64 byte buffer


on return:
AL = 1B
ES:DI = pointer to updated buffer  (see below)
```

- returns static and dynamic information about the current state and capabilities of the current video system
- bytes 0-3 of the dynamic data table at ES:DI contain a far pointer to the video static information table

Video BIOS Dynamic Functionality State Table (MCGA/VGA)

Dynamic Video State Table

```
00  dword   address of static functionality table
04  byte    video mode
05  word    number of columns
07  word    length of displayed video buffer (# bytes)
09  word    start address of upper left corner of video
buffer
0B  16bytes cursor position table for 8 pages (col,row)
1B  byte    cursor end line
1C  byte    cursor start line
1D  byte    active video page
1E  word    I/O port for CRTC address register
20  byte    current value of CRTC 3x8 register
21  byte    current value of CRTC 3x9 register
```

```
        22  byte    number of displayed character rows
        23  word    height of character matrix (points)
        25  byte    active display combination code
        26  byte    inactive display combination code
        27  word    number of displayed colors (mono = 0)
        29  byte    number of supported video pages
        2A  byte    raster scan lines 0=200, 1=350, 2=400, 3=480
        2B  byte    text character table used
        2C  byte    text character table used
        2D  byte    other state information:

                |7|6|5|4|3|2|1|0| State Information byte at offset
2D
                 | | | | | | | `--- 1 = all modes active (MCGA
always 0)
                 | | | | | | `---- 1 = gray scale summing enabled
                 | | | | | `----- 1 = monochrome display attached
                 | | | | `------ 1 = default palette loading
disabled
                 | | | `------- 1 = cursor emulation enabled
                 | | `-------- 1 = blinking attribute enabled
                 `------------ 1 = reserved

        2E 3bytes   reserved
        31  byte    video RAM available 0=64K, 1=128K, 2=192K,
3=256K
        32  byte    save area status

                |7|6|5|4|3|2|1|0| Save Area Status
                 | | | | | | | `--- 1 = two text char sets are
active
                 | | | | | | `---- 1 = dynamic save area is active
                 | | | | | `----- 1 = text char set override is
active
                 | | | | `------ 1 = graphics char set is override
active
                 | | | `------- 1 = palette override is active
                 | | `-------- 1 = display combination code ext.
active
                 `------------ 1 = reserved

        33  dword   reserved
```

Video BIOS Static Functionality Table (EGA/ VGA)

```
                |7|6|5|4|3|2|1|0| Video modes supported, byte at
offset 00
                 | | | | | | | `--- 1 = mode 0
                 | | | | | | `---- 1 = mode 1
                 | | | | | `----- 1 = mode 2
                 | | | | `------ 1 = mode 3
                 | | | `------- 1 = mode 4
                 | | `-------- 1 = mode 5
                 | `--------- 1 = mode 6
                 `---------- 1 = mode 7

                |7|6|5|4|3|2|1|0| Video modes supported, byte at
offset 01
                 | | | | | | | `--- 1 = mode 8
                 | | | | | | `---- 1 = mode 9
                 | | | | | `----- 1 = mode A
                 | | | | `------ 1 = mode B
                 | | | `------- 1 = mode C
                 | | `-------- 1 = mode D
                 | `--------- 1 = mode E
                 `---------- 1 = mode F

                |7|6|5|4|3|2|1|0| Video modes supported, byte at
offset 02
                 | | | | | | | `--- 1 = mode 10
                 | | | | | | `---- 1 = mode 11
                 | | | | | `----- 1 = mode 12
                 | | | | `------ 1 = mode 13
                 `------------- reserved

        03  dword  reserved
        07  byte   scan lines supported in text modes

                |7|6|5|4|3|2|1|0| Scan lines supported, byte at
offset 07
                 | | | | | | | `--- 1 = 200 lines
                 | | | | | | `---- 1 = 350 lines
                 `--------------- 1 = 400 lines

        08  byte   max number of displayable text character sets
```

```
        09  byte   # of text definition tables in char generator RAM
        0A  byte   other capability flags

                |7|6|5|4|3|2|1|0| Other flags, byte at offset 0A
                 | | | | | | | `--- 1 = all modes (0 on MCGA)
                 | | | | | | `---- 1 = gray scale summing
                 | | | | | `----- 1 = character set loading
                 | | | | `------ 1 = default palette loading
                 | | | `------- 1 = cursor emulation
                 | | `-------- 1 = 64 color palette
                 | `--------- 1 = video DAC loading
                 `---------- 1 = DAC controlled by ACCS

        0B  byte   other capability flags

                |7|6|5|4|3|2|1|0| Other flags, byte at offset 0B
                 | | | | | | | `--- 1 = light pen support
                 | | | | | | `---- 1 = save/restore video state
                 | | | | | `----- 1 = blinking/background intensity
                 | | | | `------ 1 = display combination code
                 `------------- reserved

        0C  word   reserved
        0E  byte   save area capabilities

                |7|6|5|4|3|2|1|0|  save area capabilities at offset
0E
                 | | | | | | | `---- 1 = multiple text character
sets
                 | | | | | | `----- 1 = dynamic save area
                 | | | | | `------ 1 = text character set override
                 | | | | `------- 1 = graphics character set
override
                 | | | `-------- 1 = palette override
                 | | `--------- 1 = display combination code
extension
                 `------------ reserved

        0F  byte    reserved
```

INT 1E - Disk Initialization Parameter Table Vector

- not an interrupt, but a far pointer to the diskette base table
- this table contains initialization parameters for the disk controller used during formatting, reading, and writing to a disk

Disk Initialization Parameter Table Vector Definition:

Offset Description

00 |7|6|5|4|3|2|1|0| Disk Controller Mode byte 1
| | | | `---------- head step rate in milliseconds (0-F)
`----------------- head unload time in milliseconds (0-F)

01 |7|6|5|4|3|2|1|0| Disk Controller Mode byte 2
| | | | | | | `---- DMA flag (0 = use DMA)
`----------------- (head load time/2)-1 in milliseconds

02 clock ticks until motor off
03 FM or MFM Mode: Log2 (Bytes per Sector/128)
FM=Frequency Modulation used on single-density disks
MFM=Modified Frequency Modulation used on double-sided disks
04 last sector on track
05 gap length in bytes:

MFM	Bytes Per Sector	Sectors Per Track	Write Gap	Format Gap
1	256	18	0Ah	0Ch
1	256	16	20h	32h
2	512	8	2Ah	50h
2	512	9	1Bh	6Ch
3	1024	4	80h	F0h
4	2048	2	C8h	FFh
5	4096	1	C8h	FFh

```
  06  disk data length
      80h for 128 bytes/sector
      FFh otherwise
  07  gap length when formatting
  08  data pattern used during formatting
  09  floppy head bounce delay (in milliseconds)
  0A  floppy motor start delay (in 0.125 second intervals)
  0B  floppy motor stop delay (in 0.25 second intervals)
```

- see INT 13,18

INT 13,18 - Set Media Type for Format (BIOS date specific)

```
        AH = 18h
        CH = lower 8 bits of number of tracks  (0-1023 dec., see
below)
        CL = sectors per track (1-17 dec., see below)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        ES:DI = pointer to 11-byte Disk Base Table (DBT)
        AH = 00h if requested combination supported
           = 01h if function not available
           = 0Ch if not supported or drive type unknown
           = 80h if there is no media in the drive
        CF = 0 if successful
           = 1 if error


        - valid only for XT BIOS dated after 1/10/86, AT after
11/15/86,
          XT 286 and the PS/2 line
        - only disk number is checked for validity
        - track number is a 10 bit value taken from the 2 high order
          bits of CL and the 8 bits in CH (low order 8 bits of
track):

          |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  CX
           | | | | | | | | | | `--------------  sectors per track
count
           | | | | | | | | `--------------  high order 2 bits track/
cyl count
           `--------------------------  low order 8 bits of track/
cyl count

        - see   INT 13,STATUS
```

Machine Identification Byte

Model	Sub-Model	Machine
FF	??	Original IBM PC 4/24/81
	??	IBM PC 10/19/81
	??	IBM PC 10/27/82
FE	??	IBM XT (Original)
	??	IBM portable PC
	??	Compaq DeskPro
FD	??	PCjr
FC	??	IBM AT (6 MHz)
	01	IBM AT 3x9 (8 MHz)
	02	IBM XT 286
	04	IBM PS/2 Model 50
	05	IBM PS/2 Model 60
	0B	IBM PS/1
FB	00	IBM 256/640K XT (aka XT/2)
FA	00	IBM PS/2 Model 30
	01	IBM PS/2 Model 25
F9	00	IBM PC Convertible
F8	00	IBM PS/2 Model 80 (16 MHz)
	01	IBM PS/2 Model 80 (20 MHz)
	04	IBM PS/2 Model 70 (20 MHz)
	09	IBM PS/2 Model 70 (16 MHz)
B6	??	Hewlett Packard 110
9A	??	Compaq Plus
86		XT (BIOS 11/82+) & AT (BIOS 1/84+) (see note)
80		PC & PCjr (see note)
2D	??	Compaq PC

- found at memory location F000:FFFE or via INT 15,C0
- model values of 80 & 86 are returned by BIOS versions previous
  to the PS/2 but after the BIOS dates marked

- see INT 15,C0

INT 15,C0 - Return System Configuration Parameters (PS/2 only)

```
AH = C0

on return:
CF = 0 if successful
   = 1 if error
AH = when CF set, 80h for PC & PCjr, 86h for XT
     (BIOS after 11/8/82) and AT (BIOS after 1/10/84)

ES:BX = pointer to system descriptor table in ROM of the
format:
```

Offset	Size	Description
00	word	length of descriptor (8 minimum)
02	byte	model byte (same as F000:FFFE, not reliable)
03	byte	secondary model byte
04	byte	BIOS revision level (zero based)
05	byte	feature information, see below
06	dword	reserved

Feature Information

```
|7|6|5|4|3|2|1|0|  Feature Byte
 | | | | | | | `---- reserved
 | | | | | | `----- 0=PC bus, 1=Micro Channel
 | | | | | `------ Extended BIOS Data Area (EBDA)allocated
 | | | | `------- wait for external event supported
 | | | `-------- INT 15,4F used (kbd intercept)
 | | `---------- RTC present
 | `----------- 2nd 8259 present
 `------------ DMA channel 3 used by fixed disk BIOS
```

- see MACHINE ID

INT 15,4F - Keyboard Intercept (BIOS date specific)

```
AH = 4F
AL = scan code
CF = set to 1 (via STC instruction)

on return

AH = 80h, CF set  (PC, PCjr)
   = 86h, CF set  (XT BIOS 11/8/82, AT BIOS 1/10/84)
AL = CF set, new scan code
   = CF clear, original scancode
```

- available with XT BIOS after 11/8/82, AT BIOS after 1/10/84
- called by INT 9, makes allowance for keyboard translation
- normally returns the scan code in AL, with CF set
- if function returns with CF clear, INT 9 ignores keystroke
- do not rely on this function being called for each INT 9 since
  any user INT 9 handler can exit prematurely and circumvent this function

CPU Characteristics / Determination

8088,80188,V20 six byte prefetch queue, allows use of self modifying code to determine length of prefetch

8086,80186,V30 four byte prefetch queue, allows use of self modifying code to determine length of prefetch

8088,8086,80188,81086,v20,v30 Flag register bits 12 through 15 cannot be cleared

8088,8086 will shift left or right using all 8 bits of CL, if CL = 33, register is guaranteed to be cleared

80188,80186,80286,80386 will shift left or right using only lower 5 bits of CL. If CL = 32, the shift will not occur

80286,80386 earlier CPU's decremented SP before a PUSH SP, but 286+ pushes the value first then, decrements SP

8088,8086 non-zero multiplication result clears zero flag

V20,V30 non-zero multiplication result does not clear zero flag, set ZF before multiply, and test after, if it's still set, then it's a V20, V30

80286 allows setting of bit 15 of the flags register

80386 allows setting of bits 12 through 14 of flags register

see BIBLIO reference to PC Tech Journal, "Chips In Transition",
April 1986

Detecting ANSI.SYS

Compare segment address of a DOS interrupt with the segment of the standard CON device. If equal, then ANSI is not loaded. If the segment address of CON is greater than the DOS interrupt, then ANSI is loaded or a TSR is handling the interrupt.

- see INT 21,52 for info on CON device address

Detecting DESQView

1. call INT 2F with AX = DE00h
2. if AL = FF on return DESQView is installed

Detecting Disk Ready

1. use INT 13,4 (Verify Sector) to check ready for read
2. check for error in AH of:
   80h Time out, or Not Ready
   AAh Drive not ready
   00h drive is ready for reading
   other value indicates drive is ready, but an error occurred
3. use INT 13,2 (Read Sector) followed by INT 13,3 (Write Sector) to check ready for read/write. First read sector, test for ready; write sector back, check for 03h (write protect) or any of the other BIOS disk errors

- multiple attempts should be made to allow for diskette

motor startup
- allows testing of drive ready by bypassing DOS and critical errors
- see INT 13,STATUS

Detecting Drive Existence

1. save current drive (INT 21,19)
2. set current drive (INT 21,E)
3. get current drive (INT 21,19)
4. if current drive == drive requested
   then drive exists
   else drive doesn't exist
5. reset original drive (INT 21,E)

- no critical error is generated if invalid drive is selected

Detecting EGA Video

INT 10h, function 12h (INT 10,12), subfunction 10h (in BL) will
return BL = 10h if not an EGA since invalid functions requests
return without register modification

Detecting VGA Video

INT 10,1A can be used to detect the presence of VGA.
BL = 7 VGA with analog monochrome display
= 8 VGA with analog color display

- some enhanced EGA adapters will incorrectly identify as

VGA
when INT 10,1B is used

Detecting Extended Keyboard BIOS

1. use INT 16,5 to stuff FFFF into the keyboard buffer
2. use INT 16,10 to read the keystrokes back
3. if the data stuffed into the keyboard is returned within 16
   extended reads, the BIOS handles the extended keyboard
4. byte at 40:96 (BDA) is another method of detecting extended BIOS
   but is not reliable since some compatibles give false positives

- see BIOS DATA AREA INT 9

Detecting present of Expanded Memory Manager

1. attempt to open file/device 'EMMXXXX0' using INT 21,3D;
   if successful continue, else no EMM
2. make sure EMMXXXX0 is a character device and not a disk file
   by using IOCTL function 0 (Get Device Info); bit 7 of DX
   should be set for char device and clear for disk file
3. make sure Expanded Memory Manager is available using IOCTL
   function 7 (Get Output Status); AL should be non-zero (FF)
   if Memory Manager is available, and zero if not available.
4. call INT 67,46 to get version information

- see also IOCTL,0 IOCTL,7

Detecting Existence of Mouse Driver

1. check INT 33 vector is not 0:0
2. check INT 33 vector does not point to an IRET
3. call INT 33,0 to reset the mouse and get installed state
4. call INT 33,24 to check version, type and IRQ

Detecting Miscellaneous

Boot drive; see INT 21,33 (DOS 4.0+)

CPU type; see CPU

Hardware configuration; see INT 11

Logical Drive Assignment; see IOCTL,E

Memory size, Base; see INT 12

Memory size, Extended; see CMOS

Micro Channel Bus on PS/2; see INT 15,C0

Network or Local Disk; see IOCTL,9

Network or Local File; see IOCTL,A

Parallel ports; see BIOS DATA AREA locations 40:08-40:0F

RAM Disk; see INT 21,32 and MEDIA DESCRIPTOR

Redirection on command line; see IOCTL,0

Removable media; INT 21,1B INT 21,1C
MEDIA DESCRIPTOR BYTE and IOCTL,8

Serial ports; see BIOS DATA AREA locations 40:00-40:07

Single drive A: or B: selection; see BIOS DATA AREA byte 50:04

SUBST'ed drive; see INT 21,32

Video Screen Size in Columns; see INT 10,F

Video Screen Size in Rows; see INT 10,11 subfunction 30h (EGA+)

APPEND resident; see INT 2F,0
ASSIGN resident; see INT 2F,0
DOSSHELL resident; see INT 2F,0
NLSFUNC resident; see INT 2F,0
PRINT resident; see INT 2F,0
SHARE resident; see INT 2F,0

INT 13,4 - Verify Disk Sectors

```
        AH = 04
        AL = number of sectors to verify  (1-128 dec.)
        CH = track/cylinder number  (0-1023 dec., see below)
        CL = sector number  (1-17 dec.)
        DH = head number  (0-15 dec.)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)
        ES:BX = pointer to buffer

        on return:
        AH = status  (see INT 13,STATUS)
        AL = number of sectors verified
        CF = 0 if successful
           = 1 if error


        - BIOS disk reads should be retried at least three times
and the
          controller should be reset upon error detection
        - causes controller to calculate the CRC of the disk data
and
          compare it against the CRC stored in the sector header
        - BIOS before 11/15/85 required ES:BX point to a valid
buffer
          that doesn't cross DMA boundaries.   More recent BIOS
versions
          actually ignore the buffer and the DMA boundary
requirement
        - use this function to check for valid formatted diskette
in a
          the specified drive and for drive ready for read
        - only the disk number is checked for validity
        - the parameters in CX change depending on the number of
cylinders;
          the track/cylinder number is a 10 bit value taken from
the 2 high
          order bits of CL and the 8 bits in CH (low order 8 bits
of track):

          |F|E|D|C|B|A|9|8|7|6|5-0|  CX
           | | | | | | | | | |  `-----  sector number
```

```
          | | | | | | | `---------  high order 2 bits of track/
cylinder
          `------------------------  low order 8 bits of track/cyl
number
```

- see DETECTING

INT 13,2 - Read Disk Sectors

```
        AH = 02
        AL = number of sectors to read  (1-128 dec.)
        CH = track/cylinder number  (0-1023 dec., see below)
        CL = sector number  (1-17 dec.)
        DH = head number  (0-15 dec.)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)
        ES:BX = pointer to buffer


        on return:
        AH = status  (see INT 13,STATUS)
        AL = number of sectors read
        CF = 0 if successful
           = 1 if error


        - BIOS disk reads should be retried at least three times
and the
          controller should be reset upon error detection
        - be sure ES:BX does not cross a 64K segment boundary or a
          DMA boundary error will occur
        - many programming references list only floppy disk
register values
        - only the disk number is checked for validity
        - the parameters in CX change depending on the number of
cylinders;
          the track/cylinder number is a 10 bit value taken from
the 2 high
          order bits of CL and the 8 bits in CH (low order 8 bits
of track):

          |F|E|D|C|B|A|9|8|7|6|5-0|  CX
           | | | | | | | | | |  `-----  sector number
           | | | | | | | | `---------  high order 2 bits of track/
cylinder
           `------------------------  low order 8 bits of track/cyl
number

        - see   INT 13,A
```

INT 13,A - Read Long Sector (XT & newer)

```
AH = 0A
AL = number of sectors  (1-121 dec.)
CH = track number  (0-1023 dec., see below)
CL = sector number  (1-17 dec., see below)
DH = head number  (0-15 dec.)
DL = fixed drive number  (80h=drive 0, 81h=drive 1)
ES:BX = address of buffer


on return:
AH = status  (see INT 13,STATUS)
AL = number of sectors actually transferred
CF = 0 if successful
   = 1 if error


- BIOS disk reads should be retried at least three times
and the
  controller should be reset upon error detection
- many good programming references indicate this function
is only
  available on the AT, PS/2 and later systems, but all hard
disk
  systems since the XT have this function available
- reads regular data sectors (128-1024 bytes) with an
additional
  4 byte ECC code included
- a DMA boundary error will occur if the buffer at ES:BX
crosses
  a 64K segment boundary
- only the disk number is checked for validity
- the parameters in CX change depending on the number of
cylinders;
  the track/cylinder number is a 10 bit value taken from
the 2 high
  order bits of CL and the 8 bits in CH (low order 8 bits
of track):

  |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  CX
   | | | | | | | | | | `--------------  sector number
   | | | | | | | | `--------------  high order 2 bits of
```

```
track number
           `--------------------------  low order 8 bits of track
number
```

- see INT 13,2

INT 13,3 - Write Disk Sectors

```
        AH = 03
        AL = number of sectors to write  (1-128 dec.)
        CH = track/cylinder number  (0-1023 dec.)
        CL = sector number  (1-17 dec., see below)
        DH = head number  (0-15 dec.)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)
        ES:BX = pointer to buffer


        on return:
        AH = 0 if CF=0; otherwise disk status  (see INT 13,STATUS)
        AL = number of sectors written
        CF = 0 if successful
           = 1 if error


        - BIOS disk write attempts should reset the controller on
error
        - be sure ES:BX does not cross a 64K segment boundary or a
          DMA boundary error will occur
        - IBM PC XT 286 does not require a value in AL, though it is
          recommended that one be supplied for portability
        - many programming references list only floppy disk
register values
        - only the disk number is checked for validity
        - the parameters in CX change depending on the number of
cylinders;
          the track/cylinder number is a 10 bit value taken from
the 2 high
          order bits of CL and the 8 bits in CH (low order 8 bits
of track):

          |F|E|D|C|B|A|9|8|7|6|5-0|  CX
           | | | | | | | | | |  `-----  sector number
           | | | | | | | | `---------  high order 2 bits of track/
cylinder
           `-----------------------  low order 8 bits of track/cyl
number

        - see   INT 13,B
```

INT 13,B - Write Long Sectors (XT & newer)

```
        AH = 0B
        AL = number of sectors  (1-121 dec.)
        CH = track number  (0-1023 dec., see below)
        CL = sector number  (1-17 dec., see below)
        DH = head number  (0-15 dec.)
        DL = fixed drive number  (80h=drive 0, 81h=drive 1)
        ES:BX = address of buffer


        on return:
        AL = number of sectors actually transferred
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - BIOS disk write attempts should reset the controller on
error
        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
        - writes regular sectors (128-1024 bytes) with additional 4
byte
          ECC code included in data
        - a DMA boundary error will occur if the buffer at ES:BX
crosses
          a 64K segment boundary
        - only the disk number is checked for validity
        - the parameters in CX change depending on the number of
cylinders;
          the track/cylinder number is a 10 bit value taken from
the 2 high
          order bits of CL and the 8 bits in CH (low order 8 bits
of track):

          |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  CX
           | | | | | | | | | | `--------------  sector number
           | | | | | | | | `--------------  high order 2 bits of
track number
```

```
            `--------------------------  low order 8 bits of track
number
```

- see [INT 13,3](#)

INT 21,19 - Get Current Default Drive

```
AH = 19h


on return:
AL = current default drive (0=A,1=B,etc)


- determines the current default drive
```

INT 21,E - Select Disk

```
AH = 0E
DL = zero based, drive number (0-25, A: - Z:)


on return:
AL = one based, total number of logical drives including
     hardfiles (1-26)


- for DOS 3.x+, this function returns the number of logical
  drives or the value of LASTDRIVE (default of 5) in the
  CONFIG.SYS file
```

INT 10,12 - Video Subsystem Configuration (EGA/VGA)

```
AH = 12h


BL = 10  return video configuration information

on return:
BH = 0 if color mode in effect
   = 1 if mono mode in effect
BL = 0 if 64k EGA memory
   = 1 if 128k EGA memory
   = 2 if 192k EGA memory
   = 3 if 256k EGA memory
CH = feature bits
CL = switch settings


BL = 20  select alternate print screen routine

BL = 30  select scan lines for alphanumeric modes
   AL = 0  200 scan lines
      = 1  350 scan lines
      = 2  400 scan lines

on return:
AL = 12


BL = 31  select default palette loading
   AL = 0 enable default palette loading
      = 1 disable default palette loading

on return:
AL = 12


BL = 32  CPU access to video RAM
   AL = 0  enable CPU access to video RAM and I/O ports
      = 1  disable CPU access to video RAM and I/O ports
```

```
on return:
AL = 12
```

BL = 33 Gray scale summing

```
   AL = 0  enable gray scale summing
      = 2  disable gray scale summing

on return:
AL = 12
```

BL = 34 cursor emulation

```
   AL = 0  enable cursor emulation
      = 1  disable cursor emulation

on return:
AL = 12
```

BL = 35 PS/2 video display switching

```
   AL = 0 initial adapter video off
      = 1 initial planar video on
      = 2 switch active video off
      = 3 switch inactive video on
   ES:DX pointer to 128 byte save area (when AL = 0, 2 or 3)

on return:
AL = 12
```

BL = 36 video refresh control

```
   AL = 0 enable refresh
      = 1 disable refresh

on return:
AL = 12
```

INT 10,1A - Video Display Combination (VGA)

```
AH = 1A
AL = 00 get video display combination
   = 01 set video display combination
     BL = active display  (see table below)
     BH = inactive display


on return:
AL = 1A, if a valid function was requested in AH
BL = active display  (AL=00, see table below)
BH = inactive display  (AL=00)
```

Valid display codes:

```
 FF  Unrecognized video system
 00  No display
 01  MDA with monochrome display
 02  CGA with color display
 03  Reserved
 04  EGA with color display
 05  EGA with monochrome display
 06  Professional graphics controller
 07  VGA with analog monochrome display
 08  VGA with analog color display
 09  Reserved
 0A  MCGA with digital color display
 0B  MCGA with analog monochrome display
 0C  MCGA with analog color display

- returns value at byte 40:8A indicating display
combination status
- used to detect video display capabilities
```

INT 16,5 - Keyboard Buffer Write (AT+)

```
AH = 05
CH = scan code
CL = ASCII character


on return:
AL = 00 if success
     01 if buffer full


- available on AT and PS/2 machines with extended keyboard
support
- stores normal keystroke into keyboard buffer
- will not store attribute keys like Shift, Alt, Ctrl,
etc...
```

INT 67,46 - Get EMM Version (LIM EMS)

```
AH = 46h


on return:
AH = 00  success
   = error code (see EMS STATUS)
AL = EMM version number in BCD


- upper four bits of AL contain the BCD major version
- lower four bits of AL contain the BCD minor version
- registers not listed are preserved
```

INT 21,44,7 / IOCTL,7 - Get Output Status

```
AH = 44h
AL = 07
BX = handle


on return
CF = 0 if successful
   = 1 if error
AX = error code if CF set  (see DOS ERROR CODES)
AL = 00  if ready (files), or not ready (char devices)
   = FF  ready (files or char device)


- indicates if a device or file is ready for output
- files always return ready, character devices don't
```

INT 21,33 - Get/Set System Values (Ctl-Break/ Boot Drive)

```
        AH = 33h
        AL = 00 to get Ctrl-Break checking flag
           = 01 to set Ctrl-Break checking flag
           = 02 to set extended Ctrl-Break checking
           = 05 get boot drive (DOS 4.x)
        DL = 00 to set Ctrl-Break checking off
           = 01 to set Ctrl-Break checking on
           = boot drive for subfunction 5;  (1=A:, 2=B:, ...)


        on return:
        DL = 00 Ctrl-Break checking OFF (AL=0 or AL=2)
           = 01 Ctrl-Break checking ON  (AL=0 or AL=2)
           = boot drive number (1-26, A: - Z:) (function 05)


        - retrieves DOS Ctrl-Break or extended Ctrl-Break setting
which
          determines if DOS will check for Ctrl-Break during INT 21
calls
```

INT 21,44,E / IOCTL,E - Get Logical Drive

```
        AH = 44h
        AL = 0E
        BL = physical drive number (0=default, 1=A:, 2=B:,
3=C:, ...)
```

```
        on return
        AX = error code if CF set  (see DOS ERROR CODES)
        AL = logical drive number assigned to physical drive
           = 0 if drive accessed by only one drive specifier
           = 1 if drive A:, 2 if drive B:, etc
```

```
        - used to determine the last drive designator used to access
          a drive if more than one logical drive designation applies
          to a device
        - implemented from DOS 3.2
```

INT 21,44,9 / IOCTL,9 - Device Local or Remote Query

```
AH = 44h
AL = 09
BL = drive number (0=default, 1=A:, 2=B:, 3=C:, ...)


on return
AX = error code if CF set  (see DOS ERROR CODES)
DX = device attribute word
     bit 12 = 1 if drive is remote
     bit 12 = 0 if drive is local


- used to determine if block device is local or remote
- returns invalid function if networking not started
- implemented from DOS 3.1
```

INT 21,44,A / IOCTL,A - Handle Local or Remote Query

```
AH = 44h
AL = 0A
BX = handle


on return
AX = error code if CF set  (see DOS ERROR CODES)
DX = device attribute word
     bit 15 = 1 if drive is remote
     bit 15 = 0 if drive is local


- used to determine if block device is local or remote
- returns invalid function if networking not started
- implemented from DOS 3.1
```

INT 21,44,8 / IOCTL,8 - Device Removable Query

```
        AH = 44h
        AL = 08
        BL = drive number (0=default, 1=A:, 2=B:, 3=C:, ...)


        on return
        CF = 0 if successful
           = 1 if error
        AX = 00  removable media
           = 01  non-removable media
           = error code if CF set  (see DOS ERROR CODES)


        - used to determine if a device supports removable media
        - RAM disks are not considered removable media
        - device drivers compatible with DOS 2.0 do not always
respond
          correctly to this query
        - implemented from DOS 3.0;  for earlier DOS versions drive
A: and
          B: are removable media since DRIVER.SYS/SUBST are DOS 3.0
+ only
```

INT 10,11 - Character Generator Routine (EGA/ VGA)

```
AH = 11h

AL = 00  user character load
   BH = number of bytes per character
   BL = table in character generator RAM
   CX = count of characters in table
   DX = ASCII code of first character defined
   ES:BP = pointer to user table


AL = 01  ROM BIOS 8x14 monochrome set
   BL = table in character generator RAM


AL = 02  ROM BIOS 8x8 double dot
   BL = table in character generator RAM


AL = 03  set displayed definition table
   BL = value for character Map Select register (EGA,VGA)
      = character generator RAM table numbers (MCGA)


AL = 04  ROM BIOS 8x16 character set
   BL = table in character generator RAM


AL = 10  user specified character definition table
   BH = bytes per character (points)
   BL = table in character generator RAM
   CX = number of characters defined in table
   DX = ASCII code of first character defined
   ES:BP = pointer to user table


AL = 11  ROM BIOS 8x14 monochrome character set
   BL = table in character generator RAM
```

AL = 12 ROM 8x8 double dot character definitions
BL = table in character generator RAM

AL = 14 ROM 8x16 double dot character definitions
BL = table in character generator RAM

AL = 20 pointer to graphics character table for ~INT 1F~ (8x8)
ES:BP = pointer to user table

AL = 21 user graphics character pointer at INT 43
BL = row specifier
= 0 - user specified (DL = rows)
= 1 is 14 rows
= 2 is 25 rows
= 3 is 43 rows
CX = bytes per character (points)
DL = rows (when BL = 0)
ES:BP = pointer to user table

AL = 22 ROM 8x14 character set
BL = number of rows (see AL=21)
DL = rows (when BL = 0)

AL = 23 ROM 8x8 double dot character set
BL = row specifier (see AL=21)
DL = rows (when BL = 0)

AL = 24 ROM 8x16 character set
BL = row specifier (see AL=21)
DL = rows (when BL = 0)

AL = 30 get current character generator information
BH = information desired:
= 0 INT 1F pointer
= 1 INT 44h pointer
= 2 ROM 8x14 pointer
= 3 ROM 8x8 double dot pointer (base)
= 4 ROM 8x8 double dot pointer (top)

```
        = 5  ROM 9x14 alpha alternate pointer
        = 6  ROM 8x16 character table pointer
        = 7  ROM 9x16 alternate character table pointer

on return:
CX = bytes per character (points)
DL = rows (less 1)
ES:BP = pointer to table
```

INT 1F - Graphics Display Character Bit Map Table

- not a true interrupt, but a far pointer to a table of character bit maps for the graphics mode representation of ASCII characters 128 through 255
- can be used to change character fonts in CGA graphics mode

INT 2F,0 - DOS Multiplex Interrupt - Get Installed State

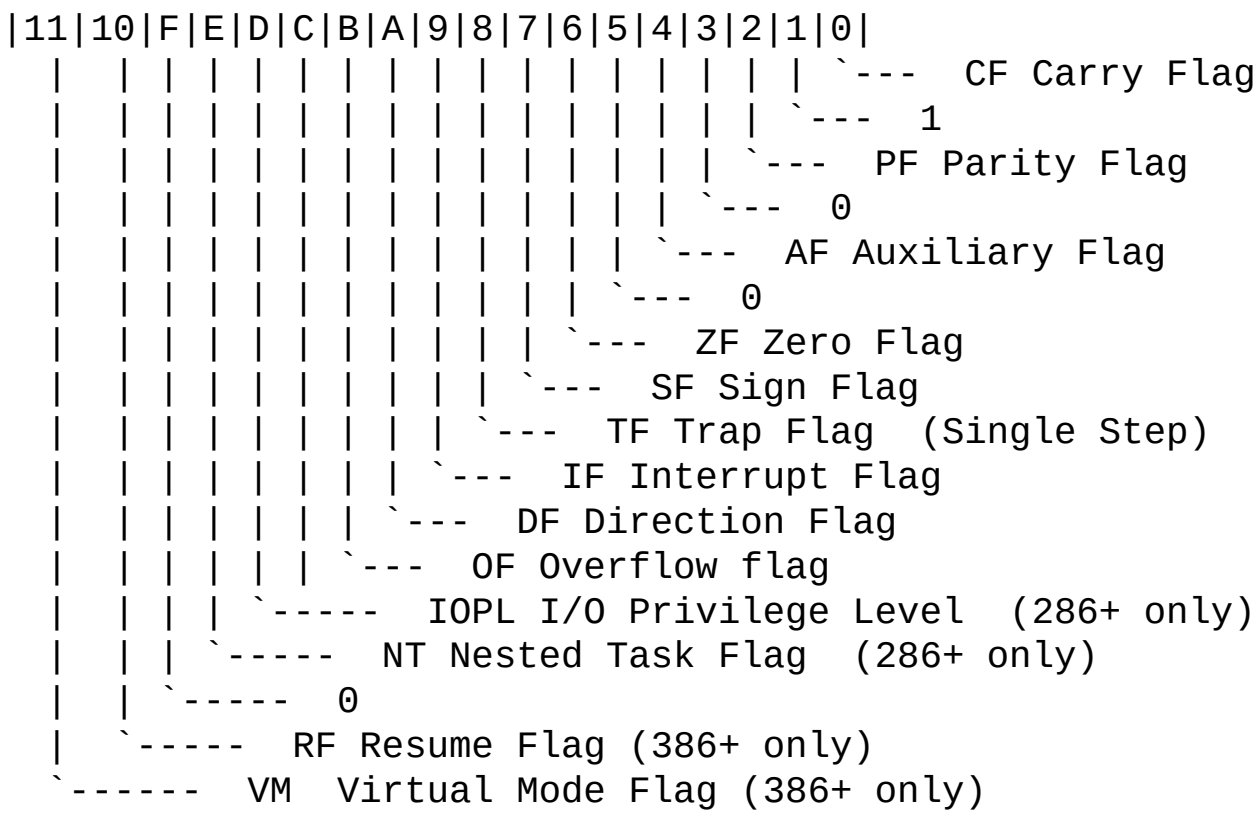
```
AL = 00
AH = 00  DOS 2.x PRINT.COM (undocumented, see note below)
     01  PRINT  (undocumented behavior also, see note below)
     05  Critical error handler (undocumented DOS 3.0+)
     06  ASSIGN  (see note)
     08  DRIVER.SYS (undocumented DOS 3.2+)
     10  SHARE
     11  Redirector/IFS is resident
     12  MultiDOS is resident
     13  Swap 13h and 19h
     14  NLSFUNC
     15  GRAPHICS.COM (see note below)
         MS CD-ROM extension (MSCDEX) (see note below)
     16  Windows 386
     17  DOS shell
     19  SHELLB.COM (DOS 4.0 only)
     1A  ANSI.SYS (DOS 4.0+)
     1B  XMA2EMS.SYS
     40  OS/2 compatibility box
     43  Himem XMS driver
     7A  Novell Netware IPX
     AD  DISPLAY.SYS internal (DOS 3.3+)
     AE  DOS installable command extension (DOS 3.3+)
     B0  GRAFTABL.COM (DOS 3.3+)
     B4  IBM 3270 emulation
     B7  APPEND  (DOS 3.3+)
     B8  LAN existence
     B9  LAN RECEIVER.COM
     BF  LAN REDIRIFS.EXE
     C0-FF reserved for user applications


on return:
AL = 00  not installed, ok to install
   = 01  not installed, do NOT install
   = FF  installed


- the installed test for append returns non-zero if
```

installed;
          zero if not installed
        - AH = 00 and 01 are used by DOS 2.x PRINT.COM and is not a check
          for installed state, see BIBLIO reference to "Undocumented DOS"
        - ASSIGN returns non-zero if installed
        - installation checks with AH=15 do not follow the standard format;
        - see Bibliography reference to "Undocumented DOS"

INT 15,85 - System Request Key Pressed

```
        AH = 85h
        AL = 00 key pressed
           = 01 key released


        on return:
        CF = 0 if successful
           = 1 if error
        AH = 80h for PC or PCjr
           = 86h for XT (BIOS after 11/8/82)


        - called by BIOS when the System Request key is pressed/
released
        - available on machines with newer BIOS and keyboards
```

FLAGS - Intel 8086 Family Flags Register



- see PUSHF POPF STI CLI STD CLD

PUSHF/PUSHFD - Push Flags onto Stack

Usage: PUSHF
PUSHFD (386+ only)
Modifies flags: None

Transfers the Flags Register onto the stack. PUSHF saves a 16 bit value while PUSHFD saves a 32 bit value.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	10/14	3	4	4	1
none (PM)	-	-	4	3	1

- see Instruction Timing

POPF/POPFD - Pop Flags off Stack

Usage: POPF
POPFD (386+ only)
Modifies flags: all flags

Pops word/doubleword from stack into the Flags Register and then
increments SP by 2 (for POPF) or 4 (for POPFD).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	8/12	5	5	9	1 (W88=12)
none (PM)	-	-	5	6	1

- see Instruction Timing

MSC: void far _setcliprgn(short x1, short y1, short x2, short y2)

```
        - prototype in graph.h

        - all graphics output to the screen is limited to the
specified
          region;  output to areas other than this area is clipped
        - (x1,y1) upper left corner of clipping region
        - (x1,y1) lower right corner of clipping region
```

Example of variable arguments in C

```
void myprintf( char *format, ... )
   {
   va_list argptr;

   va_start ( argptr, format );
   vsprintf ( text, format, argptr );
   va_end ( argptr );
   write (1, text, strlen (text));
   }
```

CLD - Clear Direction Flag

Usage: CLD
Modifies flags: DF

Clears the Direction Flag causing string instructions to increment
the SI and DI index registers.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	2	1

- see Instruction Timing

AAA - Ascii Adjust for Addition

Usage: AAA
Modifies flags: AF CF (OF,PF,SF,ZF undefined)

Changes contents of AL to valid unpacked decimal. The high order nibble is zeroed.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	8	3	4	3	1

- see Instruction Timing

AAD - Ascii Adjust for Division

Usage: AAD
Modifies flags: SF ZF PF (AF,CF,OF undefined)

Used before dividing unpacked decimal numbers. Multiplies AH by 10 and the adds result into AL. Sets AH to zero. This instruction is also known to have an undocumented behavior.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	60	14	19	14	2

- see Instruction Timing

AAM - Ascii Adjust for Multiplication

Usage: AAM
Modifies flags: PF SF ZF (AF,CF,OF undefined)

Used after multiplication of two unpacked decimal numbers, this instruction adjusts an unpacked decimal number. The high order nibble of each byte must be zeroed before using this instruction.
This instruction is also known to have an undocumented behavior.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	83	16	17	15	2

- see Instruction Timing

AAS - Ascii Adjust for Subtraction

Usage: AAS
Modifies flags: AF CF (OF,PF,SF,ZF undefined)

Corrects result of a previous unpacked decimal subtraction in AL.

High order nibble is zeroed.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	8	3	4	3	1

- see Instruction Timing

ADC - Add With Carry

Usage: ADC dest,src
Modifies flags: AF CF OF SF PF ZF

Sums two binary operands placing the result in the destination.
If CF is set, a 1 is added to the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	7	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	6	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=23+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

ADD - Arithmetic Addition

Usage: ADD dest,src
Modifies flags: AF CF OF PF SF ZF

Adds "src" to "dest" and replacing the original contents of "dest".
Both operands are binary.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	7	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	6	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=23+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

AND - Logical And

Usage: AND dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands replacing the destination
with the result.

Operands	Clocks 808x	286	386	486	Size Bytes
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	7	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	6	1	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=23+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

ARPL - Adjusted Requested Privilege Level of Selector

(286+ protected mode)

```
        Usage:  ARPL    dest,src
        Modifies flags: ZF


        Compares the RPL bits of "dest" against "src".  If the RPL
bits
        of "dest" are less than "src", the destination RPL bits are
set
        equal to the source RPL bits and the Zero Flag is set.
Otherwise
        the Zero Flag is cleared.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	10	20	9	2
mem,reg	-	11	21	9	4

- see Instruction Timing

Macro Assembler Directives

Processor Code Generation Directives

```
        .186    enables assembly of 80186 instructions
        .286    enables assembly of non privileged 80286
instructions
        .286C   same as .286
        .286P   enables assembly of all 80286 instructions
        .287    enabled assembly of 80287 instructions
        .386    enabled assembly of non privileged 80386
instructions
                If used before .MODEL segments are defined as
32bits.
                Causes all segments to default to DWORD alignment.
        .386P   enabled assembly of all 80386 instructions
(see .386)
        .387    enabled assembly of 80387 instructions
        .8086   default, enables assembly of 8088/8086 instruction
        .8087   default, enables assembly of 8087 instructions


        These directives must precede the segment they are to
effect.
        they cannot occur within a segment.
```

Memory Model Directives

```
        .CODE  [name]   starts code segment; must follow .MODEL
directive
        .CONST          starts a constant data segment with name
CONST;
                        must follow .MODEL directive; placed in
DGROUP
        .DATA           starts a near data segment for initialized
data
```

```
                        with name _DATA; must follow .MODEL
directive;
                        placed in DGROUP
        .DATA?          starts a near data segment for uninitialized
                        data with name _BSS; must follow .MODEL
                        directive; placed in DGROUP
        .FARDATA [name] not placed in any group
        .FARDATA? [name] not placed in any group
        .MODEL model    defines memory model to be one of the
following:
                        SMALL, COMPACT, MEDIUM, LARGE or HUGE;
must be
                        used prior to any other segment directive
        .STACK [size]   indicates start of stack segment named
'STACK'
                        with size indicating number of bytes to
reserve,
                        default is 1k; placed in DGROUP
```

Segment Definition, Segment Ordering and Linkage Directives

```
        .ALPHA          orders segments alphabetically
        .SEQ            orders segments sequentially (default)
       ASSUME sreg:name [,sreg:name...]   selects default segment
                        register to be used by the assembler, not
the CPU,
                        for addressing all symbols in the segment
or group.
                        Name must be associated with a SEGMENT or
GROUP
                        or set to "NOTHING" to indicate no segment
register
                        is to be associated.
        COMM def [,def...]  defines variables that are both public
and
                        external (communal).  Can be used in and
include
                        file to identify it to each source file
without
                        declaring it in each model as extern.
```

```
Actually
                        defines data once.  Communal variables
cannot be
                        initialized, and are not guaranteed to be
allocated
                        contiguously since these are allocated by
the linker.
        DOSSEG          orders segments the same as DOS.  This is
Microsoft
                        languages default order; causes paragph
alignment
        END [name]      marks end of source module and sets program
                        start address (CS:IP) if 'name' is present
   name ENDP            ends procedure 'name'
   name ENDS            ends a segment or structure
        EXTRN name:type [,name:type...]  defines one or more
external symbols
   name GROUP seg[,seg]
   name LABEL [NEAR|FAR|PROC]  defines an entry point;  If PROC is
specified,
                        it's value depends on the current MODEL
        NAME pgmName    ignored since MASM 5.0; used to set module
name
   name PROC [NEAR|FAR] defines procedure; NEAR/FAR has .MODEL
default
        PUBLIC name[,name...]  makes symbol 'name' available to
other modules
   name SEGMENT [align][combine][use]['class']
                align   = BYTE  align on byte address (no alignment)
                        = WORD  align on even address
                        = DWORD align on DWORD address
                        = PARA  align on next 16 byte paragraph
                        = PAGE  align on next 256 byte boundary
                combine = PUBLIC  similar named segments are
concatenated (CS)
                        = STACK   similar named segments are
concatenated (SS)
                        = COMMON  similar named segment are
overlapped
                        = MEMORY  similar names segments are
concatenated
                        = AT addr segment relative to absolute
address
                        = nothing segment is private and loaded
independent
                use     = USE16   segments will be 16 bits (if .386)
```

```
                        = USE32   segments will be 32 bits (if .386)
```

Data Allocation Directives

```
        ALIGN n         aligns next variable or instruction on a
boundary
                        that is a multiple of "n".  This can speed
memory
                        fetches on 16 and 32 bit CPU'S if aligned.
New to
                        MASM 5.0, previous versions used EVEN.  Can
result
                        in NOP's added to code.
 [name] DB init[,init...]  define byte
 [name] DD init[,init...]  define double word (DWORD, 4 bytes)
 [name] DF init[,init...]  define far word (FWORD, 386, 6 bytes)
 [name] DQ init[,init...]  define quad word (QWORD, 8 bytes)
 [name] DT init[,init...]  define temp word (TBYTE, 10 bytes)
 [name] DW init[,init...]  define word (WORD, 2 bytes)
  count DUP (init[,init...]) duplicate 'init' 'count' times; DUP
can be
                        nested to 17 levels; DUP'ed initial values
                        of (?) don't result in data in the object
file
                        but instead increment the next data addr
   name ENDS            end of structure or segment
        EVEN            same as align 2;  Aligns data on even
boundary
        ORG expr        sets location counter to 'expr';  If 'expr'
                        is '$' the code is ORG'ed at the current
loc.
   name RECORD fld[,fld...]  defines a byte or word variable
                        consisting of bit fields;  fields have the
format:
                        fieldname:width[=expr];  the sum of all
widths
                        must be <= 0
 [name] STRUC <[init[,init]]>  defines beginning of a structure;
Values
                        between <> are initializers;  The '<>'
symbols
                        are required.
```

Logical and Bit Oriented Directives

```
  expr1 AND  expr2      returns nonzero if any set bit matches
  expr1 EQ   expr2      returns (-1) for true or (0) for false
  expr1 GE   expr2      returns (-1) for true or (0) for false
  expr1 LE   expr2      returns (-1) for true or (0) for false
  expr1 LT   expr2      returns (-1) for true or (0) for false
        MASK {fldname|record}  returns bit mask for bits in record
  expr1 OR   expr2      returns bitwise OR on expr1 and expr2
        NOT  expr       returns 'expr' with all bits reversed
   expr SHL  count      returns expr shifted left count times
   expr SHR  count      returns expr shifted right count times
        WIDTH {fldname|record}  returns width of field in bit record
  expr1 XOR expr2       returns bitwise XOR on expr1 and expr2
```

Other Operators and Directives

```
        []              index operator, same as addition
        .MSFLOAT        encode floats in Microsoft Real Format
        .TYPE   expr    returns byte defining mode and scope of expr
   name EQU     expr    assigns expression to name. surround text
with <>
        HIGH    expr    returns high byte of 'expr'
        INCLUDE filespec  inserts code from 'filespec' into file
        INCLUDELIB filespec  stores link library info in .OBJ file
        LENGTH  var     returns number of data objects in DUPed
'var'
        LOW     expr    returns low byte of 'expr'
  expr1 MOD     expr2   return remainder of expr1/expr2
        OFFSET  expr    returns offset of expr;   When .MODEL is
used
                        the offset of a group relative segment
refers
                        to the end of the segment
   type PTR     expr    forces 'expr' to 'type'
        SEG     expr    returns segment of expression
        SHORT           sets type of label to short, less than 128
```

```
                        bytes from start of next instruction
        SIZE    var     returns # of bytes allocated by DUP
directive
        THIS    type    returns an operand of specified type whose
                        offset and segment values are equal to the
                        current location
        TYPE    expr    returns type of expression
```

Program Listing and Documentation Directives

```
        .CREF           restores listing of cross reference symbols
        .LALL           include macro expansion in listings
        .LFCOND         include false conditional blocks in listings
        .LIST           starts listing of statements
        .SALL           suppress listing of all macro expansions
        .SFCOND         suppress false conditional blocks in
listings
        .XALL           start listing of macro expansion
        .XCREF [name[,name...]]  suppress symbols in cross reference
        .XLIST          suppress program listing
        COMMENT delimiter [text]
        PAGE   [[len],wid]      sets page length&width or ejects if
no parms
        SUBTTL  text    defines program listing subtitle
        TITLE   text    defines program listing title
```

Condition Assembly Directives

```
        ELSE            else clause for conditional assembly block
        ENDIF           terminates a conditional assembly block
        IFDEF   name    conditional assembly if name is defined
```

Macro Definition Directives

```
        ENDM            terminates a macro block
        EXITM           exit macro expansion immediately
        IRP  parm, parm in the statements enclosed by the
                        IRP and ENDM will be repeated and replaced
with the
                        values of "arg" for each "arg" in the <>.
        IRPC parm,  parm in the statements enclosed by the IRPC
                        and ENDM will be repeated and replaced with
the values
```

```
                        of each char in the "string" for each
character
                        position in the string.  "string" should be
enclosed
                        in <> if it contains spaces or other
separators.
        LOCAL name[,name...]  defines scope symbol as local to a
macro
   name MACRO [parm[,parm...]]  defines a macro and it's parameters
        PURGE name[,name]  purges macros from memory
        REPT    expr    repeats all statements through ENDM
statement for
                        'expr' times
```

User Message Directives

```
        .ERR            generates and error
        .ERR1           generates an error on PASS 1
        .ERR2           generates an error on PASS 2
        .ERRB           generates an error if 'arg' is blank
        .ERRDEF name    generates an error if 'name' is previously
defined
        .ERRDIF[I] ,
        .ERRE   expr    generates and error is 'expr' is false
        %OUT    text    displays 'text' to console
```

Predefined Equates (available only if simplified segments are used)

```
        @curseg         contains the current segment
        @filename       current file name without extension
        @code           contains the current code segment
        @codesize       0 for small & compact, 1 for large, medium
& huge
        @datasize       0 for small & medium, 1 for compact &
large, 2=huge
        @const          contains segment of define by .CONST
        @data           contains segment of define by .DATA
        @data?          contains segment of define by .DATA?
        @fardata        contains segment of define by .FARDATA
        @fardata?       contains segment of define by .FARDATA?
        @stack          contains segment of define by .STACK
```

Most of these are only available if the simplified segment

system
is used. @curseg and @filename are available regardless.

Radix Specifiers

.RADIX expr sets radix [2..16] for numbers (dec. default)
B binary data specifier
Q octal data specifier
O octal data specifier
D decimal data specifier
H hexadecimal data specifier

Microsoft Assembler Command Line Options

MASM [options] srcfile[,[objfile][,[lstfile][,[xreffile]]]][;]

Options	Definition
/A	generate segments in alphabetical order
/B[size]	sets I/O buffer size in K bytes (1..63, default 32)
/C	generate cross reference file with .CRF extension
/D	generate PASS 1 listing
/Dsym[=val]	define symbol for use during assembly
/E	emulate floating point instructions (for use with HLL)
/H	list options and command syntax
/Ipath	include-file search path
/L	generate listing file with .LST extension
/ML	case sensitive for all symbols
/MU	upper case all symbols (default)
/MX	case sensitive in external and public symbols
/N	suppress symbol tables in listings
/P	check for impure code in 286 and 386 protected mode (invalid CS overrides)
/S	generate segments in the order they are found (default)
/T	terse message display; display errors only
/V	verbose message display; includes # lines and symbols
/W{0|1|2}	assembly warning level 0 = no warnings 1 = severe warnings only 2 = all warnings enabled
/X	display complete conditional assembly blocks in listing including false conditionals
/Z	display errors including line numbers to

```
screen
        /ZD             generate line numbers in .OBJ files
        /ZI             generate both symbolic and line number
information in
                        .OBJ files
```

Environment Variables

```
        INCLUDE         search path for include files
        MASM            default command line options
```

BOUND - Array Index Bound Check (80188+)

```
Usage:  BOUND   src,limit
Modifies flags: None
```

Array index in source register is checked against upper and lower bounds in memory source. The first word located at "limit" is the lower boundary and the word at "limit+2" is the upper array bound. Interrupt 5 occurs if the source value is less than or higher than the source.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,mem32	-	nj=13	nj=10	7	2
reg32,mem64	-	nj=13	nj=10	7	2

- nj = no jump taken
- see Instruction Timing

BSF - Bit Scan Forward (386+ only)

Usage: BSF dest,src
Modifies flags: ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	-	10+3n	6-42	3
reg,mem	-	-	10+3n	7-43	3-7
reg32,reg32	-	-	10+3n	6-42	3-7
reg32,mem32	-	-	10+3n	7-43	3-7

- see Instruction Timing

BSR - Bit Scan Reverse (386+ only)

```
Usage:  BSR     dest,src
Modifies flags: ZF
```

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	-	10+3n	6-103	3
reg,mem	-	-	10+3n	7-104	3-7
reg32,reg32	-	-	10+3n	6-103	3-7
reg32,mem32	-	-	10+3n	7-104	3-7

- see Instruction Timing

BSWAP - Byte Swap (486+ only)

Usage: BSWAP reg32
Modifies flags: none

Changes the byte order of a 32 bit register from big endian to little endian or vice versa. Result left in destination register is undefined if the operand is a 16 bit register.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg32	-	-	-	1	2

- see Instruction Timing

BT - Bit Test (386+ only)

Usage: BT dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the
Carry Flag.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,immed8	-	-	3	3	4-8
mem16,immed8	-	-	6	6	4-8
reg16,reg16	-	-	3	3	3-7
mem16,reg16	-	-	12	12	3-7

- see Instruction Timing

BTC - Bit Test with Compliment (386+ only)

Usage: BTC dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the
Carry Flag after being complimented (inverted).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

- see Instruction Timing

BTR - Bit Test with Reset (386+ only)

Usage: BTR dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the
Carry Flag and then cleared in the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

- see Instruction Timing

BTS - Bit Test and Set (386+ only)

Usage: BTS dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the
Carry Flag and then set in the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

- see Instruction Timing

CALL - Procedure Call

Usage: CALL destination
Modifies flags: None

Pushes Instruction Pointer (and Code Segment for far calls) onto
stack and loads Instruction Pointer with the address of proc-name.
Code continues with execution at CS:IP.

Operands	808x	286	386	486
			Clocks	
rel16 (near, IP relative)	19	7	7 +m	3
rel32 (near, IP relative)	-	-	7 +m	3
reg16 (near, register indirect)	16	7	7 +m	5
reg32 (near, register indirect)	-	-	7 +m	5
mem16 (near, memory indirect)	-	21+EA	11	10+m 5
mem32 (near, memory indirect)	-	-	10 +m	5
ptr16:16 (far, full ptr supplied)	28	13	17 +m	18
ptr16:32 (far, full ptr supplied)	-	-	17 +m	18
ptr16:16 (far, ptr supplied, prot. mode)	-	26	34 +m	20
ptr16:32 (far, ptr supplied, prot. mode)	-	-	34 +m	20
m16:16 (far, indirect)	37+EA	16	22 +m	17
m16:32 (far, indirect)	-	-	22 +m	17
m16:16 (far, indirect, prot. mode)	-	29	38	

+m 20				
	m16:32 (far, indirect, prot. mode)	-	-	38
+m 20				
	ptr16:16 (task, via TSS or task gate)	-	177	
TS 37+TS				
	m16:16 (task, via TSS or task gate)	-	180/185	5
+TS 37+TS				
	m16:32 (task)	-	-	
TS 37+TS				
	m16:32 (task)	-	-	5
+TS 37+TS				
	ptr16:16 (gate, same privilege)	-	41	52
+m 35				
	ptr16:32 (gate, same privilege)	-	-	52
+m 35				
	m16:16 (gate, same privilege)	-	44	56
+m 35				
	m16:32 (gate, same privilege)	-	-	56
+m 35				
	ptr16:16 (gate, more priv, no parm)	-	82	86
+m 69				
	ptr16:32 (gate, more priv, no parm)	-	-	86
+m 69				
	m16:16 (gate, more priv, no parm)	-	83	90
+m 69				
	m16:32 (gate, more priv, no parm)	-	-	90
+m 69				
	ptr16:16 (gate, more priv, x parms)	-	86+4x	94+4x
+m 77+4x				
	ptr16:32 (gate, more priv, x parms)	-	-	94+4x
+m 77+4x				
	m16:16 (gate, more priv, x parms)	-	90+4x	98+4x
+m 77+4x				
	m16:32 (gate, more priv, x parms)	-	-	98+4x
+m 77+4x				

- see Instruction Timing

CBW - Convert Byte to Word

Usage: CBW
Modifies flags: None

Converts byte in AL to word Value in AX by extending sign of AL
throughout register AH.

Operands	Clocks 808x	286	386	486	Size Bytes
none	2	2	3	3	1

- see Instruction Timing

CDQ - Convert Double to Quad (386+ only)

Usage: CDQ
Modifies flags: None

Converts signed DWORD in EAX to a signed quad word in EDX: EAX by extending the high order bit of EAX throughout EDX

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	-	2	3	1

- see Instruction Timing

CLC - Clear Carry

Usage: CLC
Modifies flags: CF

Clears the Carry Flag.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	2	1

- see Instruction Timing

CLI - Clear Interrupt Flag (disable)

Usage: CLI
Modifies flags: IF

Disables the maskable hardware interrupts by clearing the Interrupt
flag. NMI's and software interrupts are not inhibited.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	3	5	1

- see Instruction Timing

CLTS - Clear Task Switched Flag (286+ privileged)

Usage: CLTS
Modifies flags: None

Clears the Task Switched Flag in the Machine Status Register. This is a privileged operation and is generally used only by operating system code.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	2	5	7	2

- see Instruction Timing

CMC - Complement Carry Flag

```
Usage:  CMC
Modifies flags: CF


Toggles (inverts) the Carry Flag
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	2	1

- see Instruction Timing

CMP - Compare

Usage: CMP dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts source from destination and updates the flags but does not save result. Flags can subsequently be checked for conditions.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	9+EA	7	5	2	2-4 (W88=13+EA)
reg,mem	9+EA	6	6	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	10+EA	6	5	2	3-6 (W88=14+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

CMPS - Compare String (Byte, Word or Doubleword)

```
        Usage:  CMPS    dest,src
                CMPSB
                CMPSW
                CMPSD   (386+ only)
        Modifies flags: AF CF OF PF SF ZF

        Subtracts destination value from source without saving
results.
        Updates flags based on the subtraction and  the index
registers
        (E)SI and (E)DI are incremented or decremented depending on
the
        state of the Direction Flag.  CMPSB inc/decrements the index
        registers by 1, CMPSW inc/decrements by 2, while CMPSD
increments
        or decrements by 4.  The REP prefixes can be used to process
        entire data items.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
dest,src	22	8	10	8	1 (W88=30)

- see Instruction Timing

CMPXCHG - Compare and Exchange

Usage: CMPXCHG dest,src (486+)
Modifies flags: AF CF OF PF SF ZF

Compares the accumulator (8-32 bits) with "dest". If equal the "dest" is loaded with "src", otherwise the accumulator is loaded with "dest".

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	-	-	6	2
mem,reg	-	-	-	7	2

- add 3 clocks if the "mem,reg" comparison fails
- see Instruction Timing

CWD - Convert Word to Doubleword

Usage: CWD
Modifies flags: None

Extends sign of word in register AX throughout register DX forming
a doubleword quantity in DX:AX.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	5	2	2	3	1

- see Instruction Timing

CWDE - Convert Word to Extended Doubleword (386+ only)

Usage: CWDE
Modifies flags: None

Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	-	3	3	1

- see Instruction Timing

DAA - Decimal Adjust for Addition

Usage: DAA
Modifies flags: AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD addition operation.
Contents of AL are changed to a pair of packed decimal digits.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	4	3	4	2	1

- see Instruction Timing

DAS - Decimal Adjust for Subtraction

Usage: DAS
Modifies flags: AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD subtraction operation.
Contents of AL are changed to a pair of packed decimal digits.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	4	3	4	2	1

- see Instruction Timing

DEC - Decrement

```
Usage:  DEC     dest
Modifies flags: AF OF PF SF ZF
```

Unsigned binary subtraction of one from the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	3	2	2	1	2
mem	15+EA	7	6	3	2-4
reg16/32	3	2	2	1	1

- see Instruction Timing

DIV - Divide

Usage: DIV src
Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

Unsigned binary division of accumulator by source. If the source divisor is a byte value then AX is divided by "src" and the quotient is placed in AL and the remainder in AH. If source operand is a word value, then DX:AX is divided by "src" and the quotient is stored in AX and the remainder in DX.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	80-90	14	14	16	2
reg16	144-162	22	22	24	2
reg32	-	-	38	40	2
mem8	(86-96)+EA	17	17	16	2-4
mem16	(150-168)+EA	25	25	24	2-4 (W88=158-176+EA)
mem32	-	-	41	40	2-4

- see Instruction Timing

ENTER - Make Stack Frame (80188+)

Usage: ENTER locals,level
Modifies flags: None

Modifies stack for entry to procedure for high level language.
Operand "locals" specifies the amount of storage to be allocated
on the stack. "Level" specifies the nesting level of the routine.
Paired with the LEAVE instruction, this is an efficient method of
entry and exit to procedures.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
immed16,0	-	11	10	14	4
immed16,1	-	15	12	17	4
immed16,immed8	-	12+4(n-1)	15+4(n-1)	17+3n	4

- see Instruction Timing

LEAVE - Restore Stack for Procedure Exit (80188+)

Usage: LEAVE
Modifies flags: None

Releases the local variables created by the previous ENTER instruction by restoring SP and BP to their condition before the procedure stack frame was initialized.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	5	4	5	1

- see Instruction Timing

ESC - Escape

Usage: ESC immed,src
Modifies flags: None

Provides access to the data bus for other resident processors.
The CPU treats it as a NOP but places memory operand on bus.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
immed,reg	2	9-20	?		2
immed,mem	2	9-20	?		2-4

- see Instruction Timing

NOP - No Operation (90h)

Usage: NOP
Modifies flags: None

This is a do nothing instruction. It results in occupation of both
space and time and is most useful for patching code segments.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	3	3	3	1	1

- see Instruction Timing

WAIT/FWAIT - Event Wait

Usage: WAIT
FWAIT
Modifies flags: None

CPU enters wait state until the coprocessor signals it has finished
it's operation. This instruction is used to prevent the CPU from
accessing memory that may be temporarily in use by the coprocessor.
WAIT and FWAIT are identical.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	4	3	6+	1-3	1

- see Instruction Timing

HLT - Halt CPU

Usage: HLT
Modifies flags: None

Halts CPU until RESET line is activated, NMI or maskable interrupt received. The CPU becomes dormant but retains the current CS:IP for later restart.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	5	4	1

- see Instruction Timing

IDIV - Signed Integer Division

Usage: IDIV src
Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

Signed binary division of accumulator by source. If source is a byte value, AX is divided by "src" and the quotient is stored in AL and the remainder in AH. If source is a word value, DX:AX is divided by "src", and the quotient is stored in AL and the remainder in DX.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	101-112	17	19	19	2
reg16	165-184	25	27	27	2
reg32	-	-	43	43	2
mem8	(107-118)+EA	20	22	20	2-4
mem16	(171-190)+EA	38	30	28	2-4 (W88=175-194)
mem32	-	-	46	44	2-4

- see Instruction Timing

IMUL - Signed Multiply

```
Usage:  IMUL    src
        IMUL    src,immed  (286+ only)
        IMUL    dest,src,immed8  (286+ only)
        IMUL    dest,src  (386+ only)
Modifies flags: CF OF (AF,PF,SF,ZF undefined)
```

Signed multiplication of accumulator by "src" with result placed in the accumulator. If the source operand is a byte value, it is multiplied by AL and the result stored in AX. If the source operand is a word value it is multiplied by AX and the result is stored in DX:AX. Other variations of this instruction allow specification of source and destination registers as well as a third immediate factor.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	80-98	13	9-14	13-18	2
reg16	128-154	21	9-22	13-26	2
reg32	-	-	9-38	12-42	2
mem8	86-104	16	12-17	13-18	2-4
mem16	134-160	24	12-25	13-26	2-4
mem32	-	-	12-41	13-42	2-4
reg16,reg16	-	-	9-22	13-26	3-5
reg32,reg32	-	-	9-38	13-42	3-5
reg16,mem16	-	-	12-25	13-26	3-5
reg32,mem32	-	-	12-41	13-42	3-5
reg16,immed	-	21	9-22	13-26	3
reg32,immed	-	21	9-38	13-42	3-6
reg16,reg16,immed	-	2	9-22	13-26	3-6
reg32,reg32,immed	-	21	9-38	13-42	3-6
reg16,mem16,immed	-	24	12-25	13-26	3-6
reg32,mem32,immed	-	24	12-41	13-42	3-6

- see Instruction Timing

IN - Input Byte or Word From Port

Usage: IN accum,port
Modifies flags: None

A byte, word or dword is read from "port" and placed in AL, AX or EAX respectively. If the port number is in the range of 0-255 it can be specified as an immediate, otherwise the port number must be specified in DX. Valid port ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS/2's.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
accum,immed8	10/14	5	12	14	2
accum,immed8 (PM)			6/26	8/28/27	2
accum,DX	8/12	5	13	14	1
accum,DX (PM)			7/27	8/28/27	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing if: CPL ó IOPL
  second number is the timing if: CPL > IOPL or in VM 86 mode (386)
  CPL ò IOPL (486)
  third number is the timing when: virtual mode on 486 processor

- 486 virtual mode always requires 27 cycles

- see Instruction Timing

INC - Increment

Usage: INC dest
Modifies flags: AF OF PF SF ZF

Adds one to destination unsigned binary operand.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	3	2	2	1	2
reg16	3	2	2	1	1
reg32	3	2	2	1	1
mem	15+EA	7	6	3	2-4 (W88=23+EA)

- see Instruction Timing

INS - Input String from Port (80188+)

```
        Usage:  INS     dest,port
                INSB
                INSW
                INSD  (386+ only)
        Modifies flags: None

        Loads data from port to the destination ES:(E)DI  (even if a
        destination operand is supplied).  (E)DI is adjusted by the
size
        of the operand and increased if the Direction Flag is
cleared and
        decreased if the Direction Flag is set.  For INSB, INSW,
INSD no
        operands are allowed and the size is determined by the
mnemonic.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
dest,port	-	5	15	17	1
dest,port (PM)	-	5	9/29	10/32/30	1
none	-	5	15	17	1
none (PM)	-	5	9/29	10/32/30	1

```
        - 386+ protected mode timings depend on privilege levels.

          first number is the timing if:    CPL ó IOPL
          second number is the timing if:   CPL > IOPL
          third number is the timing if:    virtual mode on 486
processor
```

- see Instruction Timing

INT - Interrupt

Usage: INT num
Modifies flags: TF IF

Initiates a software interrupt by pushing the flags, clearing the
Trap and Interrupt Flags, pushing CS followed by IP and loading
CS:IP with the value found in the interrupt vector table. Execution
then begins at the location addressed by the new CS:IP

Operands	Clocks 808x	286	386	486	Size Bytes
3 (constant)	52/72	23+m	33	26	2
3 (prot. mode, same priv.)	-	40+m	59	44	2
3 (prot. mode, more priv.)	-	78+m	99	71	2
3 (from VM86 to PL 0)	-	-	119	82	2
3 (prot. mode via task gate)	-	167+m	TS	37+TS	2
immed8	51/71	23+m	37	30	1
immed8 (prot. mode, same priv.)	-	40+m	59	44	1
immed8 (prot. mode, more priv.)	-	78+m	99	71	1
immed8 (from VM86 to PL 0)	-	-	119	86	1
immed8 (prot. mode, via task gate)	-	167+m	TS	37+TS	1

- see Instruction Timing

INTO - Interrupt on Overflow

Usage: INTO
Modifies flags: IF TF

If the Overflow Flag is set this instruction generates an INT 4
which causes the code addressed by 0000:0010 to be executed.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none: jump	53/73	24+m	35	28	1
no jump	4	3	3	3	
(prot. mode, same priv.)	-	-	59	46	1
(prot. mode, more priv.)	-	-	99	73	1
(from VM86 to PL 0)	-	-	119	84	1
(prot. mode, via task gate)	-	-	TS	39+TS	1

- see Instruction Timing

INVD - Invalidate Cache (486+ only)

Usage: INVD
Modifies flags: none

Flushes CPU internal cache. Issues special function bus cycle which indicates to flush external caches. Data in write-back external caches is lost.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	-	-	4	2

- see Instruction Timing

INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+ only)

Usage: INVLPG
Modifies flags: none

Invalidates a single page table entry in the Translation Look-Aside Buffer. Intel warns that this instruction may be implemented differently on future processors.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	-	-	12	2

- timing is for TLB entry hit only.
- see Instruction Timing

Jump Instructions Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF=0 and ZF=0
JAE	Jump if Above or Equal	CF=0
JB	Jump if Below	CF=1
JBE	Jump if Below or Equal	CF=1 or ZF=1
JC	Jump if Carry	CF=1
JCXZ	Jump if CX Zero	CX=0
JE	Jump if Equal	ZF=1
JG	Jump if Greater (signed)	ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed)	SF=OF
JL	Jump if Less (signed)	SF != OF
JLE	Jump if Less or Equal (signed)	ZF=1 or SF != OF
JMP	Unconditional Jump	unconditional
JNA	Jump if Not Above	CF=1 or ZF=1
JNAE	Jump if Not Above or Equal	CF=1
JNB	Jump if Not Below	CF=0
JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
JNC	Jump if Not Carry	CF=0
JNE	Jump if Not Equal	ZF=0
JNG	Jump if Not Greater (signed)	ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed)	SF != OF
JNL	Jump if Not Less (signed)	SF=OF
JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed)	OF=0
JNP	Jump if No Parity	PF=0
JNS	Jump if Not Signed (signed)	SF=0
JNZ	Jump if Not Zero	ZF=0

JO	Jump if Overflow (signed)	OF=1
JP	Jump if Parity	PF=1
JPE	Jump if Parity Even	PF=1
JPO	Jump if Parity Odd	PF=0
JS	Jump if Signed (signed)	SF=1
JZ	Jump if Zero	ZF=1

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
Jx: jump	16	7+m	7+m	3	2
no jump	4	3	3	1	
Jx near-label	-	-	7+m	3	4
no jump	-	-	3	1	

- It's a good programming practice to organize code so the expected case is executed without a jump since the actual jump takes longer to execute than falling through the test.
- see JCXZ and JMP for their respective timings
- see Instruction Timing FLAGS

JAE/JNB - Jump Above or Equal / Jump on Not Below

```
Usage:  JAE     label
        JNB     label
Modifies flags: None
```

Causes execution to branch to "label" if the Carry Flag is clear.

Functionally similar to JNC. Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JNC - Jump Not Carry

Usage: JNC label
Modifies flags: None

Causes execution to branch to "label" if the Carry Flag is clear.

Functionally similar to JAE or JNB. Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JBE/JNA - Jump Below or Equal / Jump Not Above

```
Usage:  JBE     label
        JNA     label
Modifies flags: None
```

Causes execution to branch to "label" if the Carry Flag or the Zero Flag is set. Unsigned comparision.

Operands		Clocks 808x	286	386	486	Size Bytes
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JCXZ/JECXZ - Jump if Register (E)CX is Zero

Usage: JCXZ label
JECXZ label (386+ only)
Modifies flags: None

Causes execution to branch to "label" if register CX is zero. Uses unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	18	8+m	9+m	8	2
	no jump	6	4	5	5	

- see Instruction Timing

Hewlett Packard Laserjet Printer Control Codes

Printer Control and Orientation

ESC E	Reset printer
ESC z	Self Test
ESC &l0O	Portrait orientation
ESC &l1O	Landscape orientation
ESC (s0P	Select fixed space font
ESC (s1P	Select proportional font
ESC (s0S	Set upright character orientation
ESC (s1S	Set Italic character orientation
ESC &l#X	Select '#' number of copies
ESC &l0H	Eject page
ESC &l1H	Feed paper from tray
ESC &l2H	Feed paper manually
ESC &l3H	Feed envelope
ESC &l0T	Default stacking position
ESC &l1T	Togglestacking position

8 Bit Symbol Set Selection

ESC (8U	Select Roman 8 symbol set
ESC (8K	Select Kana 8 symbol set
ESC (8M	Select Math 8 symbol set

7 Bit Symbol Set Selection

ESC (0U	Select USASCII symbol set
ESC (0B	Select Line Draw symbol set
ESC (0A	Select Math symbol set
ESC (0M	Select Math 7 symbol set
ESC (0Q	Select Math 8a symbol set
ESC (1Q	Select Math 8b symbol set
ESC (1U	Select US Legal symbol set
ESC (0E	Select Roman Extension symbol set
ESC (0D	Select ISO Denmark/Norway symbol set
ESC (1E	Select ISO United Kingdom symbol set
ESC (0F	Select ISO France symbol set
ESC (0G	Select ISO German symbol set
ESC (0I	Select ISO Italy symbol set
ESC (0S	Select ISO Sweden/Finland symbol set
ESC (1S	Select ISO Spain symbol set
ESC (15U	Select PiFont symbol set
ESC (2Q	Select PiFonta symbol set

Font Management

```
ESC (s3T          Select Courier font
ESC (s0T          Select Line Printer font
ESC (s1T          Select Pica font
ESC (s2T          Select Elite font
ESC (s4T          Select Helvetica font
ESC (s5T          Select Times Roman (TMS RMN) font
ESC (s6T          Select Gothic font
ESC (s7T          Select Script font
ESC (s8T          Select Prestige font
ESC *c#D          Specify font ID '#'
ESC *c#E          Specify character code '#'
ESC *c0F          Delete all fonts, including permanent
ESC *c1F          Delete all temporary fonts
ESC *c2F          Delete last font ID specified
ESC *c3F          Delete last character code and font ID
specified
ESC *c4F          Make last font ID temporary
ESC *c5F          Make last font ID permanent
ESC *c6F          Copy or assign last font ID specified
ESC *c7F          Reestablish ROM
ESC *c8F          Set primary font
ESC *c9F          Set secondary font
ESC *c10F         Set primary and secondary font default
ESC )s#W   Create font header
ESC (s#W   Download character
ESC (#X           Designate downloaded font as primary
ESC )#X           Designate downloaded font as secondary
ESC (#@           Primary font default (see printer manual)
ESC )#@           Secondary font default(see printer manual)
```

Pitch and Point Selection

```
ESC (s10H       Set 10 pitch
ESC (s12H       Set 12 pitch
ESC (s16.6H     Set 16.66 pitch

ESC (s7V        Set point size to 7
ESC (s8V        Set point size to 8
ESC (s8.5V      Set point size to 8.5
ESC (s10V       Set point size to 10
ESC (s12V       Set point size to 12
ESC (s14.4V     Set point size to 14.4
```

Page Dimensions

```
ESC &l#P        Set page length to '#' lines
ESC &l#E        Set top margin to '#' lines
```

```
        ESC &l#F        Set text length to '#' lines
        ESC 9           Clear margins
        ESC &a#L        Set left margin to column '#'
        ESC &a#M        Set right margin to column '#'
        ESC &l#C        Set vertical motion index to '#' 1/48"
increments
        ESC &l#D        Set lines per inch to '#', valid values are:
                        1, 2, 3, 4, 6, 8, 12, 16 or 24
        ESC &k#H        Set horizontal motion index where # is
derived
                        using   # = (120.0 / cpi)     (1/10
precision)
```

Cursor Positioning

```
        ESC &a#R        Move to row '#'
        ESC &a#C        Move to col '#'
        ESC &a#H        Move to horizontal position '#' in
decipoints
        ESC &a#V        Move to vertical position '#' in decipoints
        ESC *p#X        Move to horizontal position '#' in dots
        ESC *p#Y        Move to vertical position '#' in dots
        ESC &f0S        Push cursor position
        ESC &f1         Pop cursor position
```

Raster Graphics

```
        ESC *t75R         Select 75 dots per inch graphics mode
        ESC *t100R        Select 100 dots per inch graphics mode
        ESC *t150R        Select 150 dots per inch graphics mode
        ESC *t300R        Select 300 dots per inch graphics mode
        ESC *r0A          Start graphics at left most position
        ESC *r1A          Start graphics at current cursor
        ESC *b#W    Transfer '#' byte raster image as stream ""
        ESC *rB           End graphics
```

Advanced Graphics

```
        ESC *c#A          Set horizontal rule/pattern size in dots
        ESC *c#H          Set horizontal rule/pattern size in
decipoints
        ESC *c#B          Set vertical rule/pattern size in dots
        ESC *c#V          Set vertical rule/pattern size in
decipoints
        ESC *c0P          Select black rule
        ESC *c2P          Select gray scale pattern
        ESC *c3P          Select HP-Defined pattern
        ESC *c#G          Set grey scale pattern, where # is a
value between
```

```
                        [0..6] for HP defined patterns and
[0..100] to
                        specify percentage gray scaling.  The
mode depends
                        on the rule/pattern selected using ESC *c?
P
        ESC *c1G        Vertical lines pattern
        ESC *c2G        Horizontal lines pattern
        ESC *c3G        Diagonal lines pattern (upward left to
right)
        ESC *c4G        Diagonal lines pattern (downward left to
right)
        ESC *c5G        Horizontal/vertical grid lines pattern
        ESC *c6G        Diagonal grid pattern
        ESC *c#G        Set gray scaling to '#' percent
```

Macro commands

```
        ESC &f#Y        Identify macro as ID "#'
        ESC &f0X        Start macro definition
        ESC &f1X        Stop macro definition
        ESC &f2X        Execute macro
        ESC &f3X        Call macro
        ESC &f4X        Enable auto macro overlay
        ESC &f5X        Disable auto macro overlay
        ESC &f6X        Delete all macros
        ESC &f7X        Delete all temporary macros
        ESC &f8X        Delete macro ID
        ESC &f9X        Make macro temporary
        ESC &f10X       Make macro permanent
```

Miscellaneous

```
        ESC (s#B        Set stroke weight '#'=(7..-7), 7=bold, -
7=light
        ESC &dD         Set underline on
        ESC &d@         Set underline off
        ESC =           Half line feed
        ESC Y           Turn display functions mode on
        ESC Z           Turn display functions mode off (default)
        ESC &p#X    Disable command interpretation for the '#' bytes
                        following this command
        ESC &l0L        Disable perforation skip
        ESC &l1L        Enable perforation skip
        ESC &k0G        Set line terminators to CR=CR, LF=LF,
FF=FF
        ESC &k1G        Set line terminators to CR=CR+LF, LF=LF,
FF=FF
```

```
        ESC &k2G          Set line terminators to CR=CR, LF=CR+LF,
FF=CR+FF
        ESC &k3G          Set line terminators to CR=CR+LF, LF=CR
+LF, FF=CR+FF
        ESC &s0C          Enable end of line wrap
        ESC &s1C          Disable end of line wrap
```

Escape sequence combination rules:

1. The first 2 characters following the ESC must be the same.
2. The final character in any sequence other than the last must be changed to lower case.
3. The last character in the complete sequence must be changed to upper case.
4. Escape sequences must be specified in the order in which they should be performed.

- the space following ESC is not included in the string

JGE/JNL - Jump Greater or Equal / Jump Not Less

```
Usage:  JGE     label
        JNL     label
Modifies flags: None
```

Causes execution to branch to "label" if the Sign Flag equals the Overflow Flag. Signed comparision.

Operands		Clocks 808x	286	386	486	Size Bytes
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JLE/JNG - Jump Less or Equal / Jump Not Greater

```
Usage:  JLE     label
        JNG     label
Modifies flags: None
```

Causes execution to branch to "label" if the Zero Flag is set or the
Sign Flag is not equal to the Overflow Flag. Signed comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

int setjmp(jmp_buf env)

- prototype in setjmp.h

- captures callers task state in env and returns 0
- see longjmp()

void longjmp(jmp_buf env, int id)

- prototype in setjmp.h

- call to longjmp() with env restores task state (set by setjmp()),
  returning value id
- cannot return 0; if id == 0, returns 1

JB/JNAE - Jump Below / Jump Not Above or Equal

```
Usage:  JB      label
        JNAE    label
Modifies flags: None
```

Causes execution to branch to "label" if the Carry Flag is set.

Functionally similar to JC. Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JA/JNBE - Jump Above / Jump Not Below or Equal

```
Usage:  JA      label
        JNBE    label
Modifies flags: None
```

Causes execution to branch to "label" if the Carry Flag and Zero Flag
are both clear. Unsigned comparision.

Operands		Clocks 808x	286	386	486	Size Bytes
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JNE/JNZ - Jump Not Equal / Jump Not Zero

```
Usage:  JNE     label
        JNZ     label
Modifies flags: None
```

Causes execution to branch to "label" if the Zero Flag is clear.

Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JL/JNGE - Jump Less / Jump Not Greater or Equal

```
Usage:  JL      label
        JNGE    label
Modifies flags: None
```

Causes execution to branch to "label" if the Sign Flag is not equal
to Overflow Flag. Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JG/JNLE - Jump Greater / Jump Not Less or Equal

```
Usage:  JG      label
        JNLE    label
Modifies flags: None
```

Causes execution to branch to "label" if the Zero Flag is clear or
the Sign Flag equals the Overflow Flag. Signed comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JNO - Jump Not Overflow

Usage: JNO label
Modifies flags: None

Causes execution to branch to "label" if the Overflow Flag is clear.
Signed comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JNP/JPO - Jump Not Parity / Jump Parity Odd

```
Usage:  JNP     label
        JPO     label
Modifies flags: None
```

Causes execution to branch to "label" if the Parity Flag is clear.

Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JNS - Jump Not Signed

Usage: JNS label
Modifies flags: None

Causes execution to branch to "label" if the Sign Flag is clear.

Signed comparision.

Operands		Clocks 808x	286	386	486	Size Bytes
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

Joystick / Game Port (201h)

```
|7|6|5|4|3|2|1|0|  Port at 201h used with Joysticks
 | | | | | | | `---- joystick a, x coord (0 = timing active)
 | | | | | | `----- joystick a, y coord (0 = timing active)
 | | | | | `------ joystick b, x coord (0 = timing active)
 | | | | `------- joystick b, y coord (0 = timing active)
 | | | `-------- joystick a, button 1 (0=pressed)
 | | `---------- joystick a, button 2 (0=pressed)
 | `----------- joystick b, button 1 (0=pressed)
 `------------ joystick b, button 2 (0=pressed)

|7|6|5|4|3|2|1|0|  Port at 201h used with Paddles
 | | | | | | | `---- paddle a coord (0 = timing active)
 | | | | | | `----- paddle b coord (0 = timing active)
 | | | | | `------ paddle c coord (0 = timing active)
 | | | | `------- paddle d coord (0 = timing active)
 | | | `-------- paddle a, button (0=pressed)
 | | `---------- paddle b, button (0=pressed)
 | `----------- paddle c, button (0=pressed)
 `------------ paddle d, button (0=pressed)
```

- accessed through port 201h
- bits 3-0 are resistive inputs with the length of the pulse
  determined by 0-100K ohm resistive load. Use this formula:

time = 24.2æ + (0.011æ * resistance)
or
resistance = (time - 24.2) / 0.011

- a read should be immediately preceded by a write (any data)
  to start timing for the resistive values.

JP/JPE - Jump on Parity / Jump on Parity Even

```
Usage:  JP      label
        JPE     label
Modifies flags: None
```

Causes execution to branch to "label" if the Parity Flag is set.

Unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JS - Jump Signed

Usage: JS label
Modifies flags: None

Causes execution to branch to "label" if the Sign Flag is set.
Signed comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JE/JZ - Jump Equal / Jump Zero

```
Usage:  JE      label
        JZ      label
Modifies flags: None
```

Causes execution to branch to "label" if the Zero Flag is set. Uses unsigned comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JC - Jump on Carry

Usage: JC label
Modifies flags: None

Causes execution to branch to "label" if the Carry Flag is set.

Functionally similar to JB and JNAE. Unsigned comparision.

Operands		Clocks 808x	286	386	486	Size Bytes
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

JMP - Unconditional Jump

Usage: JMP target
Modifies flags: None

Unconditionally transfers control to "label". Jumps by default are within -32768 to 32767 bytes from the instruction following the jump. NEAR and SHORT jumps cause the IP to be updated while FAR jumps cause CS and IP to be updated.

	Clocks			
Operands	**808x**	**286**	**386**	**486**
rel8 (relative)	15	7+m	7+m	3
rel16 (relative)	15	7+m	7+m	3
rel32 (relative)	-	-	7+m	3
reg16 (near, register indirect)	11	7+m	7+m	5
reg32 (near, register indirect)	-	-	7+m	5
mem16 (near, mem indirect)	18+EA	11+m	10+m	5
mem32 (near, mem indirect)	24+EA	15+m	10+m	5
ptr16:16 (far, dword immed)	-	-	12+m	17
ptr16:16 (far, PM dword immed)	-	-	27+m	19
ptr16:16 (call gate, same priv.)	-	38+m	45+m	32
ptr16:16 (via TSS)	-	175+m	TS	42 +TS
ptr16:16 (via task gate)	-	180+m	TS	43 +TS
mem16:16 (far, indirect)	-	-	43+m	13

mem16:16 (far, PM indirect)	-	-	31+m	18
mem16:16 (call gate, same priv.)	-	41+m	49+m	31
mem16:16 (via TSS)	-	178+m	5+TS	41+TS
mem16:16 (via task gate)	-	183+m	5+TS	42+TS
ptr16:32 (far, 6 byte immed)	-	-	12+m	13
ptr16:32 (far, PM 6 byte immed)	-	-	27+m	18
ptr16:32 (call gate, same priv.)	-	-	45+m	31
ptr16:32 (via TSS)	-	-	TS	42+TS
ptr16:32 (via task state)	-	-	TS	43+TS
m16:32 (far, address at dword)	-	-	43+m	13
m16:32 (far, address at dword)	-	-	31+m	18
m16:32 (call gate, same priv.)	-	-	49+m	31
m16:32 (via TSS)	-	-	5+TS	41+TS
m16:32 (via task state)	-	-	5+TS	42+TS

- see Instruction Timing

JO - Jump on Overflow

Usage: JO label
Modifies flags: None

Causes execution to branch to "label" if the Overflow Flag is set.
Signed comparision.

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	16	7+m	7+m	3	2-4
	no jump	4	3	3	1	

- see Instruction Timing

LAHF - Load Register AH From Flags

Usage: LAHF
Modifies flags: None

Copies bits 0-7 of the flags register into AH. This includes flags
AF, CF, PF, SF and ZF other bits are undefined.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	4	2	2	3	1

- see Instruction Timing

LAR - Load Access Rights (286+ protected)

```
Usage:  LAR     dest,src
Modifies flags: ZF
```

The high byte of the of the destination register is overwritten by the value of the access rights byte and the low order byte is zeroed depending on the selection in the source operand. The Zero Flag is set if the load operation is successful.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,reg16	-	14	15	11	3
reg32,reg32	-	-	15	11	3
reg16,mem16	-	16	16	11	3-7
reg32,mem32	-	-	16	11	3-7

- see Instruction Timing

LDS - Load Pointer Using DS

Usage: LDS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,mem32	16+EA	7	7	6	2-4
reg,mem (PM)	-	-	22	12	5-7

- see Instruction Timing

LEA - Load Effective Address

Usage: LEA dest,src
Modifies flags: None

Transfers offset address of "src" to the destination register.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,mem	2+EA	3	2	1	2-4

- see Instruction Timing

LES - Load Pointer Using ES

Usage: LES dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,mem	16+EA	7	7	6	2-4 (W88=24+EA)
reg,mem (PM)	-	-	22	12	5-7

- see Instruction Timing

LFS - Load Pointer Using FS (386+ only)

Usage: LFS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and FS. The offset is placed in the destination register and the segment is placed in FS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

- see Instruction Timing

LGDT - Load Global Descriptor Table (286+ privileged)

Usage: LGDT src
Modifies flags: None

Loads a value from an operand into the Global Descriptor Table (GDT) register.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
mem64	-	11	11	11	5

- see Instruction Timing

GDT - Global Descriptor Table

Offset	Size	Description
00	8bytes	dummy, set to 0
08	8bytes	GDT data segment location, set to 0
10	8bytes	source GDT pointer
18	8bytes	target GDT pointer
20	8bytes	pointer to BIOS code segment, set to 0. used by the BIOS to create protected mode code segment
28	8bytes	pointer to BIOS stack segment, set to 0. Used by the BIOS to create protected mode stack segment
30	8bytes	user code segment
38	8bytes	temporary BIOS code segment

Source/Target GDT use the following format (offsets 10h and 18h):

Offset	Size	Description
00	word	Segment limit
02	3bytes	24 bit segment physical address
05	byte	data access rights, set to 93h
06	word	reserved word, must be 0

- see LGDT SGDT

SGDT - Store Global Descriptor Table (286+ privileged)

```
Usage:  SGDT    dest
Modifies flags: none
```

Stores the Global Descriptor Table (GDT) Register into the specified operand.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
mem64	-	11	9	10	5

- see Instruction Timing

LGS - Load Pointer Using GS (386+ only)

```
Usage:  LGS     dest,src
Modifies flags: None
```

Loads 32-bit pointer from memory source to destination register and GS. The offset is placed in the destination register and the segment is placed in GS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

- see Instruction Timing

LIDT - Load Interrupt Descriptor Table (286+ privileged)

Usage: LIDT src
Modifies flags: None

Loads a value from an operand into the Interrupt Descriptor Table
(IDT) register.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
mem64	-	12	11	11	5

- see Instruction Timing

LLDT - Load Local Descriptor Table (286+ privileged)

```
Usage:  LLDT    src
Modifies flags: None
```

Loads a value from an operand into the Local Descriptor Table Register (LDTR).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	17	20	11	3
mem16	-	19	24	11	5

- see Instruction Timing

LMSW - Load Machine Status Word (286+ privileged)

```
Usage:  LMSW    src
Modifies flags: None
```

Loads the Machine Status Word (MSW) from data found at "src"

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	3	10	13	3
mem16	-	6	13	13	5

- see Instruction Timing

SMSW - Store Machine Status Word (286+ privileged)

```
Usage:  SMSW    dest
Modifies flags: none
```

Store Machine Status Word (MSW) into "dest".

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	2	10	2	3
mem16	-	3	3	3	5

- see Instruction Timing

LOCK - Lock Bus

```
Usage:  LOCK
        LOCK: (386+ prefix)
Modifies flags: None
```

This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	0	0	1	1

- see Instruction Timing

XCHG - Exchange

Usage: XCHG dest,src
Modifies flags: None

Exchanges contents of source and destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	4	3	3	3	2
mem,reg	17+EA	5	5	5	2-4 (W88=25+EA)
reg,mem	17+EA	5	5	3	2-4 (W88=25+EA)
accum,reg	3	3	3	3	1
reg,accum	3	3	3	3	1

- see Instruction Timing

int remove(char *filename)

- prototype in stdio.h

- implemented as a macro of function unlink()
- attempts to delete an open file may damage the file system

int unlink(const char *filename)

- prototype in dos.h or io.h

- returns 0 if successful, else -1
- attempts to delete an open file may damage the file system

LODS - Load String (Byte, Word or Double)

```
Usage:  LODS    src
        LODSB
        LODSW
        LODSD  (386+ only)
Modifies flags: None

Transfers string element addressed by DS:SI (even if an operand is
supplied) to the accumulator.   SI is incremented based on the size
of the operand or based on the instruction used.  If the Direction
Flag is set SI is decremented, if the Direction Flag is clear SI
is incremented.  Use with REP prefixes.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
src	12/16	5	5	5	1

- see Instruction Timing

LOOP - Decrement CX and Loop if CX Not Zero

Usage: LOOP label
Modifies flags: None

Decrements CX by 1 and transfers control to "label" if CX is not Zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	18	8+m	11+m	6	2
	no jump	5	4	?	2	

- see Instruction Timing

LOOPE/LOOPZ - Loop While Equal / Loop While Zero

```
        Usage:  LOOPE   label
                LOOPZ   label
        Modifies flags: None

        Decrements CX by 1 (without modifying the flags) and
transfers
        control to "label" if CX != 0 and the Zero Flag is set.  The
        "label" operand must be within -128 or 127 bytes of the
instruction
        following the loop instruction.
```

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	18	8+m	11+m	9	2
	no jump	5	4	?	6	

- see Instruction Timing

LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal

```
        Usage:  LOOPNZ  label
                LOOPNE  label
        Modifies flags: None

        Decrements CX by 1 (without modifying the flags) and
transfers
        control to "label" if CX != 0 and the Zero Flag is clear.
The
        "label" operand must be within -128 or 127 bytes of the
instruction
        following the loop instruction.
```

		Clocks				Size
Operands		**808x**	**286**	**386**	**486**	**Bytes**
label:	jump	19	8+m	11+m	9	2
	no jump	5	4	?	6	

- see Instruction Timing

LSL - Load Segment Limit (286+ protected)

Usage: LSL dest,src
Modifies flags: ZF

Loads the segment limit of a selector into the destination register
if the selector is valid and visible at the current privilege level.
If loading is successful the Zero Flag is set, otherwise it is
cleared.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,reg16	-	14	20/25	10	3
reg32,reg32	-	-	20/25	10	3
reg16,mem16	-	16	21/26	10	5
reg32,mem32	-	-	21/26	10	5

- 386 times are listed "byte granular" / "page granular"
- see Instruction Timing

LSS - Load Pointer Using SS (386+ only)

```
Usage:  LSS     dest,src
Modifies flags: None
```

Loads 32-bit pointer from memory source to destination register and SS. The offset is placed in the destination register and the segment is placed in SS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

- see Instruction Timing

LTR - Load Task Register (286+ privileged)

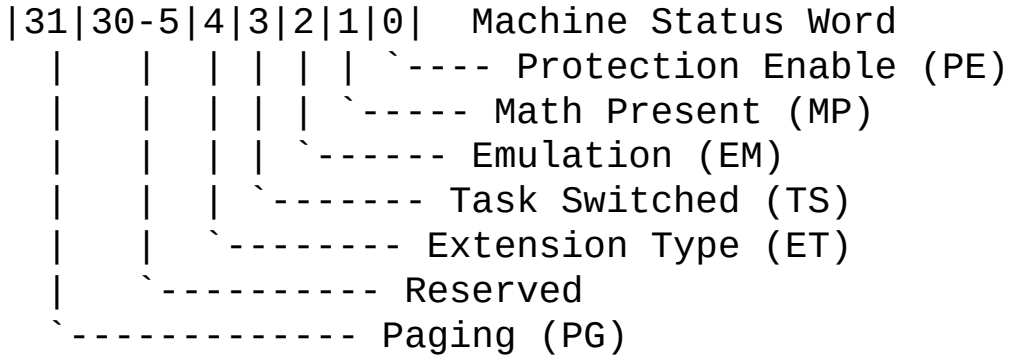
Usage: LTR src
Modifies flags: None

Loads the current task register with the value specified in "src".

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	17	23	20	3
mem16	-	19	27	20	5

- see Instruction Timing

MSW - Machine Status Word (286+ only)



Bit 0	PE	Protection Enable, switches processor between protected and real mode
Bit 1	MP	Math Present, controls function of the WAIT instruction
Bit 2	EM	Emulation, indicates whether coprocessor functions are to be emulated
Bit 3	TS	Task Switched, set and interrogated by coprocessor on task switches and when interpretting coprocessor instructions
Bit 4	ET	Extension Type, indicates type of coprocessor in system
Bits 5-30		Reserved
bit 31	PG	Paging, indicates whether the processor uses page tables to translate linear addresses to physical addresses

- see SMSW LMSW

Memory Model Programming & Segment Information

Model

TINY Data and code fit in one 64K segment. All code and data are accessed via near pointers.

SMALL 64k data segment max and 64k code segment max. All code and data are accessed via near pointers.

COMPACT 1Mb data segment max and 64K code segment max. Code is accessed via near pointers, data is accessed via far pointers. No array can be greater than 64K

MEDIUM 64K data segment max and 1Mb code segment max. Code is accessed via far pointers, data is accessed via near pointers.

LARGE 1Mb data segment max and 1Mb code segment max. All code and data are accessed via far pointers. No single element can be greater than 64K.

HUGE 1Mb data segment max and 1Mb code segment max. All code and data are accessed via far pointers. This is the only model where arrays can be larger than 64K. In this mode is C will normalize all data pointers to avoid segment wrapping.

Small Memory Model

Directive	Segment	Alignment	Combine	Class
.CODE	_TEXT	WORD	PUBLIC	'CODE'
.DATA	_DATA	WORD	PUBLIC	'DATA'
.CONST	CONST	WORD	PUBLIC	'CONST'
.DATA?	_BSS	WORD	PUBLIC	'BSS'
.STACK	STACK	PARA	STACK	'STACK'

Compact Memory Model

Directive	Segment	Alignment	Combine	Class
.CODE	_TEXT	WORD	PUBLIC	'CODE'
.FARDATA	FAR_DATA	PARA	private	'FAR_DATA'
.FARDATA?	FAR_BSS	PARA	private	'FAR_BSS'
.DATA	_DATA	WORD	PUBLIC	'DATA'
.CONST	CONST	WORD	PUBLIC	'CONST'
.DATA?	_BSS	WORD	PUBLIC	'BSS'
.STACK	STACK	PARA	STACK	'STACK'

Medium Memory Model

Directive	Segment	Alignment	Combine	Class
.CODE	name_TEXT	WORD	PUBLIC	'CODE'
.DATA	_DATA	WORD	PUBLIC	'DATA'
.CONST	CONST	WORD	PUBLIC	'CONST'
.DATA?	_BSS	WORD	PUBLIC	'BSS'
.STACK	STACK	PARA	STACK	'STACK'

Large or Huge Memory Models

Directive	Segment	Alignment	Combine	Class
.CODE	name_TEXT	WORD	PUBLIC	'CODE'
.FARDATA	FAR_DATA	PARA	private	'FAR_DATA'
.FARDATA?	FAR_BSS	PARA	private	'FAR_BSS'
.DATA	_DATA	WORD	PUBLIC	'DATA'
.CONST	CONST	WORD	PUBLIC	'CONST'
.DATA?	_BSS	WORD	PUBLIC	'BSS'
.STACK	STACK	PARA	STACK	'STACK'

- all segments fall into DGROUP except for ???_TEXT, FAR_DATA
  and FAR_BSS
- see ASM DIRECTIVES

MOV - Move Byte or Word

Usage: MOV dest,src
Modifies flags: None

Copies byte or word from the source operand to the destination operand. If the destination is SS interrupts are disabled except on early buggy 808x CPUs. Some CPUs disable interrupts if the destination is any of the segment registers

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	2	2	2	1	2
mem,reg	9+EA	3	2	1	2-4 (W88=13+EA)
reg,mem	8+EA	5	4	1	2-4 (W88=12+EA)
mem,immed	10+EA	3	2	1	3-6 (W88=14+EA)
reg,immed	4	2	2	1	2-3
mem,accum	10	3	2	1	3 (W88=14)
accum,mem	10	5	4	1	3 (W88=14)
segreg,reg16	2	2	2	3	2
segreg,mem16	8+EA	5	5	9	2-4 (W88=12+EA)
reg16,segreg	2	2	2	3	2
mem16,segreg	9+EA	3	2	3	2-4 (W88=13+EA)
reg32,CR0/CR2/CR3	-	-	6	4	
CR0,reg32	-	-	10	16	
CR2,reg32	-	-	4	4	3
CR3,reg32	-	-	5	4	3
reg32,DR0/DR1/DR2/DR3	-	22	10		3
reg32,DR6/DR7	-	-	22	10	3
DR0/DR1/DR2/DR3,reg32	-	22	11		3
DR6/DR7,reg32	-	-	16	11	3
reg32,TR6/TR7	-	-	12	4	3

TR6/TR7,reg32	-	-	12	4	3
reg32,TR3				3	
TR3,reg32				6	

- when the 386 special registers are used all operands are 32 bits
- see Instruction Timing

MOVS - Move String (Byte or Word)

```
Usage:  MOVS    dest,src
        MOVSB
        MOVSW
        MOVSD  (386+ only)
Modifies flags: None
```

Copies data from addressed by DS:SI (even if operands are given) to the location ES:DI destination and updates SI and DI based on the size of the operand or instruction used. SI and DI are incremented when the Direction Flag is cleared and decremented when the Direction Flag is Set. Use with REP prefixes.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
dest,src	18	5	7	7	1 (W88=26)

- see Instruction Timing

REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero

```
        Usage:  REPNE
                REPNZ
        Modifies flags: None


        Repeats execution of string instructions while CX != 0 and
the Zero
        Flag is clear.   CX is decremented and the Zero Flag tested
after
        each string operation.   The combination of a repeat prefix
and a
        segment override on processors other than the 386 may
result in
        errors if an interrupt occurs before CX=0.
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2		1

- see Instruction Timing

MOVSX - Move with Sign Extend (386+ only)

Usage: MOVSX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register
with the sign extended.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	-	3	3	3
reg,mem	-	-	6	3	3-7

- see Instruction Timing

MOVZX - Move with Zero Extend (386+ only)

Usage: MOVZX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register
with the zeroes extended.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	-	-	3	3	3
reg,mem	-	-	6	3	3-7

- see Instruction Timing

MUL - Unsigned Multiply

Usage: MUL src
Modifies flags: CF OF (AF,PF,SF,ZF undefined)

Unsigned multiply of the accumulator by the source. If "src" is a byte value, then AL is used as the other multiplicand and the result is placed in AX. If "src" is a word value, then AX is multiplied by "src" and DX:AX receives the result. If "src" is a double word value, then EAX is multiplied by "src" and EDX:EAX receives the result. The 386+ uses an early out algorithm which makes multiplying any size value in EAX as fast as in the 8 or 16 bit registers.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	70-77	13	9-14	13-18	2
reg16	118-113	21	9-22	13-26	2
reg32	-	-	9-38	13-42	2-4
mem8	(76-83)+EA	16	12-17	13-18	2-4
mem16	(124-139)+EA	24	12-25	13-26	2-4
mem32	-	-	12-21	13-42	2-4

- see Instruction Timing IMUL

NEG - Two's Complement Negation

Usage: NEG dest
Modifies flags: AF CF OF PF SF ZF

Subtracts the destination from 0 and saves the 2s complement of
"dest" back into "dest".

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg	3	2	2	1	2
mem	16+EA	7	6	3	2-4 (W88=24+EA)

- see Instruction Timing

NOT - One's Compliment Negation (Logical NOT)

Usage: NOT dest
Modifies flags: None

Inverts the bits of the "dest" operand forming the 1s complement.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg	3	2	2	1	2
mem	16+EA	7	6	3	2-4 (W88=24+EA)

- see Instruction Timing

OR - Inclusive Logical OR

Usage: OR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Logical inclusive OR of the two operands returning the result in the destination. Any bit set in either operand will be set in the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	7	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	6	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem8,immed8	17+EA	7	7	3	3-6
mem16,immed16	25+EA	7	7	3	3-6
accum,immed	4	3	2	1	2-3

- see Instruction Timing

OUT - Output Data to Port

Usage: OUT port,accum
Modifies flags: None

Transfers byte in AL,word in AX or dword in EAX to the specified hardware port address. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
immed8,accum	10/14	3	10	16	2
immed8,accum (PM)	-	-	4/24	11/31/29	2
DX,accum	8/12	3	11	16	1
DX,accum (PM)	-	-	5/25	10/30/29	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL
  third number is the timing when: virtual mode on 486 processor

- see Instruction Timing

OUTS - Output String to Port (80188+ only)

```
Usage:  OUTS    port,src
        OUTSB
        OUTSW
        OUTSD  (386+ only)
Modifies flags: None
```

Transfers a byte, word or doubleword from "src" to the hardware port specified in DX. For instructions with no operands the "src" is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
port,src	-	5	14	17	1
port,src (PM)	-	-	8/28	10/32/30	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL

third number is the timing when: virtual mode on 486 processor

- see Instruction Timing

POP - Pop Word off Stack

Usage: POP dest
Modifies flags: None

Transfers word at the current stack top (SS:SP) to the destination
then increments SP by two to point to the new stack top. CS is not
a valid destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	8	5	4	4	1
reg32	4	-	-	4	1
segreg	8	5	7	3	1
mem16	17+EA	5	5	6	2-4
mem32	5	-	-	6	2-4

- see Instruction Timing

POPA/POPAD - Pop All Registers onto Stack (80188+ only)

```
Usage:  POPA
        POPAD  (386+ only)
Modifies flags: None
```

Pops the top 8 words off the stack into the 8 general purpose 16/32 bit registers. Registers are popped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value popped from the stack is actually discarded.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	19	24	9	1

- see Instruction Timing

PUSH - Push Word onto Stack

```
Usage:  PUSH    src
        PUSH    immed   (80188+ only)
Modifies flags: None
```

Decrements SP by the size of the operand (two or four, byte values are sign extended) and transfers one word from source to the stack top (SS:SP).

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	11/15	3	2	1	1
reg32	-	-	2	1	1
mem16	16+EA	5	5	4	2-4 (W88=24+EA)
mem32	-	-	5	4	2-4
segreg	10/14	3	2	3	1
immed	-	3	2	1	2-3

- see Instruction Timing

PUSHA/PUSHAD - Push All Registers onto Stack (80188+ only)

```
Usage:  PUSHA
        PUSHAD  (386+ only)
Modifies flags: None
```

Pushes all general purpose registers onto the stack in the following order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI. The value of SP is the value before the actual push of SP.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	19	24	11	1

- see Instruction Timing

RCL - Rotate Through Carry Left

Usage: RCL dest,count
Modifies flags: CF OF

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag holds the last bit rotated out.

	Clocks				Size	
Operands	**808x**	**286**	**386**	**486**	**Bytes**	
reg,1	2	2	9	3	2	
mem,1	15+EA	7	10	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	9	8-30	2	
mem,CL	20+EA+4n	8+n	10	9-31	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	9	8-30	3	
mem,immed8	-	8+n	10	9-31	3-5	

- see Instruction Timing

RCR - Rotate Through Carry Right

Usage: RCR dest,count
Modifies flags: CF OF

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag holds the last bit rotated out.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,1	2	2	9	3	2
mem,1	15+EA	7	10	4	2-4 (W88=23+EA)
reg,CL	8+4n	5+n	9	8-30	2
mem,CL	20+EA+4n	8+n	10	9-31	2-4 (W88=28+EA+4n)
reg,immed8	-	5+n	9	8-30	3
mem,immed8	-	8+n	10	9-31	3-5

- see Instruction Timing

REP - Repeat String Operation

```
        Usage:  REP
        Modifies flags: None

        Repeats execution of string instructions while CX != 0.
After
        each string operation, CX is decremented and the Zero Flag
is
        tested.  The combination of a repeat prefix and a segment
override
        on CPU's before the 386 may result in errors if an
interrupt occurs
        before CX=0.  The following code shows code that is
susceptible to
        this and how to avoid it:

         again:  rep movs  byte ptr ES:[DI],ES:[SI]   ; vulnerable
instr.
                 jcxz  next              ; continue if REP
successful
                 loop  again             ; interrupt goofed
count
          next:
```

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2		1

- see Instruction Timing REPE REPNE

REPE/REPZ - Repeat Equal / Repeat Zero

Usage: REPE
REPZ
Modifies flags: None

Repeats execution of string instructions while CX != 0 and the Zero Flag is set. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2		1

- see Instruction Timing

RET/RETF - Return From Procedure

```
        Usage:  RET     nBytes
                RETF    nBytes
                RETN    nBytes
        Modifies flags: None

        Transfers control from a procedure back to the instruction
address
        saved on the stack.  "n bytes" is an optional number of
bytes to
        release.  Far returns pop the IP followed by the CS, while
near
        returns pop only the IP register.
```

Operands	Clocks 808x	286	386	486	Size Bytes
retn	16/20	11+m	10+m	5	1
retn immed	20/24	11+m	10+m	5	3
retf	26/34	15+m	18+m	13	1
retf (PM, same priv.)	-		32+m	18	1
retf (PM, lesser priv.)	-		68	33	1
retf immed	25/33	15+m	18+m	14	3
retf immed (PM, same priv.)			32+m	17	1
retf immed (PM, lesser priv.)			68	33	1

- see Instruction Timing

ROL - Rotate Left

Usage: ROL dest,count
Modifies flags: CF OF

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,1	2	2	3	3	2
mem,1	15+EA	7	7	4	2-4 (W88=23+EA)
reg,CL	8+4n	5+n	3	3	2
mem,CL	20+EA+4n	8+n	7	4	2-4 (W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3
mem,immed8	-	8+n	7	4	3-5

- see Instruction Timing

ROR - Rotate Right

Usage: ROR dest,count
Modifies flags: CF OF

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the value of the last bit rotated out.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,1	2	2	3	3	2
mem,1	15+EA	7	7	4	2-4 (W88=23+EA)
reg,CL	8+4n	5+n	3	3	2
mem,CL	20+EA+4n	8+n	7	4	2-4 (W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3
mem,immed8	-	8+n	7	4	3-5

- see Instruction Timing

SAHF - Store AH Register into FLAGS

Usage: SAHF
Modifies flags: AF CF PF SF ZF

Transfers bits 0-7 of AH into the Flags Register. This includes
AF, CF, PF, SF and ZF.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	4	2	3	2	1

- see Instruction Timing

SAL/SHL - Shift Arithmetic Left / Shift Logical Left

```
Usage:  SAL     dest,count
        SHL     dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)
```

Shifts the destination left by "count" bits with zeroes shifted in on right. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	**808x**	**286**	**386**	**486**	**Bytes**	
reg,1	2	2	3	3	2	
mem,1	15+EA	7	7	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3	3	2	
mem,CL	20+EA+4n	8+n	7	4	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3	
mem,immed8	-	8+n	7	4	3-5	

- see Instruction Timing

SAR - Shift Arithmetic Right

Usage: SAR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)

Shifts the destination right by "count" bits with the current sign bit replicated in the leftmost bit. The Carry Flag contains the last bit shifted out.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,1	2	2	3	3	2
mem,1	15+EA	7	7	4	2-4 (W88=23+EA)
reg,CL	8+4n	5+n	3	3	2
mem,CL	20+EA+4n	8+n	7	4	2-4 (W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3
mem,immed8	-	8+n	7	4	3-5

- see Instruction Timing

SBB - Subtract with Borrow

Usage: SBB dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts the source from the destination, and subtracts 1 extra if the Carry Flag is set. Results are returned in "dest".

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	6	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	7	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=25+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

SCAS - Scan String (Byte, Word or Doubleword)

```
Usage:  SCAS    string
        SCASB
        SCASW
        SCASD   (386+ only)
Modifies flags: AF CF OF PF SF ZF
```

Compares value at ES:DI (even if operand is specified) from the accumulator and sets the flags similar to a subtraction. DI is incremented/decremented based on the instruction format (or operand size) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
string	15	7	7	6	1 (W88=19)

- see Instruction Timing

SETAE/SETNB - Set if Above or Equal / Set if Not Below

(unsigned, 386+ only)

```
Usage:  SETAE   dest
        SETNB   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Carry Flag is clear otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETB/SETNAE - Set if Below / Set if Not Above or Equal

(unsigned, 386+ only)

```
Usage:  SETB    dest
        SETNAE  dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Carry Flag is set otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETBE/SETNA - Set if Below or Equal / Set if Not Above

(unsigned, 386+ only)

```
Usage:  SETBE   dest
        SETNA   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Carry Flag or the Zero
Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETC - Set if Carry (386+ only)

```
Usage:  SETC    dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Carry Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETE/SETZ - Set if Equal / Set if Zero (386+ only)

```
Usage:  SETE    dest
        SETZ    dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Zero Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETG/SETNLE - Set if Greater / Set if Not Less or Equal

(signed, 386+ only)

```
Usage:  SETG    dest
        SETNLE  dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Zero Flag is clear or the Sign Flag equals to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETGE/SETNL - Set if Greater or Equal / Set if Not Less

(signed, 386+ only)

```
Usage:  SETGE   dest
        SETNL   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Sign Flag equals the
Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETL/SETNGE - Set if Less / Set if Not Greater or Equal

(signed, 386+ only)

```
Usage:  SETL    dest
        SETNGE  dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Sign Flag is not equal
to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETLE/SETNG - Set if Less or Equal / Set if Not greater or Equal

(signed, 386+ only)

```
        Usage:  SETLE   dest
                SETNG   dest
        Modifies flags: none
```

Sets the byte in the operand to 1 if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETNC - Set if Not Carry (386+ only)

Usage: SETNC dest
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETNE/SETNZ - Set if Not Equal / Set if Not Zero (386+ only)

```
Usage:  SETNE   dest
        SETNZ   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Zero Flag is clear, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETNO - Set if Not Overflow (386+ only)

Usage: SETNO dest
Modifies flags: none

Sets the byte in the operand to 1 if the Overflow Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+ only)

```
Usage:  SETNP   dest
        SETPO   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Parity Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETNS - Set if Not Signed (386+ only)

```
Usage:  SETNS   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Sign Flag is clear, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETO - Set if Overflow (386+ only)

Usage: SETO dest
Modifies flags: none

Sets the byte in the operand to 1 if the Overflow Flag is set,
otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETP/SETPE - Set if Parity / Set if Parity Even (386+ only)

```
Usage:  SETP    dest
        SETPE   dest
Modifies flags: none
```

Sets the byte in the operand to 1 if the Parity Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SETS - Set if Signed (386+ only)

Usage: SETS dest
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is set, otherwise
sets the operand to 0.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg8	-	-	4	3	3
mem8	-	-	5	4	3

- see Instruction Timing

SHLD/SHRD - Double Precision Shift (386+ only)

```
        Usage:  SHLD    dest,src,count
                SHRD    dest,src,count
        Modifies flags: CF PF SF ZF (OF,AF undefined)
```

SHLD shifts "dest" to the left "count" times and the bit positions opened are filled with the most significant bits of "src". SHRD shifts "dest" to the right "count" times and the bit positions opened are filled with the least significant bits of the second operand. Only the 5 lower bits of "count" are used.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16,reg16,immed8	-	-	3	2	4
reg32,reg32,immed8	-	-	3	2	4
mem16,reg16,immed8	-	-	7	3	6
mem32,reg32,immed8	-	-	7	3	6
reg16,reg16,CL	-	-	3	3	3
reg32,reg32,CL	-	-	3	3	3
mem16,reg16,CL	-	-	7	4	5
mem32,reg32,CL	-	-	7	4	5

- see Instruction Timing

SHR - Shift Logical Right

Usage: SHR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)

Shifts the destination right by "count" bits with zeroes shifted in on the left. The Carry Flag contains the last bit shifted out.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,1	2	2	3		2
mem,1	15+EA	7	7		2-4 (W88=23+EA)
reg,CL	8+4n	5+n	3		2
mem,CL	20+EA+4n	8+n	7		2-4 (W88=28+EA+4n)
reg,immed8	-	5+n	3		3
mem,immed8	-	8+n	7		3-5

- see Instruction Timing

SIDT - Store Interrupt Descriptor Table (286+ privileged)

```
Usage:  SIDT    dest
Modifies flags: none
```

Stores the Interrupt Descriptor Table (IDT) Register into the specified operand.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
mem64	-	12	9	10	5

- see Instruction Timing

SLDT - Store Local Descriptor Table (286+ privileged)

Usage: SLDT dest
Modifies flags: none

Stores the Local Descriptor Table (LDT) Register into the specified operand.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	2	2	2	3
mem16	-	2	2	3	5

- see Instruction Timing

STC - Set Carry

Usage: STC
Modifies flags: CF

Sets the Carry Flag to 1.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	2	1

- see Instruction Timing

STD - Set Direction Flag

Usage: STD
Modifies flags: DF

Sets the Direction Flag to 1 causing string instructions to auto-decrement SI and DI instead of auto-increment.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	2	1

- see Instruction Timing

STI - Set Interrupt Flag (Enable Interrupts)

```
Usage:  STI
Modifies flags: IF
```

Sets the Interrupt Flag to 1, enabling recognition of all CPU hardware interrupts.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	2	2	2	5	1

- see Instruction Timing

STOS - Store String (Byte, Word or Doubleword)

```
Usage:  STOS    dest
        STOSB
        STOSW
        STOSD
Modifies flags: None
```

Stores value in accumulator to location at ES:(E)DI (even if operand is given). (E)DI is incremented/decremented based on the size of the operand (or instruction format) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
dest	11	3	4	5	1 (W88=15)

- see Instruction Timing

STR - Store Task Register (286+ privileged)

```
Usage:  STR     dest
Modifies flags: None
```

Stores the current Task Register to the specified operand.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	2	2	2	3
mem16	-	3	2	3	5

- see Instruction Timing

SUB - Subtract

Usage: SUB dest,src
Modifies flags: AF CF OF PF SF ZF

The source is subtracted from the destination and the result is stored in the destination.

Operands	Clocks 808x	286	386	486	Size Bytes
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	6	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	7	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=25+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

TEST - Test For Bit Pattern

Usage: TEST dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands updating the flags register without saving the result.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	1	1	2
reg,mem	9+EA	6	5	1	2-4 (W88=13+EA)
mem,reg	9+EA	6	5	2	2-4 (W88=13+EA)
reg,immed	5	3	2	1	3-4
mem,immed	11+EA	6	5	2	3-6
accum,immed	4	3	2	1	2-3

- see Instruction Timing

VERR - Verify Read (286+ protected)

Usage: VERR src
Modifies flags: ZF

Verifies the specified segment selector is valid and is readable
at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	14	10	11	3
mem16	-	16	11	11	5

- see Instruction Timing

VERW - Verify Write (286+ protected)

Usage: VERW src
Modifies flags: ZF

Verifies the specified segment selector is valid and is ratable
at the current privilege level. If the segment is writable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg16	-	14	15	11	3
mem16	-	16	16	11	5

- see Instruction Timing

WBINVD - Write-Back and Invalidate Cache

Usage: WBINVD
Modifies flags: None

Flushes internal cache, then signals the external cache to write back current data followed by a signal to flush the external cache.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
none	-	-	-	5	2

- see Instruction Timing

XLAT/XLATB - Translate

Usage: XLAT translation-table
XLATB (masm 5.x)
Modifies flags: None

Replaces the byte in AL with byte from a user table addressed by
BX. The original value of AL is the index into the translate table.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
table	11	5	5	4	1

- see Instruction Timing

XOR - Exclusive OR

Usage: XOR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a bitwise exclusive OR of the operands and returns the result in the destination.

	Clocks				Size
Operands	**808x**	**286**	**386**	**486**	**Bytes**
reg,reg	3	2	2	1	2
mem,reg	16+EA	7	6	3	2-4 (W88=24+EA)
reg,mem	9+EA	7	7	2	2-4 (W88=13+EA)
reg,immed	4	3	2	1	3-4
mem,immed	17+EA	7	7	3	3-6 (W88=25+EA)
accum,immed	4	3	2	1	2-3

- see Instruction Timing

C Programming Topics

_based	_bfree	_bheapchk	_bheapset	_bheapwalk
_bios_disk	_bios_equiplist	_bios_keybrd	_bios_memsize	_bios_printer
_bios_serialcom	_bios_timeofday	_bmalloc	_bmsize	_chain_intr
_chmod	_clear87	_clearscreen	_close	_control87
_creat	_disable	_displaycursor	_dos_allocmem	_dos_close
_dos_creat	_dos_creatnew	_dos_findfirst	_dos_findnext	_dos_freemem
_dos_getdate	_dos_getdiskfree	_dos_getdrive	_dos_getfileattr	_dos_getftime
_dos_gettime	_dos_getvect	_dos_keep	_dos_open	_dos_read
_dos_setblock	_dos_setdate	_dos_setdrive	_dos_setfileattr	_dos_setftime
_dos_settime	_dos_setvect	_dos_write	_elipse_wxy	_ellipse
_ellipse_w	_enable	_exit	_expand	_fexpand
_ffree	_fheapchk	_fheapset	_fheapwalk	_floodfill
_floodfill_w	_fmalloc	_fmsize	_fpreset	_freect
_getcolor	_getcurrentposition	_getfillmask	_getimage	_getimage_w
_getimage_wxy	_getlinestyle	_getlogcoord	_getphyscoord	_getpixel
_getpixel_w	_gettextcolor	_gettextposition	_getvideoconfig	_getviewcoord
_harderr	_hardresume	_hardretn	_heapchk	_heapset
_heapwalk	_lineto	_lineto_w	_lrotl	_lrotr
_makepath	_matherr	_memavl	_moveto	_moveto_w
_msize	_nexpand	_nfree	_nheapchk	_nheapset
_nheapwalk	_nmalloc	_nmsize	_open	_outtext
_pie	_pie_wxy	_putimage	_putimage_w	_read
_rectangle	_rectangle_w	_rectangle_wxy	_remapallpalette	_remappalette
_rotl	_rotr	_searchenv	_selectpalette	_setactivepage
_setbkcolor	_setcliprgn	_setcolor	_setfillmask	_setfont
_setlinestyle	_setlogorg	_setpixel	_setpixel_w	_settextcolor
_settextposition	_settextwindow	_setvideomode	_setvieworg	_setviewport
_setvisualpage	_splitpath	_status87	_strdate	_strerror
_strtime	_tolower	_toupper	_wrapon	_write
abort	abs	absread	abswrite	access
acos	alloca	allocmem	asctime	asin

assert	atan	atan2	atexit	atof
atoi	atol	bdos	bdosptr	bioscom
biosdisk	biosequip	bioskey	biosmemory	biosprint
biostime	brk	bsearch	byteregs	C declarations
C errors (MSC)	C escape sequences	C operators	cabs	calloc
ceil	cgets	chdir	chmod	cl
clearerr	clock	close	coreleft	cos
cosh	country	cprintf	cputs	creat
creatnew	creattemp	cscanf	ctime	ctrlbrk
dieeetomsbin	difftime	disable	div (c)	dmsbintoieee
dosexterr	dostounix	dup	dup2	ecvt
enable	eof	exec	exec...	exit
exp	fabs	farcalloc	farcoreleft	farfree
farrealloc	fclose	fcloseall	fcvt	fdopen
feof	ferror	fflush	fgetc	fgetchar
fgets	fieeetomsbin	filelength	fileno	findfirst
findnext	floor	flushall	fmod	fmsbintoieee
fnmerge	fnsplit	fopen	fp_off	fp_seg
fprintf	fputc	fputs	fread	free
freemem	freopen	frexp	fscanf	fseek
fstat	ftell	fwrite	gcvt	geninterrupt
getc	getcbrk	getch	getchar	getche
getcolor	getcurdir	getcwd	getdate	getdfree
getdisk	getdta	getenv	getfat	getfatd
getftime	getimage	getpass	getpid	getpsp
gets	gettime	getvect	getverify	getw
gmtime	gsignal	halloc	harderr	hardresume
hardretn	hfree	hypot	imagesize	inp
inport	inportb	inpw	int86	int86x
intdos	intdosx	intr	ioctl	is...
isalnum	isalpha	isascii	isatty	iscntrl
isdigit	isgraph	islower	isprint	ispunct
isspace	isupper	isxdigit	itoa	kbhit
keep	labs	ldexp	ldiv	lfind

localtime | lock (C) | locking | log | log10
longjmp | lsearch | lseek | ltoa | malloc
matherr | mem... | memccpy | memchr | memcmp
memcpy | memicmp | memmove | memset | mk_fp
mkdir | mktemp | modf | movedata | movmem
MSC errors | MSC options | onexit | open | operator precedence
outp | outport | outportb | outpw | parsfnm
peek | peekb | perror | poke | pokeb
poly | pow | pow10 | printf | printf specifiers
putc | putch | putchar | putcolor | putenv
putimage | puts | putw | qsort | raise
rand | randbrd | randbrw | read | realloc
regpack | regs | remove | rename | rewind
rmdir | rmtmp | sbrk | scanf | scanf specifiers
searchpath | segread | setblock | setbuf | setcbrk
setdate | setdisk | setdta | setftime | setjmp
setmem | setmode | settime | setvbuf | setvect
setverify | sin | sinh | sleep | sopen
spawn | spawn... | sprintf | sqrt | srand
sregs | sscanf | ssignal | stackavail | stat
stdarg | stime | stpcpy | str... | strcat
strchr | strcmp | strcpy | strcspn | strdup
strerror | stricmp | strings | strlen | strlwr
strncat | strncmp | strncpy | strnicmp | strnset
strpbrk | strrchr | strrev | strset | strspn
strstr | strtod | strtok | strtol | strtoul
strupr | swab | system | tan | tanh
tell | tempnam | time | tmpfile | tmpnam
toascii | tolower | toupper | tzset | ultoa
umask | ungetc | ungetch | unixtodos | unlink
unlock | utime | va_end | va_list | va_start
vfprintf | vfscanf | vprintf | vscanf | vsprintf
vsscanf | wordregs | write

MSC: void *_expand(void *block, size_t size) MSC: void far *_fexpand(void far *block, size_t size) MSC: void near *_nexpand(void near *block, size_t size) MSC: void _based(void) *_bexpand(_segment seg, void _based(void) *block, size_t size) - prototype in malloc.h - block = pointer to previously allocated memory block - size = new size in bytes - seg = based segment value - returns pointer to reallocated memory block on success or NULL if not; (_bexpand returns -1 on failure)

MSC: void _bfree(_segment seg, void _based(void) *block) MSC: void _ffree(void far *block) MSC: void _nfree(void near *block) - prototype in malloc.h - block = previously allocated buffer - in large data memory models free() maps to _ffree() - in small data memory models free() maps to _nfree() - NULL pointer parameter is ignored

MSC: int _heapchk(void) MSC: int _bheapchk(_segment seg) MSC: int _fheapchk(void) MSC: int _nheapchk(void) - prototype in malloc.h - runs consistency check on different heaps - heapchk maps to the other functions depending on memory model - returns one of the following: _HEAPOK heap is ok _HEAPBADBEGIN initials allocation header couldn't be found _HEAPBADNODE heap/node has been damaged _HEAPEMPTY heap has not been initialized - see heapset() heapwalk()

MSC: int _heapset(unsigned fill) MSC: int _bheapset(_segment seg, unsigned fill) MSC: int _fheapset(unsigned fill) MSC: int _nheapset(unsigned fill) - prototype in malloc.h - heapset maps to the other functions depending on memory model - first check heap consistency via heapchk() then fills memory with value specified in "fill" - returns one of the following: _HEAPOK heap is ok _HEAPBADBEGIN initials allocation header couldn't be found _HEAPBADNODE heap/node has been damaged _HEAPEMPTY heap has not been initialized - see heapchk() heapwalk()

MSC: int _heapwalk(struct _heapinfo *entry) MSC: int _bheapwalk(_segment seg, struct _heapinfo *entry) MSC: int _fheapwalk(struct _heapinfo *entry) MSC: int _nheapwalk (struct _heapinfo *entry) - prototype in malloc.h - walks the heap for each entry specified and returns information about the entry via the other fields of the _heapinfo structure - heapwalk maps to the other functions depending on memory model - returns one of the following: _HEAPOK heap is ok _HEAPBADBEGIN initials allocation header couldn't be found _HEAPBADNODE heap/node has been damaged _HEAPEMPTY heap has not been initialized _HEAPBADPTR parameter does not contain valid pointer to heap _HEAPEND end of help found w/o problem struct _heapinfo { int _far *_pentry; - heap entry pointer size_t _size; - size of heap entry int _useflag; - entry in use return value }; - see heapchk() heapset()

MSC: unsigned _bios_disk(unsigned cmd,struct diskinfo_t *dskinfo)

```
        - prototype in bios.h

        - cmd   = _DISK_RESET   resets disk system  (INT 13,0)
                = _DISK_STATUS  return disk status  (INT 13,1)
                = _DISK_READ    read disk sectors  (INT 13,2)
                = _DISK_WRITE   write disk sectors  (INT 13,3)
                = _DISK_VERIFY  verify disk sectors  (INT 13,4)
                = _DISK_FORMAT  format disk track  (INT 13,5)

        - dskinfo = pointer to structure containing disk
information:

        struct diskinfo_t
           {
           unsigned drive;              - physical drive number
           unsigned head;               - head number
           unsigned track;              - track number
           unsigned sector;             - sector number
           unsigned nsectors;           - sectors to deal with
           void _far *buffer;           - data buffer
           };

        - returns status of operation (AX register, AH contains
status)
        - drive numbers represent physical drives (0 = A:, 1 = B:,
          80h = drive 0, 81h = drive 1
        - disk reads should be retried at least 3 times
        - Turbo C uses biosdisk()

        - see  INT 13
```

TC: int biosdisk(int cmd, int drive, int head, int track, int sector, int nsects, void *buffer) - prototype in bios.h - issues interrupt 0x13 for BIOS disk operations - drive = (0=A:, 1=2nd floppy, 0x80=drive 0, 0x81=drive 1 - cmd = 0 reset diskette system (hard disk drive reset) 1 return status of last disk operation, ignore parms 2 read nsects disk sectors, with starting sector defined by head, track, & sector, reading 512 bytes per sector into buffer 3 write nsects disk sectors, with starting sector defined by head, track, & sector, writing 512 bytes per sector from buffer 4 verify nsects disk sectors, with starting sector defined by head, track, & sector 5 format a track defined by head & track, where buffer points to a table of sector headers to be written For XT or AT only: 6 - formats a track & sets bad sector flags 7 - formats drive beginning at specific track 8 - returns current drive parms in 1rst 4 bytes of buffer 9 - initializes drive-pair characteristics 10 - long read (4 extra bytes per sector for ECC) 11 - long write (4 extra bytes per sector for ECC) 12 - disk seek 13 - alternate disk reset 14 - reads sector buffer 15 - writes sector buffer 16 - tests if named drive is ready 17 - recalibrates drive 18 - controller ram diagnostic 19 - drive diagnostic 20 - controller internal diagnostic - drive numbers represent physical drives (0 = A:, 1 = B:, 80h = drive 0, 81h = drive 1 - disk reads should be retried at least 3 times - MS C uses _bios_disk() - see INT 13 for more return codes and more information

MSC: unsigned _bios_equiplist(void)

- prototype in bios.h

- returns flags found in BIOS data area at 40:10 & 40:11
- Turbo C uses biosequip()
- see INT 11

TC: int biosequip(void)

- prototype in bios.h

- returns integer showing equipment in system
- MS C uses _bios_equiplist()
- see INT 11 or EQUIPMENT FLAGS for bit meanings

MSC: int _bios_keybrd(int cmd)

- prototype in bios.h

- cmd = _KEYBRD_READ return next keystroke w/o echo (INT 16,0)
= _KEYBRD_READY test if key avail; 0=none (INT 16,1)
= _KEYBRD_SHIFTSTATUS return shift key status (INT 16,2)
= _NKEYBRD_READ return next keystroke w/o echo (INT 16,10)
= _NKEYBRD_READY test if key avail; 0=none (INT 16,11)
= _NKEYBRD_SHIFTSTATUS return shift key status (INT 16,12)

- AND returned value with 0x00FF to get ASCII value
- _NKEYBRD_READ, _NKEYBRD_READY, _NKEYBRD_SHIFTSTATUS are available
on AT and newer machines only
- Turbo C uses bioskey()
- see INT 16

TC: int bioskey(int cmd)

- prototype in bios.h

- keyboard operations
- cmd = 0 return next keystroke w/o echo (INT 16,0)
= 1 test if key waiting; 0=none (INT 16,1)
= 2 return shift key status (see INT 16,2)

- AND returned value with 0x00FF to get ASCII value
- Ctrl-Break causes "while (!bioskey(1))" to loop infinitely in
Turbo C 2.0 and earlier since it places a zero in the keyboard queue
- MS C uses _bios_keybrd()
- see INT 16

MSC: unsigned _bios_memsize(void)

- prototype in bios.h

- returns memory size in 1K blocks (16..640K)
- Turbo C uses biosmemory()
- see INT 12

TC: int biosmemory(void)

- prototype in bios.h

- returns BIOS size memory in 1K blocks
- MS C uses _bios_memsize()
- see INT 12

unsigned _bios_printer(unsigned cmd,unsigned port,unsigned data) - prototype in bios.h - cmd = _PRINTER_WRITE print character (INT 17,0) = _PRINTER_INIT initialize printer port (INT 17,1) = _PRINTER_STATUS read port status (INT 17,2) - port = printer port; 0=LPT1, 1=LPT2 - data = value to write - returns status of printer (INT 17, AX reg, AH contains status) - Turbo C uses biosprint () - see INT 17

TC: int biosprint(int cmd, int byte, int port)

- prototype in bios.h

- outputs byte to port
- port = 0 (LPT1)
  1 (LPT2), etc...
- cmd = 0 (print byte);
  1 (init printer port);
  2 (get LPT status)

- returns status of (values are or'd flags):
  0x01 time out (cmd=0 means output error)
  0x08 I/O error
  0x10 selected
  0x20 out of paper
  0x40 acknowledge
  0x80 not busy

- MS C uses _bios_printer()
- see INT 17

MSC: unsigned _bios_serialcom(unsigned cmd, unsigned port,unsigned fmt)

- prototype in bios.h

- cmd = _COM_INIT initialize port (INT 14,0)
  = _COM_SEND send character (INT 14,1)
  = _COM_RECEIVE read character from port (INT 14,2)
  = _COM_STATUS get serial port status (INT 14,3)

- fmt = combination of the following:
  _COM_CHR7 _COM_CHR8
  _COM_STOP1 _COM_STOP2

  _COM_NOPARITY _COM_EVENPARITY _COM_ODDPARITY

  _COM_110 _COM_150 _COM_300
  _COM_600 _COM_1200 _COM_2400
  _COM_4800 _COM_9600

- returns status word (INT 14, AX reg, modem status in AL, port status in AH)
- Turbo C uses bioscom()
- see INT 14

TC: int bioscom(int cmd, char byte, int port)

- prototype in bios.h

- RS232 communications

- cmd = 0 to set comm parms to value in byte
  1 to send char in byte out on RS232 line
  2 to receive char
  3 to return current port status

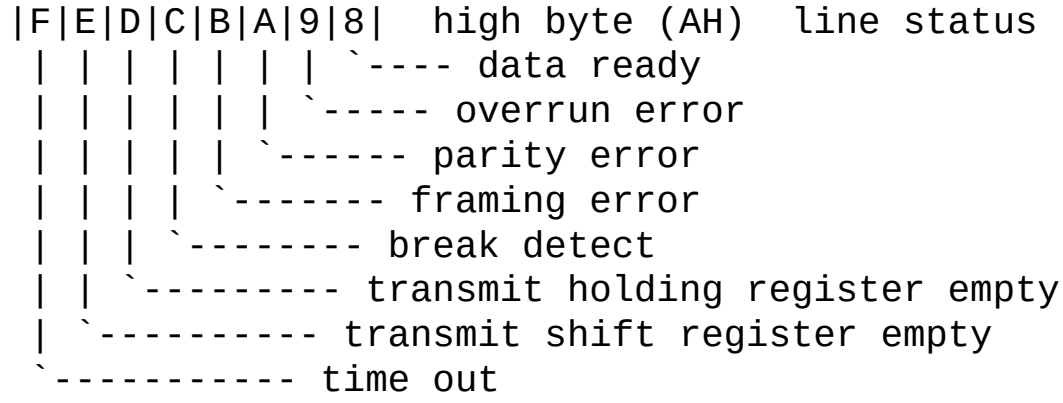
- port = 0 for COM1
  1 for COM2
  2 for COM3
  3 for COM4

- byte = bitwise OR of following bits
  0x02 = 7 data bits 0x00 = 1 stop bit
  0x03 = 8 data bits 0x04 = 2 stop bits
  0x00 == no parity;
  0x08 == odd parity;
  0x18 == even parity

0x00	110 bps	0x20	150 bps
0x40	300 bps	0x60	600 bps
0x80	1200 bps	0xa0	2400 bps
0xc0	4800 bps	0xe0	9600 bps

- returns 16-bit integer, upper 8 bits are status & lower 8 bits
  depending on cmd sent

```
|7|6|5|4|3|2|1|0|  low byte (AL)  modem status
 | | | | | | | `---- delta clear to send
 | | | | | | `----- delta data set ready
 | | | | | `------ trailing edge ring detector
 | | | | `------- delta receive line signal detect
 | | | `-------- clear to send
 | | `--------- data set ready
 | `---------- ring indicator
 `----------- receive line signal detect
```



- MS C uses _bios_serialcom()
- see INT 14

MSC: unsigned _bios_timeofday(unsigned cmd, long *timeval)

- prototype in bios.h

- cmd = _TIME_GETCLOCK get real time clock (INT 1A,0)
  = _TIME_SETCLOCK set real time clock (INT 1A,1)

- returns current BIOS clock count
- Turbo C uses biostime()
- see INT 1A

TC: int biostime(int cmd, long newtime)

- prototype in bios.h

- reads or sets BIOS timer
- cmd = 0 returns current value of timer
        1 sets timer to value of newtime
- MS C uses _bios_timeofday()

MSC: void _based(void) *_bmalloc(_segment seg, size_t size) MSC: void far *_fmalloc(size_t size) MSC: void near *_nmalloc(size_t size) - prototype in malloc.h - size = number of bytes to allocate - seg = base segment value - malloc() maps to _fmalloc or _nmalloc based on memory model - _fmalloc allocates memory outside the default data segment if possible - returns pointer to allocated block or NULL on error; _bmalloc() function returns _NULLOFF on error - these functions will allocate a zero length block

MSC: size_t _msize(void *block) MSC: size_t _bmsize(_segment seg, void _based(void) *block) MSC: size_t _fmsize(void far *block) MSC: size_t _nmsize(void near *block) - prototype in malloc.h - returns size of memory block in bytes - _msize() maps to other function depending on memory model

MSC: void _chain_intr(void (interrupt far *target)())

- prototype in dos.h

- chains from one interrupt handler to another; sets up the stack as if the interrupt was called directly by pushing the
  flags and the original return address on the stack
- this function does not return to caller but returns to the code where the interrupt occurred
- no corresponding function in Turbo C

TC: int _chmod(const char *filename, int func [, int attrib])

- prototype in io.h, also include dos.h

- if func = 0, returns access mode of file
- if func = 1, sets access mode of file
- attrib can be one of the following:

Turbo C	MSC
FA_RDONLY	_A_RDONLY
FA_HIDDEN	_A_HIDDEN
FA_SYSTEM	_A_SYSTEM

- if func = 1,
  returns 0 if OK
  -1 with errno set to ENOENT or EACCES

- if func = 0, returns MSDOS attribute

unsigned int _clear87(void)

- prototype in float.h

- clears 8087/80287 status word & other exceptions
- returns old status word
- see _status87() _control87()

unsigned int _status87(void)

- prototype in float.h

- returns floating-point status word & other conditions detected
  by 8087/80287 exception handler
- see _clear87() _control87() (float.h also)

MSC: unsigned _control87(unsigned new, unsigned mask)

```
        - prototype in float.h

        - used to change floating point handling and exception
masking
        - new = contains corresponding bit values for any bit set
in "mask"
                to be moved into the coprocessor control word
        - mask  = zero; gets floating-point coprocessor control word
                = non-zero; if any bit is set, the corresponding
bit in
                  "new" is moved into the coprocessor control word
        - returns floating-point coprocessor control word
        - default control word is the logical OR of: IC_AFFINE,
RC_NEAR,
          PC_64, EM_DENORMAL, EM_UNDERFLOW, EM_INEXACT
```

Control Word Mask and Bits

MCW_EM	interrupt exception control
MCW_IC	infinity control
MCW_RC	rounding control
MCW_PC	precision control

Exception Masks

EM_INVALID	invalid
EM_DENORMAL	denormal
EM_ZERODIVIDE	zero divide
EM_OVERFLOW	overflow
EM_UNDERFLOW	underflow
EM_INEXACT	inexact (precision)

Infinity Control

IC_AFFINE	affine
IC_PROJECTIVE	projective

Rounding Control

```
RC_CHOP         truncate
RC_UP           round up
RC_DOWN         round down
RC_NEAR         near
```

Precision Control

```
PC_24           24 bits precision
PC_53           53 bits precision
PC_64           64 bits precision
```

User Status Word Bits

```
SW_INVALID      invalid
SW_DENORMAL     denormal
SW_ZERODIVIDE   zero divide
SW_OVERFLOW     overflow
SW_UNDERFLOW    underflow
SW_INEXACT      inexact precision
```

Invalid Subconditions (if SW_INVALID)

```
SW_UNEMULATED   unemulated instruction
SW_SQRTNEG      square root of a neg number
SW_STACKOVERFLOW  FP stack overflow
SW_STACKUNDERFLOW FP stack underflow
```

Floating Point Errors

```
FPE_INVALID
FPE_DENORMAL
FPE_ZERODIVIDE
FPE_OVERFLOW
FPE_UNDERFLOW
FPE_INEXACT
FPE_UNEMULATED
FPE_SQRTNEG
FPE_STACKOVERFLOW
FPE_STACKUNDERFLOW
FPE_EXPLICITGEN
```

- see _clear87() _status87()

MSC: void far _clearscreen(short target)

- prototype in graph.h

- target = _GCLEARSCREEN clears screen
  = _GVIEWPORT clears current viewport
  = _GWINDOW clears current text window
- available in graphics mode only
- clears target area by filling with current background color

TC: int _close(int fh)

- prototype in io.h

- closes file fh where fh is file handle from _creat(), creat(), creatnew(), creattemp(), dup(), dup2(), _open(), or open().
- use fdopen() to change file handle to file stream.
- does not write a Ctrl-Z to a text file on closing
- returns 0 if OK; -1 & errno=EBADF
- MS C uses _dos_close()

- see INT 21,3E

MSC: unsigned _dos_creatnew(char *fname,unsigned attr,int *handle) MSC: unsigned _dos_creat(char *fname,unsigned attr,int *handle) - prototype in dos.h - fname = a valid DOS file name - attr = DOS file attribute = _A_RDONLY = _A_HIDDEN = _A_SYSTEM = _A_ARCH - handle = pointer to location to receive handle - returns 0 on success or DOS error code on failure - if file sharing is installed, file is opened in compatibility mode - see INT 21,3C INT 21,5B

TC: int creattemp(char *filename, int attr)

- prototype in io.h, also include dos.h

- creates new file (overwrites existing file) of filename with file
  attribute attrib, for both read/write
- attr consists of a combination of the following:

  FA_RDONLY set read only attribute
  FA_HIDDEN set hidden attribute
  FA_SYSTEM set system attribute

- filename is a path name ending in a backslash (\); when file is
  created, filename has a unique path name
- file is created set to _fmode (O_TEXT or O_BINARY)
- returns nonnegative integer if OK, else -1

char *strdup(const char *str)

- prototype in string.h

- duplicates str, getting space with a call to malloc()
- returns pointer to duplicated string, or NULL if space could not be allocated

void *malloc(unsigned size)

- prototype in stdlib.h, alloc.h (malloc.h for MS C)

- allocates memory of length size in bytes
- returns pointer if successful, else NULL
- MS C will return a zero length block (allocates header only)

int dup2(int oldhandle, int newhandle)

- prototype in io.h

- returns a new file handle that duplicates the old handle
  equal to newhandle. If newhandle exists, the corresponding
  file is closed.
- returns -1 on error

TC: int _open(const char *path, int access)

- prototype in io.h, also include fcntl.h

- this function will not create a file, use _creat() if it doesn't
  currently exist
- access is a combination of the following:

DOS 2.0+ Attributes

```
O_RDONLY        open read only
O_WRONLY        open write only
O_RDWR  open read/write
```

DOS 3.1+ Attributes

```
O_NOINHERIT     file is not to be passed to child processes
O_DENYALL       file is not accessible to others (exclusive)
O_DENYWRITE     file is read only to all other opens
O_DENYREAD      file is write only to all other opens
O_DENYNONE      file is to be shared by all
```

- MS C uses _dos_open()
- see INT 21,3D open() _creat()

MSC: unsigned _dos_open(char *fname, unsigned mode, int *handle)

```
        - prototype in dos.h, fcntl.h (also include share.h)

        - fname = valid DOS file name
        - mode  = file access permission (a logical or of the
following)
                = O_NOINHERIT   file can't be inherited by child
process
                = O_RDONLY      file is read only
                = O_RDWR        file is read/write
                = O_WRONLY      file is write only
                = SH_COMPAT     file sharing compatibility mode
                = SH_DENYNONE   file sharing deny neither read nor
write
                = SH_DENYRD     file sharing deny read
                = SH_DENYRW     file sharing deny read/write
                = SH_DENYWR     file sharing deny write

        - handle = pointer to memory to contain handle
        - does not creat files, they must currently exist (see
_dos_creat())
        - returns 0 on success or DOS error code

        - Turbo C uses _open()
        - see   INT 21,3D  _dos_close()  _dos_read()
                _dos_write()    _dos_creat()
```

MSC: unsigned _dos_close(int handle)

- prototype in dos.h

- handle is standard DOS file handle attained through _dos_creat(), _dos_creat(), _dos_creatnew(), _dos_open(), dup(), open()
- returns 0 on success or DOS error code on failure
- does not write a Ctrl-Z to a text file on closing
- see INT 21,3E

int sopen(const char *path, int access, int shflag, int mode)

- prototype in io.h, also include share.h, stat.h & fcntl.h

- actually a macro defined: open(path, (access | shflag), mode)
- opens file "path" and prepares it for shared I/O
- sharing mode is determined using access, shflag and mode
- access is a combination of the following ("permis" follows):

Read/Write Access Flags (mutually exclusive):

O_RDONLY	open read only
O_WRONLY	open write only
O_RDWR	open read/write

Other Access Mode Flags:

O_APPEND	file pointer is placed at EOF before each write
O_CREAT	if file doesn't exist, create with "permis" attributes
O_TRUNC	if exists, truncate length to zero, but leave file attributes unchanged
O_BINARY	binary mode
O_TEXT	text mode
O_EXCL	used with O_CREAT, error occurs if file already exists
O_NDELAY	UNIX only

Permission Attributes (if creating):

S_IWRITE	write permission
S_IREAD	read permission
S_IREAD | S_IWRITE	read/write permission

- if O_BINARY nor O_TEXT is given, file is opened in translation
  mode (O_TEXT) given by global variable _fmode
- returns non-negative number as file handle, or -1 on error

MSC: unsigned _dos_read(int handle, void far *buffer, unsigned count, unsigned *nread) - prototype in dos.h - handle = opened file handle - buffer = receives data read from file - count = number of bytes to read - nread = receives the number of bytes actually read - returns 0 on success or DOS error code - Turbo C uses _open() - see INT 21,3F _dos_open() _dos_write() _dos_close()

MSC: unsigned _dos_write(int handle, void far *buffer, unsigned count, unsigned *nbytes) - prototype in dos.h - handle = opened file handle - buffer = buffer of data to write to file - count = number of bytes to write - nbytes = receives the number of bytes actually written - returns 0 on success or DOS error code on failure - Turbo C uses _write() - see INT 21,40 _dos_open() _dos_close() _dos_read()

TC: int _write(int handle, char *buf, int count)

- prototype in io.h

- writes "count" bytes from "buf" to file or device at "handle"
- count cannot be greater that 65534
- is a direct call to MS-DOS write function INT 21,40
- does NO conversion of CR/LFs; all output is binary
- returns a count of bytes written or -1 on error
- MS C uses _dos_write()

- see write() _open() _creat() _read()

int write(int handle, char *buf, int count)

- prototype in io.h

- writes "count" bytes from "buf" to file or device at "handle"
- converts LF to CR-LF for text output
- returns a count of bytes written (excluding any CRs generated for a text file), or -1 on error

TC: int _read(int handle, void *buf, int size)

- prototype in io.h

- reads "size" bytes from file "handle" into "buf"
- size must be less than 65534 bytes
- is a direct call to MS-DOS read function INT 21,3F
- does not translate CR/LF; all input in binary
- returns number of bytes successfully transferred or -1
- MS C uses _dos_read()

- see read() _open() _creat() _write() _close()

void segread(struct SREGS *segregs)

- prototype in dos.h

- places current values of segment registers (SEGREGS) into segtbl
- available in Turbo C and MS C
- unique to DOS
- see REGS

Register Structures for Turbo C and MS C

Turbo C

```
struct WORDREGS
   {
   unsigned int ax;
   unsigned int bx;
   unsigned int cx;
   unsigned int dx;
   unsigned int si;
   unsigned int di;
   unsigned int cflag;
   unsigned int flags;
   };

struct BYTEREGS
   {
   unsigned char al;
   unsigned char ah;
   unsigned char bl;
   unsigned char bh;
   unsigned char cl;
   unsigned char ch;
   unsigned char dl;
   unsigned char dh;
   };

union REGS
   {
   struct   WORDREGS x;
   struct   BYTEREGS h;
   };

struct SREGS
   {
   unsigned int es;
   unsigned int cs;
   unsigned int ss;
   unsigned int ds;
   };

struct REGPACK
```

MS C

```
struct WORDREGS
    {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int cflag;
    };
```

- same as Turbo C

- same as Turbo C

- same as Turbo C

```
        {
        unsigned r_ax;
        unsigned r_bx;
        unsigned r_cx;
        unsigned r_dx;
        unsigned r_bp;                  Turbo C only
        unsigned r_si;
        unsigned r_di;
        unsigned r_ds;
        unsigned r_es;
        unsigned r_flags;
        };
```

FILE *fdopen(int handle, char *type)

- prototype in dos.h

- associates a file STREAM with a file HANDLE
- handle is returned by creat(), dup(), dup2() or open()
- type must match mode of the handle
- returns file stream or NULL on error
- see fopen()

FILE *fopen(const char *filename, const char *type)

- prototype in stdio.h

- type is a combination of the following:

Mode	Description
"r"	read text/binary
"w"	write text/binary
"a"	append text/binary
"+"	allow update access
"t"	text file
"b"	binary file

Read	Write	Append	
"rt"	"wt"	"at"	(text)
"rb"	"wb"	"ab"	(binary)
"r+"	"w+"	"a+"	(update)
"r+t"	"w+t"	"a+t"	(update text)
"r+b"	"w+b"	"a+b"	(update binary)

read - read only (unless "r+")
write - create
append - seek to end of file or create file

- text mode input, will have CRs discarded
- using any of the stdio functions results in a default allocation
  of 512 bytes for the I/O buffer and the inclusion of the standard
  memory allocation functions
- returns stream or NULL on error

TC: int _creat(const char *fname, int attr)

- prototype in io.h, also include dos.h

- creates new file (overwrites existing file) of filename with file
  attribute attr, for both read/write
- attr can be a combination of the following

  FA_RDONLY   set read only attribute
  FA_HIDDEN   set hidden attribute
  FA_SYSTEM   set system attribute

- file is created in O_BINARY mode
- returns nonnegative integer if OK, else -1
- uses DOS function INT 21,3C
- MSC uses _dos_creat()

TC: void disable(void) MSC: void _disable(void) - prototype in dos.h - disables all interrupts except NMI via CLI instruction

MSC: short far _displaycursor(short flag)

```
        - prototype in graph.h

        - flag = _GCURSORON  turn cursor on
                 _GCURSOROFF turn cursor off
        - cursor is normally turned off while in graphics mode;
calling
          this function determines whether cursor is displayed upon
          exit from graphics mode
        - returns previous value of cursor state flag
```

MSC: unsigned _dos_allocmem(unsigned size, unsigned *seg)

- prototype in dos.h

- size = number of paragraphs to allocate
- seg = pointer to buffer to contain allocated segment value
- returns 0 on success or DOS error code on failure
- Turbo C uses allocmem()
- see INT 21,48

MSC: unsigned _dos_freemem(unsigned seg)

- prototype in dos.h

- frees memory allocated via _dos_allocmem() or INT 21,49
- returns 0 on success or DOS error code
- Turbo C uses freemem()
- see INT 21,49

TC: int freemem(unsigned seg)

- prototype in dos.h

- frees previously allocated DOS block defined by seg
- returns 0 on success, -1 on error errno=ENONMEM
- MS C uses _dos_freemem()
- see INT 21,49

MSC: void _dos_getdate(struct dosdate_t *date)

- prototype in dos.h

- date = pointer to structure to contain returned data of format:

```
struct dosdate_t
  {
  unsigned char day;            - [1-31]
  unsigned char month;          - [1-12]
  unsigned int  year;           - [1980-2099]
  unsigned char dayofweek;      - [0-6] = Sun..Sat
  };
```

- Turbo C uses getdate()
- see INT 21,2A

TC: void getdate(struct date *dateblk)

- prototype in dos.h

- gets DOS date, filling it into the following structures:

```
struct date
    {
    int  da_year;              - Year including century
    char da_day;               - Day of the month
    char da_mon;               - Month (1 = Jan)
    };
```

- MS C uses _dos_getdate()
- see INT 21,2A

INT 21,2A - Get Date

```
AH = 2A


on return:
AL = day of the week (0=Sunday)
CX = year (1980-2099)
DH = month (1-12)
DL = day (1-31)


- retrieves system date based on the DOS maintained clock
- updates BIOS Data Area current date and date rollover flag
  at location 40:70
```

MSC: unsigned _dos_getdiskfree(unsigned drv,struct diskfree_t *spc) - prototype in dos.h - drive = 1 = A:, 2 = B:, 3 = C:, ... - spc = structure to contains disk status information - returns 0 on success or DOS error code on failure struct diskfree_t { unsigned total_clusters; - count of all disk clusters unsigned avail_clusters; - free unallocated clusters unsigned sectors_per_cluster; unsigned bytes_per_sector; }; - Turbo C uses getdfree() - see INT 21,36

TC: void getdfree(unsigned char drive, struct dfree *dfreep)

- prototype in dos.h

- fills in structure with disk status information.

```
struct dfree
    {
    unsigned df_avail;          - available clusters
    unsigned df_total;          - total clusters
    unsigned df_bsec;           - bytes per sector
    unsigned df_sclus;          - sectors per cluster
    };
```

- drive is specified as A = 1, B = 2, C = 3, etc...
- in event of error, df_sclus is set to -1.
- MS C uses _dos_getdiskfree()
- see INT 21,36

INT 21,36 - Get Disk Free Space

```
AH = 36h
DL = drive number (0=default, 1=A:)


on return:
AX = sectors per cluster
   = FFFF if drive is invalid
BX = number of available clusters
CX = number of bytes per sector
DX = number of clusters per drive


- used to determine available space on specified disk
- see   INT 21,1B   INT 21,1C
```

MSC: void _dos_getdrive(unsigned *drive)

- prototype in dos.h

- drive = pointer to buffer to receive drive number;
  0 = A:, 1 = B:, ...
- Turbo C uses getdisk()
- see INT 21,19

TC: int getdisk(void)

- prototype in dir.h

- returns integer drive number; 0 = A:, 1 = B:, ...
- MS C uses _dos_getdrive()
- see INT 21,19

MSC: unsigned _dos_getfileattr(char *path, unsigned *attr)

- prototype in dos.h

- path = any valid DOS file name
- attr = file attribute:

```
_A_NORMAL
_A_RDONLY
_A_HIDDEN
_A_SYSTEM
_A_VOLID
_A_SUBDIR
_A_ARCH
```

- returns zero on success; nonzero on failure
- no corresponding function in Turbo C
- see INT 21,43

MSC: unsigned _dos_getftime(int h,unsigned *date,unsigned *time)

- prototype in dos.h

- h = open file handle
- date = pointer to memory to receive file date
- time = pointer to memory to receive file time
- returns 0 on success or DOS error code
- Turbo C uses getftime()
- see INT 21,57 _dos_setftime()

TC: int getftime(int handle, struct ftime *ftimep)

- prototype in dos.h

- retrieves file time and date for the file associated with handle into ftimep
- structure ftime is defined in dos.h
- MS C uses _dos_getftime()
- see INT 21,57

MSC: unsigned _dos_setftime(int handle, unsigned date,unsigned time)

- prototype in dos.h

- handle = opened file handle
- date = date to set last file write (see FILE ATTRIBUTES)
- time = time to set last file write (see FILE ATTRIBUTES)
- returns 0 on success or DOS error code on failure

- Turbo C uses setftime()
- see INT 21,57 _dos_getftime() FILE ATTRIBUTES

TC: int setftime(int handle, struct ftime *ftimep)

- prototype in dos.h

- sets file time and date for the file associated with handle as
  defined by ftimep
- structure ftime is defined in dos.h
- MS C uses _dos_setftime()
- see INT 21,57

MSC: void _dos_gettime(struct dostime_t *time)

```
- prototype in dos.h

- returns DOS time into "time"

struct dostime_t
    {
    unsigned char hour;         - [0-23]
    unsigned char minute;       - [0-59]
    unsigned char second;       - [0-59]
    unsigned char hsecond;      - hundreths of a second 0-99
    };

- Turbo C uses gettime()
- see   _dos_settime()
```

TC: void gettime(struct time *timep)

- prototype in dos.h

- gets MS-DOS time into the following data structure:

```
  struct time
    {
    unsigned char  ti_min;
    unsigned char  ti_hour;
    unsigned char  ti_hund;
    unsigned char  ti_sec;
    };
```

- MS C uses _dos_gettime()

MSC: unsigned _dos_settime(struct dostime_t *time)

- prototype in dos.h

- time = structure containing new time values
- returns 0 on success or DOS error code on failure
- sets MS-DOS time via INT 21,2D
- DOS 3.3+ also sets CMOS clock

- Turbo C uses settime()
- see INT 21,2D _dos_gettime()

TC: void settime(struct time *timep)

- prototype in dos.h

- sets MS-DOS time via INT 21,2D
- DOS 3.3+ also sets CMOS clock

- MS C uses _dos_settime()
- see INT 21,2D

INT 21,2D - Set Time

```
AH = 2D
CH = hour (0-23)
CL = minutes (0-59)
DH = seconds (0-59)
DL = hundredths (0-99)


on return:
AL = 00 if time change successful
   = FF if time invalid


- changes DOS maintained clock
- DOS version 3.3+ also update CMOS clock where applicable
```

MSC: void (interrupt far *_dos_getvect (unsigned intnum))()

- prototype in dos.h

- intnum = interrupt to get address of (0..255)
- returns far pointer to interrupt handler routine

- Turbo C uses getvect()
- see INT 21,35 _dos_setvect()

TC: void interrupt (*getvect(int intr_num))()

- prototype in dos.h

- returns the value of the interrupt vector named by intr_num
- returns 4-byte far pointer to current interrupt service routine
  stored in interrupt vector table
- Example:

```
void interrupt (*old_int_1c)();

old_int_1c = getvect( 0x1c );
```

- MS C uses _dos_getvect()
- see INT 21,35

INT 21,35 - Get Interrupt Vector

```
AH = 35h
AL = interrupt vector number


on return:
ES:BX = pointer to interrupt handler


- standard method for retrieving interrupt vectors


- see   INT 21,25
```

INT 21,25 - Set Interrupt Vector

```
AH = 25h
AL = interrupt number
DS:DX = pointer to interrupt handler


returns nothing


- provides a safe method for changing interrupt vectors


- see   INT 21,35
```

MSC: void _dos_setvect(unsigned intnum, void (interrupt far *handler)()) - prototype in dos.h - intnum = interrupt vector to set (0..255) - handler = new interrupt routine - Turbo C uses setvect() - see INT 21,25 _dos_getvect()

TC: void setvect(int intr_num, void interrupt (*isr)())

- prototype in dos.h

- sets the value of interrupt vector named by intr_num (corresponds to 0-255 for MS-DOS) in DOS interrupt vector table to a far pointer to "isr" an interrupt service routine
- address of a C routine may be used only if it has been declared to be an interrupt routine. Ex:

void interrupt func(void);

- MS C uses _dos_setvect()
- see INT 21,25

MSC: void _dos_keep(unsigned rc, unsigned size)

- prototype in dos.h

- rc = return code to pass to parent process
- size = number of paragraphs to remain resident

- Turbo C uses keep()
- see INT 21,31

TC: void keep(unsigned char status, int size)

- prototype in dos.h

- keep() returns to MS-DOS with exit status supplied in status,
  retaining current program resident in memory with size paras and
  rest of memory freed
- MS C uses _dos_keep()
- see INT 21,31

MSC: unsigned _dos_setblock(unsigned size, unsigned seg, unsigned *maxsize) - prototype in dos.h - size = new block size - seg = segment of DOS memory block to change - maxsize = size of buffer actually allocated if size not available - returns 0 on success or DOS error code on failure - Turbo C uses setblock() - see INT 21,4A _dos_allocmem() _dos_freemem()

TC: int setblock(unsigned seg, unsigned newsize)

- prototype in dos.h

- modifies size of previously allocated DOS memory segment
- MSC uses _dos_setblock()
- see INT 21,4A

MSC: unsigned _dos_setdate(struct dosdate_t *date)

- prototype in dos.h

- date = receives date information
- returns 0 on success or DOS error code on failure
- sets DOS date via INT 21,2B
- DOS 3.3+ also sets CMOS clock

- MS C uses _dos_setdate()
- see INT 21,2B _dos_getdate()

INT 21,2B - Set Date

```
AH = 2B
CX = year (1980-2099)
DH = month (1-12)
DL = day (1-31)


on return:
AL = 00 if date change successful
   = FF if invalid date


- sets DOS maintained clock
- DOS versions 3.3+ also update the CMOS date where
applicable
```

MSC: void _dos_setdrive(unsigned drive, unsigned *count)

- prototype in dos.h

- drive = drive to set as default; 1 = A:, 2 = B:, ...
- count = receives simple count of total drives in system
- does not return error if invalid drive is requested, use _dos_getdrive() to see if requested drive was set

- Turbo C uses setdisk()
- see INT 21,E _dos_getdrive()

TC: int setdisk(int drive)

- prototype in dir.h

- set current drive to set as default; 0 = A:, 1 = B:, ...
- returns total number of drives available

- MS C uses _dos_setdrive()
- see INT 21,E

MSC: unsigned _dos_setfileattr(char *path, unsigned attr)

- prototype in dos.h

- path = valid DOS file name
- attr = file attribute (logical OR of the following):
        = _A_ARCH
        = _A_HIDDEN
        = _A_NORMAL
        = _A_RDONLY
        = _A_SUBDIR
        = _A_SYSTEM
        = _A_VOLID

- returns 0 on success or DOS error code on failure

- see INT 21,43 _dos_getfileattr()

MSC: short _ellipse(short ctl,short x1,short y1,short x2,short y2) MSC: short _ellipse_w(short ctl, double wx1, double wy1, double wx2, double wy2) MSC: short far _ellipse_wxy(short ctl, struct _wxycoord far *pwxy1, struct _wxycoord far *pwxy2) - prototype in graph.h - ctl = _GFILLINTERIOR fill ellipse with current fill mask = _GBORDER don't fill ellipse - returns nonzero value if ellipse is drawn ok, zero otherwise - center of ellipse is center of bounding rectangle specified by supplied coordinates - see _setfillmask()

MSC: void far _setfillmask(unsigned char far *mask)

```
        - prototype in graph.h

        - mask =  8 by 8 bit fill mask array
        - sets the fill mask used to fill screen regions
        - any bit set to 1 will have the default color; any bit
cleared
          to zero leaves the pixel unchanged
        - when no fill mask is set (NULL), the default color only
is used
```

TC: void enable(void) MSC: void _enable(void) - prototype in dos.h - enables all interrupts via STI instruction

void _exit(int completioncode)

- prototype in process.h

- terminates program, BUT doesn't close files or write the buffered
  output and NO registered exit functions (via atexit()) are called

int atexit(void func)

- prototype in stdlib.h

- func is defined as void (*func)(void)
- calls func() before exiting to DOS
- up to 32 functions can be called (if 32 atexit calls registered)
- should be located in main & executed before anything else
- returns 0 if OK, else nonzero if no space to register function

MSC: short far _floodfill_w(double wx, double wy, short bcolor) MSC: short far _floodfill (short x, short y, short bcolor) - prototype in graph.h - x, y = coordinates - bcolor = fill boundary color - if (x,y) falls inside a figure, the figure is filled with the with the current fill color; if it falls outside the figure the background is filled - returns nonzero value on success or 0 on failure

void _fpreset(void)

- prototype in float.h

- resets floating point math package, usually used with signal(),
  system(), exec...(), spawn...()
- should be called before using 8087/80287 after using one of the
  above functions
- define identically for Turbo C and MS C

int (*ssignal(int sig, int (*action)))) (void)

- prototype in signal.h

- ssignal() and gsignal() implement a software-signaling facility
  where software signals are integers 1-15
- ssignal() establishes an action routine for servicing a signal
  where sig is associated with the action routine
- ssignal() returns action previously established for ssignal()
  or SIG_DFL
- UNIX based

int gsignal(int sig)

- prototype in signal.h

- ssignal() and gsignal() implement a software-signaling facility
  where software signals are integers 1-15
- gsignal() raises the signal given by sig and executes the action
  routine
- gsignal() returns value by action or SIG_IGN or SIG_DFL
- UNIX based
- see ssignal()

BIOS System Descriptor Table (PS/2)

Offset	Size	Description
00	word	length of descriptor (8 minimum)
02	byte	model byte (same as FFFF:E, not reliable)
03	byte	secondary model byte
04	byte	BIOS revision level (zero based)
05	byte	feature information (see table)
06	dword	reserved

```
|7|6|5|4|3|2|1|0| feature information at offset 5
 | | | | | | | `--- reserved
 | | | | | | `---- 0=PC bus, 1=Micro Channel
 | | | | | `----- Extended BIOS Data Area (EBDA) allocated
 | | | | `------ wait for external event supported
 | | | `------- INT 15,4F used (kbd intercept)
 | | `-------- RTC present
 | `---------- 2nd 8259 present
 `----------- DMA channel 3 used by fixed disk BIOS
```

- see also MACHINE IDENT and INT 15,C0

Exec Function Call

```
int execl(pathname,arg0,arg1,arg2,...,argN,NULL )
int execle(pathname,arg0,arg1,arg2,...,argN,NULL,envp )
int execlp(pathname,arg0,arg1,arg2,...,argN,NULL )
int execlpe(pathname,arg0,arg1,arg2,....,argN,NULL,envp )
    char *pathname,*arg0,*arg1,*arg2,....,*argN,*envp[];

int execv(pathname,arg,NULL )
int execve(pathname,arg,NULL,envp )
int execvp(pathname,arg,NULL )
int execvpe(pathname,arg,NULL,envp )
    char *pathname,*arg[],*envp[];
```

- prototype in process.h

- loads and runs child processes
- pathname search based on MS-DOS search algorithm
  - o if no extension or period - search for exact file name - if not found, add .exe and search again
  - o if extension given, search only for exact file name
  - o if period given, search for file name with no extension

- arg0 can be the same as the pathname
- at least one argument must be passed
- combined argument list cannot exceed 128 bytes
- execl..: arg0, arg1,...,argN passed as SEPARATE arguments
- execv..: arg[0], arg[1],...,arg[N] passed as argument ARRAY
- execlp, execlpe, execvp, execvpe: search for child in PATH
- returns no value if OK,
  returns -1, with errno set to:

```
E2BIG  (too many args)
EACCES (permission denied)
EMFILE (too many open files)
ENOENT (path or file not found)
ENOEXEC (exec format error)
ENOMEM (not enough memory).
```

- if successful, there is no return to the caller; the caller is killed off

exec...

Function spawn(), etc...

```
        int spawnl( int mode, char *path, char *arg0,
                        char *arg1, ..., NULL )
        int spawnle( int mode, char *path, char *arg0,
                        char *arg1, ..., NULL, char *envp[] )
        int spawnlp( int mode, char *path, char *arg0,
                        char *arg1, ..., NULL )
        int spawnlpe( int mode, char *path, char *arg0,
                        char *arg1, ..., NULL, char *envp[] )
        int spawnv( int mode, char *path, char *argv )
        int spawnve( int mode, char *path, char *argv, char *envp
[] )
        int spawnvp( int mode, char *path, char *argv )
        int spawnvpe( int mode, char *path, char *argv, char *envp
[] )

        - prototype in process.h

        - creates & runs child processes
        - mode is P_WAIT which puts parent process "on hold" until
child
          completes execution; P_NOWAIT which continues to run
parent
          while child runs (not supported yet); or P_OVERLAY which
overlays
          child in memory location formerly occupied by parent
(same as
          exec... call)
        - use P_WAIT to perform a DOS shell
        - returns 0 for SUCCESS and non-zero for failure based on
the
          errorlevel set by the application

        - see   exec...
```

MSC: unsigned _freect(size_t size)

```
        - prototype in malloc.h

        - size = size of allocation in bytes
        - returns the count of calls to an allocation function a
program
          can make before failing
        - test is made in default data segment only
```

MSC: short far _getcolor(void)

- prototype in graph.h

- returns the current color number
- default color is the highest valid color in current palette

MSC: struct xycoord far _getcurrentposition(void) MSC: struct _wxycoord far _getcurrentposition_w(void) - prototype in graph.h - returns current position coordinates in struct xycoord format

MSC: unsigned char far *_getfillmask(unsigned char far *mask)

- prototype in graph.h

- mask = receives current fill mask or NULL if mask not present

MSC: void far _getimage(short x1, short y1, short x2, short y2, char huge *image) MSC: void far _getimage_w(double wx1, double wy1, double wx2, double wy2, char huge *image) MSC: void far _getimage_wxy(struct _wxycoord far *pwxy1, struct _wxycoord far *pwxy2,char huge *image) - prototype in graph.h - (x1, y1) upper left coordinates of rectangle - (x2, y2) lower right coordinates of rectangle - image = buffer to receive screen image - Turbo C uses getimage()

TC: void far getimage(int left, int top, int right, int bottom, void far *bitmap) - prototype in graphics.h - copies video data from the screen rectange to buffer "bitmap" - bitmap must be large enough to hold entire buffer plus 4 bytes (2 words) for height and width information. Function imagesize() should be used to determine the size of the buffer to avoid overwriting adjacent memory. - MS C uses _getimage() - see putimage() imagesize()

void far putimage(int left, int top, void far *bitmap, int option)

- prototype in graphics.h

- left and top represent the upper left corner of the area the
  bitmap data will be placed

- op may be one of the following:
  COPY_PUT copy data from bitmap to screen and erasing contents
  XOR_PUT xor data in bitmap with current screen
  OR_PUT   or data in bitmap with current screen
  AND_PUT and data in bitmap with current screen
  NOT_PUT similar to COPY_PUT, but inverse of data

- the getimage()/putimage() buffer has the following format:
  00 width of image (WORD)
  02 height of image (WORD)
  04 actual bitmap pixel data, format and size is related to video mode and dimensions

unsigned far imagesize(int left, int top, int right, int bottom)

- prototype in graphics.h

- returns the size of the buffer required to hold the screen image
  represented by the coordinates
- returns size or 0xFFFF on error (image >= 64K-1)

MSC: unsigned short far _getlinestyle(void)

- prototype in graph.h

- returns current line style mask
- each 1 bit represents a pixel (in current color) in the line
  mask; each 0 represent a pixel that is left alone

- see _setlinestyle()

MSC: void far _setlinestyle(unsigned short mask)

- prototype in graph.h

- each 1 bit represents a pixel (in current color) in the line
  mask; each 0 represent a pixel that is left alone
- default line style is FFFFh

- see _getlinestyle()

struct xycoord far _getlogcoord(short x, short y)

- prototype in graph.h

- translates physical coordinates to logical coordinates returning
  the result in the format:

```
struct xycoord
    {
    short xcoord;
    short ycoord;
    };
```

- _getviewcoord() is obsolete; use _getlogcoord()
- see _getphyscoord()

MSC: struct xycoord far _getphyscoord(short x, short y)

- prototype in graph.h

- translates logical coordinates to physical coordinates returning
  the result in the format:

```
struct xycoord
    {
    short xcoord;
    short ycoord;
    };
```

- see _getlogcoord()

MSC: short far _getpixel(short x, short y) MSC: short far _getpixel_w(double wx, double wy) - prototype in graph.h - returns pixel value on success or -1 on failure - Turbo C uses getpixel()

MSC: short far _gettextcolor(void)

- prototype in graph.h
- returns current text color value

MSC: struct rccoord far _gettextposition(void)

```
- prototype in graph.h

- returns current text position via rccoord structure:

struct rccoord
    {
    short  row;
    short col;
    };
```

MSC: struct videoconfig far *_getvideoconfig(struct videoconfig far *config) - prototype in graph.h - returns video configuration information via struct videoconfig: struct videoconfig { short numxpixels; - pixels on X axis short numypixels; - pixels on Y axis short numtextcols; - text columns available short numtextrows; - text rows available short numcolors; - actual colors available short bitsperpixel; - bits per pixel short numvideopages; - available video page count short mode; - current video mode short adapter; - active display adapter short monitor; - active display monitor short memory; - adapter video memory in K };

MSC: void _hardresume(int result) MSC: void _hardretn(int error) MSC: void _harderr (void (far *handler)()) - prototype in dos.h - result = return value from handler - error = number of error - _harderr() registers a user critical error handler with it's own error handler which is to be called during a critical error - handler = new INT 24 handler with the format: handler(unsigned deverr,unsigned errcode,unsigned far *devhdr) where: deverr = device error code (AX value DOS passes to INT 24) errcode = error code (DI value DOS passes to INT 21) devhdr = pointer to device header on which error occurred - handler must return via one of three methods: simple return returns to DOS error handler _hardresume() returns to DOS error handler _hardretn() returns to the application - hardresume() returns one of the following: _HARDERR_ABORT DOS should abort the process via INT 23 _HARDERR_FAIL DOS should fail the call (DOS 3.x+) _HARDERR_IGNORE DOS should ignore the error _HARDERR_RETRY DOS should retry the operation - due to the complexity of these functions consult the vendor documentation for more specific details - Turbo C uses hardretn(), harderr() and hardresume() - see INT 24

TC: void hardretn(int errcode)

- prototype in dos.h

- hardretn() may be called to return directly to the application
- MS C uses _hardretn
- see harderr()

TC: void harderr(int (*fptr)())

- prototype in dos.h

- harderr() establishes a hardware error handler for current program, invoked wherever interrupt 0x24 occurs
- function fptr is called when such an interrupt occurs
- handler function will be called with the following arguments= handler(int errval, int ax, int bp, int si) where errval is error code in DI register by MS-DOS, and ax, bp, si are values MS-DOS has in AX, BP, and SI regs
- ax indicates if disk or other device error occurs; if ax is not negative, then disk error, else device error. For disk error, ax ANDed with 0x00ff will give bad drive number
- bp and si together point to device driver header
- hardresume() may be called with rescode to return to MS-DOS, where rescode is

  2 - for abort
  1 - retry
  0 - ignore

- hardrtn() may be called to return directly to the application
- handler must return:

  0 - ignore
  1 - retry
  2 - abort

- handler may issue DOS calls 1 through 0xC, but no others, and no C standard I/O or UNIX I/O calls may be used
- MS C uses _harderr()

- see hardresume() hardretn() INT 24

TC: void hardresume(int rescode)

- prototype in dos.h

- hardresume() may be called with rescode to return to DOS, where rescode is:

      2 - abort
      1 - retry
      0 - ignore

- MS C uses _hardresume()
- see harderr()

MSC: short far _lineto(short x, short y) MSC: short far _lineto_w(double wx, double wy) - prototype in graph.h - draws a line from the current graphics position up to and including the point specified - current graphics position is then updated to (x,y) - returns nonzero if success; zero otherwise - Turbo C uses lineto()

MSC: unsigned long _lrotl(unsigned long value, int n)

MSC: unsigned long _lrotr(unsigned long value, int n)

- prototype in stdlib.h

- rotates long value by 'n' bits left or right
- returns rotated value

- see _rotl() _rotr()

MSC: unsigned _rotl(unsigned value, int shift)

MSC: unsigned _rotr(unsigned value, int shift)

- prototype in stdlib.h

- rotates value by 'n' bits left or right
- returns rotated value

- see _lrotl() _lrotr()

MSC: void _makepath(char *path, char *drive, char *dir, char *fname, char *ext) - prototype in stdlib.h - creates fully qualified filename from parts - path = receives fully qualified filename created from the parts - drive = drive letter string, with or without the colon - dir = directory name; both slashes '/' or '\' are allowed; trailing slash is optional - fname = base file name w/o extension (max 8 bytes) - ext = file extension (max 3 bytes) - see _splitpath()

MSC: void _splitpath(char *path, char *drive, char *dir, char *fname, char *ext) - prototype in stdlib.h - dissects the path into components - path = fully qualified file name to dissect - drive = is the drive letter followed by a colon if drive was specified in string - dir = any directories included in "path" plus '\' - fname = base filename w/o extension - ext = file extension - the maximum size of each component is: _MAX_DRIVE, _MAX_DIR, _MAX_FNAME, _MAX_EXT (all include the NULL and are actually much larger than needed, except _MAX_DRIVE is exact fit) - see _makepath ()

double _matherr(_mexcep why, char *fun, double *arg1p, double *arg2p, double retval) - prototype in math.h - used with matherr(), calling matherr() and processes the return value from matherr() - floating point error handling routine

int matherr(struct exception *e)

- prototype in math.h
- provided as customizable math error-handling routine

MSC: size_t _memavl(void)

- prototype in malloc.h

- returns size in bytes of memory available in the default data segment

MSC: struct xycoord far _moveto(short x, short y) MSC: struct _wxycoord far _moveto_w (double wx, double wy) - prototype in graph.h - moves current drawing position to specified coordinates - returns previous coordinates in the following structures: struct xycoord struct _wxycoord { { short xcoord; double wx; - window x coord short ycoord; double wy; - window y coord }; }; - see _lineto()

MSC: void far _outtext(unsigned char far *text)

- prototype in graph.h
- writes string in "text' to the graphics display

MSC: short _pie(short control, short x1, short y1, short x2, short y2, short x3, short y3, short x4, short y4) MSC: short far _pie_wxy(short ctl, struct _wxycoord far *pwxy1, struct _wxycoord far *pwxy2, struct _wxycoord far *pwxy3, struct _wxycoord far *pwxy4) - prototype in graph.h - (x1, y1) upper left rectangle corner - (x2, y2) lower right rectangle corner - (x3, y3) start of vector - (x4, y4) end of vector - ctl = _GFILLINTERIOR fills region with current color and mask _GBORDER don't fill region - returns nonzero value on success; zero otherwise

MSC: void _putimage(short x, short y, char huge *image, short action) MSC: void far _putimage_w(double wx, double wy, char huge *image, short action) - prototype in graph.h - restores previously saved screen image to screen - (x, y) upper left corner of image - image = buffer containing previously saved image - action = defines interaction between current screen buffer and the data stored in "image" = _GAND AND image with current screen = _GOR OR image with current screen = _GPRESET overwrite current screen with inverse of image = _GPSET overwrite current screen with exact image = _GXOR XOR's image to current screen - Turbo C uses putimage() - see _getimage() _imagesize()

MSC: short _rectangle(short ctl, short x1, short y1, short x2, short y2) MSC: short far _rectangle_w(short control, double wx1, double wy1, double wx2, double wy2) MSC: short far _rectangle_wxy(short control, struct _wxycoord far *pwxy1, struct _wxycoord far *pwxy2) - prototype in graph.h - (x1,y1) upper left corner of rectangle - (x2,y2) lower right corner of rectangle - ctl = _GFILLINTERIOR fill rect. with current color and mask = _GBORDER don't fill rectangle - returns nonzero on success; zero otherwise

MSC: short far _remapallpalette(long far *colors) MSC: long far _remappalette(short pixnum, long color) - prototype in graph.h - colors = color number array containing one color per video mode color - pixnum = pixel to change - color = new color number **Valid colors are:** _BLACK _BLUE _BRIGHTWHITE _BROWN _CYAN _GRAY _GREEN _LIGHTBLUE _LIGHTCYAN _LIGHTGREEN _LIGHTMAGENTA _LIGHTRED _YELLOW _MAGENTA _RED _WHITE - _remapallpalette() remaps all pixel values; _remappalette() affects on the pixel specified by pixnum - _remapallpalette() returns -1 on success; zero otherwise - _remappalette() previous color value of "pixnum" on success; -1 otherwise

MSC: void _searchenv(char *fname, char *var, char *path)

- prototype in stdlib.h

- searches for "fname" in environment variable "var"
- fname = name of file to search for
- var = environment variable to use for search
- path = receives path if file found
- attempts to locate filename using MS-DOS file search strategy

- Turbo C uses searchpath()

TC: char *searchpath(const char *filename)

- prototype in dir.h

- filename = name of file to search for
- attempts to locate filename using MS-DOS path using standard
  DOS file search strategy
- returns pointer to full pathname for filename (in a static
  array) or NULL if not found
- MS C uses _searchenv()

MSC: short far _selectpalette(short palette)

- prototype in graph.h

- palette = palette number
- works only in MRES4COLOR and MRESNOCOLOR video modes
- returns value of previous palette

MSC: short far _setactivepage(short page)

- prototype in graph.h

- page = memory page where graphics output will be written
- if successful returns page number of former active page or negative value on failure
- swapping video pages is a good method of handling animation

MSC: long far _setbkcolor(long color)

- prototype in graph.h

- color = new color for background
- returns previous background color

MSC: short far _setcolor(short color)

- prototype in graph.h

- color = new color value
- returns previous color index or -1 on error
- all subsequent graphics calls will use the specified color

MSC: short far _setfont(unsigned char far *options)

- prototype in graph.h

- returns font with matching "options"
  returns negative value on error
  -1 Font not registered
  -4 not enough memory for font

struct xycoord far _setlogorg(short x, short y)

- prototype in graph.h

- moves the logical origin (0,0) to the physical point (x,y)
- returns previous logical origin in physical coordinates

to:

```
struct xycoord
    {
    short xcoord;
    short ycoord;
    };
```

MSC: short far _setpixel(short x, short y) MSC: short far _setpixel_w(double wx, double wy) - prototype in graph.h - returns previous value of pixel or -1 on error - Turbo C uses setpixel() - see _getpixel()

MSC: short far _settextcolor(short color)

- prototype in graph.h

- sets color of graphics text to color
- default color is the highest valid color
- returns previous color

- see _gettextcolor()

MSC: struct rccoord far _settextposition(short row, short column) - prototype in graph.h - sets the current text position to that specified - returns previous text position in rccoord structure - see _gettextposition()

MSC: void far _settextwindow(short r1,short c1, short r2,short c2)

```
        - prototype in graph.h

        - sets graphics text output window;  text scrolls in window
when
          the window becomes full
        - (r1,c1) upper left corner of window
        - (r2,c2) lower right corner of window
```

MSC: short far _setvideomode(short mode)

- prototype in graph.h

- sets video mode specified
- mode= _DEFAULTMODE hardware default
  _MAXCOLORMODE graphics mode with most colors
  _MAXRESMODE graphics mode with highest resolution
  _TEXTBW40 40x25 16 color CGA/Text
  _TEXTC40 40x25 16 color CGA/Text
  _TEXTBW80 80x25 16 shades CGA/Text
  _TEXTC80 80x25 16 color CGA/Text
  _MRES4COLOR 320x200 4 color CGA/Graphics
  _MRESNOCOLOR 320x200 4 color CGA/Graphics
  _HRESBW 640x200 2 color CGA/Graphics
  _TEXTMONO 80x25 mono MDA/Text
  _MRES16COLOR 320x200 16 color EGA/Graphics
  _HRES16COLOR 640x200 16 color EGA/Graphics
  _ERESNOCOLOR 640x350 1 color EGA/Text
  _ERESCOLOR 640x350 64 color EGA/Graphics
  _VRES2COLOR 640x480 2 color VGA/Graphics
  _VRES16COLOR 640x480 16 color VGA/Graphics
  _MRES256COLOR 320x200 256 color VGA/Graphics
  _ORESCOLOR 640x400 16 color CGA/Graphics/Olivetti
  _HERCMONO 720x348 2 color HGC/Graphics/Hercules

- returns non-zero on success; zero on error

MSC: void far _setviewport(short x1, short y1, short x2, short y2)

- prototype in graph.h

- defines a clipping region like _setcliprgn()
- sets the logical origin to the upper left corner coordinates
  specified as parameters
- (x1,y1) upper left corner of window
- (x2,y2) lower right corner of window

MSC: short far _setvisualpage(short page)

- prototype in graph.h

- sets visual page to that specified
- default page is zero
- returns previous page number or a negative value on error

MSC: char *_strdate(char *date) MSC: char *_strtime(char *time) - prototype in time.h - date = formatted date string: mm/dd/yy (9 bytes inc NULL) - time = formatted time string: hh:mm:ss (9 bytes inc NULL) - returns pointer to the parameter string - Turbo C uses strdate() and strtime()

MSC: char *_strerror(char *string)

- prototype in string.h

- creates an error message based on the user supplied string followed by a colon, then the library error message
- string = user error message string
- returns pointer to the generated string
- does not print the string

int _tolower(int c)

- prototype in ctype.h

- converts c to lowercase if c is UPPER case
- use tolower() preferentially

int tolower(int c)

- prototype in ctype.h
- converts c to LOWER case only if c is UPPER case

int _toupper(int c)

- prototype in ctype.h

- converts c to UPPER case only if c is LOWER case
- use toupper() preferentially

int toupper(int c)

- prototype in ctype.h
- converts c to UPPER case only if c is LOWER case

MSC: short far _wrapon(short wrap)

- prototype in graph.h

- sets wrapping/truncating of text
- wrap = _GWRAPOFF truncates text at window border
        = _GWRAPON wraps line at window border

- returns previous wrap value

void abort(void)

- prototype in stdlib.h & process.h
- terminates a process & calls exit(3)

int abs(int n)

- prototype in stdlib.h
- returns the absolute value of int n

TC: int absread(int drive, int count, int start, void *buffer)

- prototype in dos.h

- reads absolute logical disk sectors
- drive = (0=A:, 1=B:, 2=C:, ...)
- count = # of sectors to read
- start = beginning logical sector number
- buffer = address of buffer
- TC++ accounts for DOS 4.x+ long sector values while TC 2.0 and
  earlier do not (see INT 25)
- returns 0 if OK; -1 on error; errno has DOS error number

TC: int abswrite(int drive, int sectors, int start, void *buffer)

- prototype in dos.h

- writes absolute logical disk sectors
- drive = (0=A:, 1=B:, 2=C:, ...)
- sectors = # of sectors to write
- start = beginning logical sector number
- buffer = address of buffer
- TC++ accounts for DOS 4.x+ long sector values while TC 2.0 and
  earlier do not (see INT 25)
- returns 0 if OK; -1 on error; errno has DOS error number

int access(const char *filename, int amode)

```
- prototype in io.h

- checks filename for existence & R/W access

- amode = 06 -- check for R/W permission
          04 -- check for R
          02 -- check for W
          01 -- execute (ignored)
          00 -- check if filename exists

- filename can be directory
- returns 0 if OK
         -1 on error
            errno = ENOENT if path or filename not found
            errno = EACCES if permission denied
```

double acos(double x)

- prototype in math.h

- returns arc cosine of x
- values must be in range -1 to 1

MSC: void *alloca(size_t size)

- prototype in malloc.h

- allocates size bytes for the application stack
- memory is automatically freed upon exiting the calling function
- this function cannot be used as a parameter to another function
  or stack corruption will occur
- the pointer returned CANNOT be passed as an argument to free()
  since it isn't allocated from DOS
- see stackavail

MSC: size_t stackavail(void)

- prototype in malloc.h

- used to determine approximate memory available on the stack for
  allocation via alloca()
- returns approximate memory available on the stack

TC: int allocmem(unsigned size, unsigned *seg)

- prototype in dos.h

- allocates memory in size paragraphs, where seg is pointer to
  a word containing segment address
- returns -1 if ok, else largest available memory in paragraphs
- MS C uses _dos_allocmem
- see INT 21,48

char *asctime(const struct tm *tm)

- prototype in time.h

- converts time in struct tm to 26-character ASCII string
- returns a pointer to the ASCII string

double asin(double x)

- prototype in math.h

- returns arc sine of x
- values must be in range -1 to 1

void assert(int test)

- prototype in assert.h

- if test fails, prints error message & aborts program
- test can be a condition

double atan(double x)

- prototype in math.h
- returns arc tangent of x

double tan(double y, double x)

- prototype in math.h

- returns arc tangent of x*2
- used instead of atan when the resulting angle is near pi/2 or -pi/2

double atof(const char *s)

- prototype in math.h & stdlib.h
- converts ASCII string to double value

int atoi(const char *s)

- prototype in stdlib.h
- converts ASCII string to integer value

long atol(const char *s)

- prototype in stdlib.h
- converts ASCII string to long value

int bdos(int dosfun, unsigned dosdx, unsigned dosal)

- prototype in dos.h

- make DOS system call for function dosfun, passing DX and AL
  values via dosdx and dosal
- use only in small memory model
- unique to DOS
- returns value of AX set by system call

TC: int bdosptr(int dosfun, void *argument, unsigned dosal)

- prototype in dos.h

- make DOS system call for function dosfun, passing POINTER to values via argument and passing AL via dosal
- use for small or large memory models
- returns value of AX set by system call, else -1 on failure (where errno & _doserrno are then set)

int brk(void *endds)

- prototype in alloc.h (malloc.h for MS C)

- dynamically changes the amount of space allocated to the calling
  programs data segment. Amount of allocated space increases as
  the break value increases.
- returns 0 if OK
  -1 with errno set to ENOMEM

void *bsearch(void *key, void *base, int nelem, int width, int (*fcmp)()) - prototype in stdlib.h - does binary search for items in a table which is ALREADY sorted in ascending order - base points to 0th element of table - nelem contains number of entries in table - width contains number of bytes in each entry - key points to the search key - fcmp() points to comparison routine, where key and elem are passed to it as pointers. Function fcmp() returns: integer < 0 if search key < elem; integer = 0 if equal; returns > 0 if search key > elem - returns 0 if no match found, else address of first entry that matches

Complex C Declarations

```
        int i;                  i as an int
        int *i;                 i as a pointer to an int
        int **i;                i is a pointer to a pointer to an
int
        int *(*i)();            i is a pointer to a function
returning a
                                  pointer to int
        int *(*i[])();          i is an array of pointers to
functions
                                  returning pointers to an int
        int *i[5];              i is an array of 5 pointers to int
        int (*i)[5];            i is a pointer to an array of 5 ints
        int *i();               i is a function returning a pointer
to an int
        int (*i)();             i is a pointer to a function
returning int
        int *(*(*i)())[5]       i is a pointer to a function
returning a
                                  pointer to an array of 5 pointers
to an int
```

Microsoft C Compiler Fatal Errors

```
        C1000   unknown fatal error; contact Microsoft
        C1001   internal compiler error; contact Microsoft
        C1002   compiler out of heap space in C2.EXE
        C1003   error count exceeded; stopping compilation
        C1004   unexpected end-of-file found; disk full or
unmatched #if
        C1005   string too big for buffer
        C1006   compiler intermediate file create/write error
        C1007   invalid option on compiler command line
        C1008   no source file specified
        C1009   macros nested too deeply
        C1010   macro expansion exceeded available space
        C1011   macro definition too large
        C1012   unmatched parenthesis in preprocessor directive
        C1014   nesting of #include's exceed 10-level nesting limit
        C1016   #ifdef & #ifndef directives expect an identifier
        C1018   unexpected #elif;  #elif without #if directive
        C1019   unexpected #else;  #else without #if directive
        C1020   unexpected #endif;  #endif without #if directive
        C1021   invalid preprocessor command
        C1022   expected #endif;  #if directive not terminated by
#endif
        C1023   can't open specified source file
        C1024   can't open specified include file
        C1026   parser stack overflow; module too complex
        C1027   DGROUP default data allocation exceeds 64K
        C1028   modules far data segment allocation exceeds 64K (QC)
        C1031   function calls nested too deeply
        C1032   can't open object listing file
        C1033   can't open assembly language output file
        C1035   expression too complex; must be simplified
        C1036   can't open source listing file
        C1037   can't open object file
        C1039   unrecoverable heap overflow in Pass 3 (post-
optimizer)
        C1040   unexpected EOF in source file; file disappeared
        C1041   can't open intermediate file; no free file handles
        C1042   can't open intermediate file; invalid TMP
environment var
        C1043   can't open intermediate file; unknown error
        C1044   no disk space available for intermediate file
```

```
        C1047   compiler option was specified too many times
        C1048   unknown option specified
        C1049   invalid numerical argument
        C1050   code segment too large; within 36 bytes of 64K
        C1052   #if/#ifdef nested too deeply;  max of 32 levels
        C1053   struct/union nested too deeply; max of 15
        C1054   initializers nested too deeply; max from 10 to 15
levels
        C1055   out of keys; file has too many symbols
        C1056   out of macro expansion space; macro to large/complex
        C1057   unexpected EOF in macro expansion or missing ')'
        C1059   compiler has run out of near heap space
        C1060   compiler has run out of far heap space
        C1062   error writing preprocessor output file for /P option
        C1063   compiler stack overflow; module too complex
        C1064   identifier too long causing token to overflow
internal buffer
        C1068   can't open file specified file
        C1069   file write error on specified file; disk space low
        C1070   mismatched #if/#endif pair; missing #endif
        C1071   unexpected EOF found in comment; missing end of
comment
        C1072   can't read indicated file
        C1090   data allocation exceeds 64K on _based allocation
        C1015   can't open include file (check name, searchpath and
FILES=)
        C1126   automatic allocation (local variable) exceeds size
        C1127   segment redefinition; overwritten by specified
segment
```

Microsoft C Compiler Nonfatal Errors

```
        C2015   too many characters in constant; escape sequence
too large
        C2016   missing closing single quotation mark on code line
        C2017   illegal escape sequence; occurred outside of string
        C2018   unknown character found in source file
        C2019   expected preprocessor directive; # followed by non-
directive
        C2021   expected exponent value but found non-numeric
character
        C2022   octal value following '\' is too large for a single
character
```

C2025 given identifier redefines enum, struct or union tag
C2026 given identifier already used as enum constant
C2027 given identifier refers to undefined enum, struct or union
C2028 struct/union member needs to be inside a struct/union
C2030 identifier used more than once in struct or union tag
C2031 function can't be use as struct or union member
C2033 bit field can't be declared as a pointer or use indirection
C2034 type of host bit field is too small for number of bits
C2035 given structure or union has undefined size
C2037 undefined struct or union identifier on left of -> or .
C2038 identifier is not a struct or union member
C2055 expected formal-parameter list instead of argument-type list
C2056 illegal expression; possibly from previous unreported problem
C2057 expected constant expression
C2058 expected integral constant expression
C2059 invalid token caused a syntax error
C2060 syntax error; EOF found before expected token
C2061 identifier caused syntax error
C2062 unexpected type
C2063 identifier is not a function but used as such
C2064 term does not evaluate to a function pointer
C2065 identifier not defined
C2066 illegal cast to function type
C2067 illegal cast to array type
C2068 illegal cast type used in expression
C2069 cast of void term to non-void is invalid
C2070 illegal sizeof operand; must be expression or type name
C2071 illegal storage class for identifier
C2072 attempt to initialize a function identifier
C2092 arrays of functions illegal; use array of ptrs to function
C2093 can't use address of auto variable as static initializer
C2094 label was undefined in function
C2095 void argument cannot be passed to function; (void *) may
C2096 struct & union comparison is illegal; compare

```
members
        C2097   illegal initialization
        C2098   expected address as initialization expression
        C2099   non-constant initializer
        C2100   illegal indirection; * applied to a non-pointer
value
        C2101   address of operator '&' used on constant; requires
lvalue
        C2102   address of operator '&' requires lvalue
        C2103   address of operator '&' can't be used on a register
variable
        C2104   address of operator '&' used on bit field not
allowed
        C2105   operator needs lvalue
        C2106   left side of an operation must be lvalue
        C2107   subscript applied to expression didn't evaluate to
pointer
        C2108   non-integral expression used as array subscript
        C2109   subscript used on non-array variable
        C2127   allocation for function parameters exceeds 32K
        C2128   array crosses 2 segments & element size not power
of 2
        C2129   static function referenced was never defined
        C2130   #line expected string containing file name; found
other
        C2131   more than one memory attribute applied to and
identifier
        C2132   syntax error : unexpected identifier
        C2133   attempt to declare an unsized array as a local
variable
        C2134   struct or union too large; exceeded 64k
        C2136   prototype must have a valid type for each variable
        C2137   use of empty character constant '' is illegal
        C2139   more than one type used in a variable declaration
        C2140   argument can't be a function
        C2141   enum constant value out of int range
        C2143   expected 'token1' before 'token2' or missing ')',
'}' or ';'
        C2144   expected 'token' before 'type' or missing ')', '}'
or ';'
        C2145   expected 'token' before identifier or declaration
missing ';'
        C2146   expected 'token' before identifier
        C2147   increment of index or array pointer with unknown
base type
        C2162   token following stringizing operator # isn't formal
```

parameter
C2163 function in pragma not available as intrinsic function
C2164 intrinsic function not declared before use wit /Oi option
C2165 _pascal, _cdecl, _???? keywords can't modify pointers to data
C2166 attempt to modify item declared as const; lvalue is constant
C2167 too many parameters in call to an intrinsic function
C2168 too few parameters in call to an intrinsic function
C2169 function definition for function already declared intrinsic
C2170 intrinsic pragma used for function without intrinsic form
C2171 unary operator used on illegal operand type
C2172 non-pointer argument passed to a function expecting pointer
C2173 non-pointer argument passed to a function expecting pointer
C2174 cant pass parameter with void type to function
C2176 static huge data not supported by /qc; use halloc()
C2177 constant too large for data type
C2178 storage class for same_seg pragma variables must be extern
C2179 same_seg pragma variable class has changed from extern
C2207 middle member of struct/union has zero-sized array
C2208 enum, struct or union defined w/o members using /Za option
C2209 type cast used in _based construct must be (_segment)
C2210 base in _based declarator must be near/far data pointer
C2211 item cast in _based declarator can't be a function
C2212 _based not available for functions or pointers to functions
C2213 symbol used as base must be type _segment, near far pointer
C2214 _based pointer based on void can't be de-referenced; use :>
C2215 :> operator only for objects based on void
C2216 given function attributes are incompatible
C2217 function attribute requires another attributes also
C2218 type in _based construct must be void
C2219 const/volatile appeared where type or qualifier not

allowed
C2220 no object file generated; warning treated as error due to /WX
C2221 left operand of '.' is pointer to struct/union, must use '->'
C2222 left operand of '->' is struct or union, must use '.'
C2223 left operand of '->' must be pointer struct or union
C2411 identifier is not a member of the specified struct or union
C2412 identifier redefined within current function
C2413 alignment size used with ALIGN directive missing or invalid
C2414 illegal number of operands in assembly code; see /G1 & /G2
C2415 improper operand type
C2416 illegal opcode for processor in assembly code; see /G1 & /G2
C2417 divisor used within the given context is zero
C2418 in-line asm identifier referenced as register and isn't
C2419 divisor argument to mod is zero in given context
C2420 given identifier is illegal in this context
C2421 PTR must not be used with register operand in this context
C2422 illegal segment override used in given context
C2424 given token used to form improper expression in this context
C2425 token used fto form non-constant expression in this context
C2426 given token is illegal operator in this context
C2427 jump referencing label is out of range
C2429 FAR PTR can't be use on jmp or call to label

Microsoft C Command-Line Errors

D2000 unknown CL command line error; contact Microsoft
D2001 too many symbols predefined with /D; max is 30
D2002 memory-model specification conflict; only 1 allowed
D2003 missing source file name for CL command
D2008 option specified too many times
D2011 more than one /FP option specified; only 1 allowed
D2012 too many linker flags in command; max of 128 flags/

```
files
        D2013   incomplete model specification in /A:
        D2018   can't open linker response file
        D2019   can't overwrite source with object file; output =
input
        D2020   /Gc requires extended keywords be enabled via /Ze
        D2021   invalid numerical argument; numeric args must be <
65534
        D2022   can't open help file for /HELP
        D2027   CL could not execute one of the compiler components
        D2028   too many open files; can't dup/redirect specified
stream
        D2030   internal compiler component error; contact Microsoft
        D2031   too many flags/files in command; max of 128
arguments to CL
```

Microsoft C Command-Line Warning

```
        D4000   unknown command line warning in CL; contact
Microsoft
        D4001   listing overrides assembly output; /Fc and /Fa used
together
        D4002   unknown flag ignored by CL command
        D4003   multiple processors selected for code generation
        D4005   CL could not locate compiler component
        D4007   /C must be used in conjunction with /P or /E or /EP
        D4009   threshold valid for far/huge data models only; /Gt
ignored
        D4011   preprocessor listing specified; source listing not
generated
        D4012   function prototyping specified; source listing not
generated
        D4013   combined listing /Fc overrides object listing /Fl
        D4014   invalid value for option; default value used
        D4018   .DEF files supported in OS/2 only (/Lr /Lc)
        D4019   string too long for /ND, /NT, /NM, /St, /Ss option
of CL
```

Microsoft Link Errors (incomplete list)

```
        L2002   fixup overflow (probably incompatible memory
models .OBJs)
```

Microsoft C Floating-Point Math Errors

```
        M6101   invalid operation occurred; usually NAN or infinity
problem
        M6102   denormal; significance loss with very small
generated number
        M6103   attempt to divide by zero in floating-point
operation
        M6104   overflow in floating-point operation
        M6105   underflow occurred in floating-point operation
        M6106   inexact; loss of precision occurred in FP operation
        M6107   unemulated/unsupported 8087/287/387 instruction
executed
        M6108   square root operand somehow became negative; C
won't allow
        M6110   coprocessor or emulator stack overflow
        M6111   stack underflow due to unretrieved double return
values
```

Microsoft C Run-Time Errors

```
        R6000   stack overflow; stack needs enlarged or problem
program
        R6001   null pointer assignment, NULL segment modified
        R6002   floating-point support not linked into executable
        R6005   not enough memory to load child process (EXEC)
        R6006   target executable is not a valid executable (EXEC)
        R6007   invalid environment on exec of child process (EXEC)
        R6008   not enough space for argument vector after program
loads
        R6009   not enough space for environment after program loads
        R6010   abnormal program termination
        R6012   illegal null near-pointer use
        R6013   illegal out of range far-pointer use
        R6016   not enough space for thread data
        R6017   unexpected multithread lock error
```

C Escape Sequences

\a	Bell
\b	Backspace
\f	Form feed
\n	Line feed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quote
\"	Double quote
\\	Backslash
\0	NULL character
\7	Bell
\ddd	Octal value
\xddd	Hexadecimal value

C Operators

Operation	C Operator	Associativity
Array	[]	(--)
Function	()	(--)
Member	->	(--)
Member	.	(--)
Unary minus	-	(--)
Unary plus	+	(--)
Logical NOT	!	(--)
Bitwise compliment	~	(--)
Address of	&	(--)
Indirection	*	(--)
Sizeof	sizeof	(--)
Increment	++	(--)
Decrement	--	(--)
Cast	(type)	(--)
Multiplication	*	(--)
Division	/	(--)
Modulus	%	(--)
Addition	+	(--)
Subtraction	-	(--)
Shift right	>>	(--)
Shift left	<<	(--)
Greater than	>	(--)
Greater or equal	>=	(--)
Less than	<	(--)
Less or equal	<=	(--)
Equal	==	(--)
Not equal	!=	(--)
Bitwise AND	&	(--)
Bitwise XOR	^	(--)
Bitwise OR	|	(--)

Logical AND	&&	(--)
Logical OR	||	(--)
Conditional	?:	(--)
Simple assignment	=	(--)
Addition assignment	+=	(--)
Subtraction assignment	-=	(--)
Multiplication assign	*=	(--)
Division assignment	\=	(--)
Modulus assignment	%=	(--)
Left shift assignment	<<=	(--)
Right shift assignment	>>=	(--)
Bitwise XOR assignment	^=	(--)
Bitwise AND assignment	&=	(--)
Bitwise OR assignment	|=	(--)
Compliment assignment	~=	(--)
Sequence operator	,	(--)

- all operations grouped together have the same precedence

double cabs(struct complex z)

- prototype in math.h
- returns absolute value of int n

void *calloc(unsigned count, unsigned sizeeach)

- prototype in stdlib.h & alloc.h (malloc.h for MS C)

- allocates main memory
- see malloc()

double ceil(double x)

- prototype in math.h
- rounds up x

char *cgets(char *s)

- prototype in conio.h

- reads string from console
- CR/LF combination is replaced by \0
- before calling, s[0] should be the max length of string to input
- on return, s[1] contains the actual string length
- returns pointer to s[2], the actual data input

int chdir(const char *path)

- prototype in dir.h

- changes current working directory to path.
- returns 0 if OK; else -1 with errno set to ENOENT

int chmod(const char *filename, int permiss)

```
        - prototype in io.h, also include stat.h

        - changes access mode of file according to mask given by
permiss,
          where permiss is:

                        S_IWRITE
                        S_IREAD
                        S_IWRITE | S_IREAD

        - returns 0 if OK
                -1 with errno set to ENOENT or EACCES
```

Microsoft C Options (v6.0)

CL [options] [filenames] [libraries link-options]

```
        /AT       Tiny Memory Model; combined code and data limited
to 64K
        /AS       Small Memory Model; code and data limited to 64K
each
        /AM       Medium Memory Model; data is limited to 64K
        /AC       Compact Memory Model; code is limited to 64K
        /AL       Large Memory Model; unlimited code and data but
arrays
                  are limited to 64K
        /AH       Huge Memory Model; unlimited code, data and array
size
        /Astring  where string is compose of the following:

                  Code Pointer Control
                  s     all code pointers are near
                  l     all code pointers are far

                  Data Pointer Control
                  n     all data pointers are near
                  f     all data pointer are far
                  h     all data pointers are huge

                  Segment Setup Control
                  d     stack segment (SS) defaults to data segment
(DS)
                  u     stack segment (SS) is not equal to data
segment (DS);
                        DS is loaded on function entry
                  w     stack segment (SS) is not equal to data
segment (DS);
                        DS is NOT loaded on function entry

        /B1[path] used to invoke the large version of the compiler
C1L.EXE;
                  normally used when the compiler runs out of heap
space
        /B2       invokes large compiler version pass 2
        /B3       invokes large compiler version pass 3
        /C        preserves file comments when preprocessing a file
```

```
        /c        do not link, create .OBJ file only
        /D defines string as a preprocessor constant.  If string is
                  of the form "ID=xxx" then "ID" represents the
value "xxx"
                  If no equals sign is present, "ID" has the value
0.      If
                  an equals sign is present without a value, ID has
the
                  value 1.
        /E        output from preprocessor is written to stdout with
                  interspersed #line directives.
        /EP       output from preprocessor is written to stdout
without,
                  adding #line directives.
        /F  sets stack to  bytes.  Size must be hexadecimal.
        /Fa[file] generate assembly listing.  Filename defaults to
source
                  file with .ASM extension.
        /Fb creates bound executable.  Should be used only with /Lp.
        /Fc[file] generates combined assembly/source listing.   The
filename
                  defaults to the source file name with the .COD
extension.
        /Fe creates the executable with specified name
        /Fl[file] generates object code listing.   Default file
name is the
                  source file name with a .COD extension.
        /Fm[file] generate a link MAP file.  Default file name is
the source
                  file name with the extension .MAP.
        /Fo generate and object file using the name .
        /FPa      generates floating point calls and selects the
alternate
                  math library.
        /FPc      generates floating point calls and selects the
emulation
                  library.
        /FPc87    generates floating point calls and selects the
80x87
                  library.
        /FPi      generates inline 80x87 instructions and selects
the 80x87
                  library.  Default floating point option.
        /FPi87    generates floating point calls and selects the
80x87
                  library.
```

```
        /Fr[file] generates a standard PWB Source Browser
Database.  The
                  default name is the source file with the .SBR
extension.
        /FR[file] generates a standard PWB Source Browser
Database.  The
                  default name is the source file with the .SBR
extension.
        /Fs       generate a source file listing with .LST
extension.
        /Fx       generate a cross reference listing with .CRF
extension.
        /G0       generate code using 8088/8086 instruction set
        /G1       generate code using 80188/80186 instruction set
        /G2       generate code using 80286 instruction set
        /Gc       generate code using FORTRAN/PASCAL calling and
naming
                  conventions
        /Gd       generate code using standard C calling and naming
                  conventions (default)
        /Ge       generate code using stack checking routines
(default)
        /Gi       incremental compile if used with /qc; only
functions that
                  have changed are compiled;  implies /Li option
        /Gm       strings are generated in the CONST segment
        /Gr       _fastcall functions are enabled.   Parameters can
be
                  passed via registers instead of on the stack
        /Gs       suppress generation of stack checking code
        /Gt[num]  places data items larger than "num" bytes into
different
                  segments. The default value for "num is 256 bytes.
        /Gw       generate Windows functions entry/exit code
        /GW       generates more efficient Windows functions entry/
exit
                  code.  Use /Gw for callback routines.
        /H   external name length restriction (default 31)
        /HELP     calls QuickHelp utility.  /help is also valid.
        /I  inserts  in front of current include search path
        /J        all chars should be unsigned (default is signed)
        /Lc       linker is to generate compatibility mode programs.
        /Li[num]  incremental link using ILINK instead of LINK.
Results
                  in larger EXE size.
        /Lp       linker is to generate protected mode executable.
```

```
        /Lr       linker is to create real mode executable.
        /link  passes library names in  to LINK
        /MA  passes  to MASM if .ASM files found on command line
        /MD       creates Dynamic Link Library for OS/2;  The same
as:
                  /ALw /FPi /G2 /DDLL /DMT.   No library search
record is defined
        /ML       links the C run-time library as part of the
Dynamic Link
                  Library.  The library search record is set to
LLIBCDLL.LIB.

        /MT       enable multithread program support in OS/2
        /ND set data segment name to
        /NM set module name to
        /nologo   suppress sign-on banner display
        /NT set code segment name to
        /O[opts]  where "opts" is a string containing the following:

                  a   relax alias checking
                  c   enable block level local common expressions
                  d   disable all optimizations
                  e   enable global register optimization
                  g   enable global optimizations and global common
                      expressions
                  i   enable generation of intrinsic functions
                  l   enable loop optimization
                  n   disable unsafe loop optimization (default)
                  p   improve consistency in floating-point
calculations
                  r   disable inline returns from functions
                  s   optimize for size
                  t   optimize for speed (default)
                  w   assume no aliasing except across function calls
                  x   maximize optimization (/Ocegilt/Gs)
                  z   enable maximum loop and global-register
allocation
                      optimization

        /P         preprocess source generating .I intermediate file
        /qc        quick compile option.  Incompatible with the
following
                   options: /Fa /Fc /Fl /FPa /FPc /FPc87 /Fs /Gm /
H /Ow /Zc
        /Sl   set listing file line width to  (79-132, default 79)
        /Sp   set page length  (15-255, default 63)
```

```
/Ss set source listing subtitle
/St
```

void clearerr(FILE *stream)

- prototype in stdio.h

- resets error indication
- see ferror()

int ferror(FILE *stream)

- prototype in stdio.h

- tests stream for read/write error, return nonzero if error
- error remains set until clearerr() or rewind() is called

void rewind(FILE *fp)

- prototype in stdio.h

- equivalent to fseek(fp,0L,SEEK_SET), except EOF and error indicators are cleared
- returns 0 if pointer moved OK, else nonzero

int fseek(FILE *fp, long offset, int mode)

- prototype in stdio.h

- sets file pointer associated with fp to position which is offset
  bytes beyond file location given by mode

- mode is 0 (beginning of file or SEEK_SET)
          1 (current position or SEEK_CUR)
          2 (end of file or SEEK_END

- discards any character pushed back by ungetc()
- fseek() clears eof indicator but not file error indicator
- returns 0 if pointer moved OK, nonzero if file not opened or
  invalid seek for device. DOS does not report an error if an
  attempt to seek past EOF is made

- see ftell()

long ftell(FILE *fp)

- prototype in stdio.h
- returns current file position in bytes from beginning of file
  or -1L on error
- see fseek()

clock_t clock(void)

- prototype in time.h

- returns elapsed processor time used by the calling process,
  or -1L on error
- C startup code retrieves the clock count as preparation for a
  later call to clock() (Turbo C++ and some versions of MS C)
- resolution is limited to 54ms for both MS C and Turbo C due
  to the dependency on INT 8 clock tics
- CLK_TCK is shouldn't be used unless CLOCKS_PER_SEC isn't available
  since the latter is defined by ANSI.
- MS C uses INT 21,2C to get the processor time with 1000 as the
  value of CLK_TCK
- Turbo C uses INT 1A,2 to get the processor time with 18 as the
  value of CLK_TCK

- Example usage:

```
clock_t start, finish, duration;

start = clock();
delay(2000);
finish = clock();
duration = finish - start;
printf("%ld - %ld = %ld tics\n", start, finish, duration);
```

int close(int fh)

- prototype in io.h

- closes file fh where fh is file handle from _creat(), creat(), creatnew(), creattemp(), dup(), dup2(), _open(), or open()
- use fdopen() to change file handle to file stream
- does not write a Ctrl-Z to a text file on closing
- returns 0 if OK; else -1 & errno may be set of EBADF

TC: unsigned coreleft(void) TC: unsigned long coreleft(void) - prototype in alloc.h - returns a measure of unused memory (see malloc()) - return value is int for models tiny, small, medium return value is long for models compact, large, huge - no corresponding function in MS C

double cos(double x)

- prototype in math.h

- returns the cosine of angle x
- x must be in radians

double cosh(double x)

- prototype in math.h

- returns the hyperbolic cosine of angle x
- x must be in radians

TC: struct country *country(int cntrycode,struct country *cntryp)

- prototype in dos.h
- returns country-dependent information

int cprintf(char *format [, argument,...])

- prototype in conio.h

- sends formatted output to the console
- TC use either direct memory writes or BIOS calls depending on the
  value of the external integer directvideo
- see PRINTF SPEC for format specifications

Specifications for printf()

%[-][+][space][#][width][.prec][size]type

Field		Description
-		left justify
+		prefix positive numbers with '+', overrides blank
space		prefix positive numbers with space
#		prefix octal or hex with 0, (see manual for Turbo C)
width		width of output string in total characters, a 0 before width causes padding with zeros on left.
.prec		decimal precision in characters
size	F	far pointer
	N	near pointer
	h	specifies short
	l	specifies long
	L	long double
type	c	single character
	d	signed decimal integer
	e	signed exponential, 'e' is output in string
	E	signed exponential, 'E' is output in string
	f	signed floating point in form of sddd.ddd
	g	formats e or f (depends on size), e is output
	G	formats e or f (depends on size), E is output
	i	signed decimal integer
	n	integer pointer
	o	unsigned octal integer
	p	void pointer; MS C always uses far pointers, in Turbo C, size is dependent on memory model
	s	string pointer
	u	unsigned decimal integer
	x	unsigned integer in lower case hex format
	X	unsigned integer in upper case hex format

- if width or precision are specified as an asterisk '*', an int from the argument list is used as the width or precision. If the

width is too small the field is expanded. Example:

```
int len = 5;
char *string = "This is a string"

printf("%*.*s", len, len, "string");
```

is functionally similar to a specification string of "%5.5s"
which prints a max of 5 characters of the string.

void cputs(char *s)

- prototype in conio.h

- writes string s to console
- TC: uses either direct memory writes or BIOS calls depending
  on the value of the extern directvideo

int creat(const char *filename, int permis)

- prototype in io.h, also include stat.h

- creates new file (overwrites existing file) of filename with access permission "permis" which is:

```
      S_IWRITE
      S_IREAD
      S_IWRITE | S_IREAD
```

  for permission to write, read or read/write.

- file is created set to _fmode (O_TEXT or O_BINARY)

TC: int creatnew(const char *fname, int attr) MSC: unsigned creatnew(const char *fname, unsigned attr, int fh) - prototype in io.h, also include dos.h - creates new file filename with file attribute attr: - attr consists of a combination of the following: FA_RDONLY set read only attribute FA_HIDDEN set hidden attribute FA_SYSTEM set system attribute - returns nonnegative integer if OK, else -1 - file is created set to _fmode (O_TEXT or O_BINARY) - TC: returns nonnegative integer if OK, else -1 MSC: return 0 if ok, or DOS error code

int cscanf(const char *format [,argument,...])

- prototype in conio.h
- performs formatted input from console
- see SCANF and SCANF SPEC

int vsscanf(const char *string, const char *format, va_list argp) - prototype in stdio.h - like sscanf(), except arguments accepted from va_list parm

Specifications for scanf()

%[*][width][size]type

Field		Description
*		read next field but suppress assignment
width		maximum number of characters to read
size	F	far pointer
	N	near pointer
	h	specifies short
	l	specifies long
type	%	prints % character
	c	character data
	d	signed decimal integer
	D	signed long int
	e,E	signed exponential
	f	signed floating point in form of sddd.ddd
	g,G	used for e and f formats
	i	signed decimal, octal or hex integer
	I	signed decimal, octal, or hex long integer
	n	integer pointer
	o	unsigned octal integer
	O	unsigned octal long integer
	p	void pointer; MS C always uses far pointers, in Turbo C, size is dependent on memory model
	s	string pointer
	u	unsigned decimal integer
	U	unsigned decimal long integer
	x	unsigned hexadecimal integer
	X	unsigned hexadecimal long integer

- The unpredictability of scanf() in the DOS environment precludes it's use in a user friendly user interface.

char *ctime(const long *clock)

- prototype in time.h

- converts date and time to a string
- clock is a long integer (such as that returned by time())
- generates a 26-character string for date and time
- returns pointer to string

TC: void ctrlbrk(int (*fptr)(void))

- prototype in dos.h

- sets a new control-break handler function pointed to by fptr
- if Ctrl-C hit, then function pointed to by fptr is called
- use longjmp() to return to a given point in the program

MSC: int dieeetomsbin(double *srcdbl, double *dstdbl) MSC: int dmsbintoieee(double *srcdbl, double *dstdbl) MSC: int fieeetomsbin(float *srcflt, float *dstflt) MSC: int fmsbintoieee(float *srcflt, float *dstflt) - prototype in math.h dieeetomsbin() converts IEEE double to Microsoft double dmsbintoieee() converts Microsoft double to IEEE double fieeetomsbin() converts IEEE float to Microsoft float fmsbintoieee() converts Microsoft float to IEEE float - srcdbl and srcflt are pointers to source values - dstdbl and dstflt are pointers to the destination - returns 0 for success; 1 for failure (overflow)

double difftime(time_t time2, time_t time1)

- prototype in time.h
- returns (time2-time1) difference in seconds (double)

MSC: struct _div_t div(int numer, int denom) MSC: struct _ldiv_t ldiv(long int numer, long int denom) - prototype in stdlib.h - divides numerator by denominator giving quotient and remainder - returns div_t or ldiv_t structure of the form: struct _div_t struct _ldiv_t { { int quot; long quot; int rem; long rem; }; };

TC: int dosexterr(struct DOSERR *errinfo) MSC: int dosexterr(struct DOSERROR *errinfo) - prototype in dos.h - returns extended error information (unique to MSDOS 3.0+) - errinfo is pointer to DOSERR/DOSERROR structure (defined in dos.h) - if 0 returned, previous DOS call did not result in error - see perror

TC: long dostounix(struct date *dateptr, struct time *timeptr)

- prototype in dos.h

- converts date (as from getdate()) and time (as from gettime())
  into UNIX format

int dup(int handle)

- prototype in io.h

- returns a new file handle that duplicates the old handle
- returns -1 on error

char *ecvt(double value, int ndigit, int *decpt, int *sign)

- prototype in stdlib.h

- converts a floating pt number to a string of ndigit digits,
  returning pointer to that string
- decpt is position of decimal point (negative value means to left of returned digits) and sign is 0 if positive, else negative

int eof(int handle)

```
- prototype in io.h

- returns 1 if end of file for file associated with handle,
          0 if not end of file
         -1 if bad file number
```

void exit(int completioncode)

- prototype in process.h and stdlib.h

- terminates program, closes all files, buffered output is written,
  and any registered exit functions (via atexit()) are called

double exp(double x)

- prototype in math.h
- returns y = e**x or HUGE_VAL on error

double fabs(double x)

- prototype in math.h
- returns absolute value of x

TC: void far *farcalloc(unsigned long nunits, unsigned long units)

- prototype in alloc.h

- allocates memory from the far heap for an array of nunits elements, each units bytes long
- returns pointer to allocated block or NULL if not enough memory
- can allocate all of memory & chunks bigger than 64K
- must use far pointers

TC: unsigned long farcoreleft(void)

- prototype in alloc.h

- returns unused memory in bytes beyond highest allocated block

TC: void farfree(void far *block)

- prototype in alloc.h
- frees block of previously allocated far memory (block must be allocated by farcalloc())

TC: void far *farrealloc(void far *block, unsigned long newsize)

- prototype in alloc.h (malloc.h for MS C)

- adjusts size of allocated block to newsize, copying contents
  to a new location if necessary
- returns pointer to reallocated block or NULL on error

int fclose(FILE *fp)

- prototype in stdio.h

- closes a file stream (see fdopen() to convert handle to stream)
- generally flushes buffers, too
- return code should be checked. Attempts to delete an open file
  will damage the file system.
- returns 0 on success or EOF on error

int fcloseall(void)

- prototype in stdio.h

- closes all file streams except stdin & stdout
- returns 0 on success or EOF on error

char *fcvt(double value, int ndigit, int *decpt, int *sign)

- prototype in stdlib.h

- converts a floating pt number to a string of ndigit digits,
  returning pointer to that string
- correct digit is rounded for Fortran-F format output of the
  number of digits equal to ndigit
- decpt is position of decimal point (negative value means to left
  of returned digits) and sign is 0 if positive, else negative

int feof(FILE *stream)

- prototype in stdio.h

- detects end of file on a file stream
- returns zero if NOT eof, else nonzero

int fflush(FILE *fp)

- prototype in stdio.h

- writes contents of output buffers associated with fp to stream
  and clears input buffer contents; fp is NOT closed
- returns 0 on success or EOF on error

int fgetc(FILE *fp)

- prototype in stdio.h

- function (not a macro) which gets character from stream
- returns char (converted to an int) or EOF
- see getc()

void ungetch(int c)

- prototype in conio.h

- similar to ungetc() except pushes character back to keyboard buffer

int fgetchar(void)

- prototype in stdio.h

- gets character from stream
- same as fgetc(stdin)
- returns char (converted to an int) or EOF
- see getc()

char *fgets(char *s, int size, FILE *fp)

- prototype in stdio.h

- gets a string from a stream, reading until size - 1 characters
  have been read or a newline is detected
- newline character is retained
- returns s argument if OK, else NULL on eof or error

long filelength(int handle)

- prototype in io.h

- gets file size in bytes for file associated with handle
- value returned includes ^Z in ASCII files; value is the same as
  displayed in the DOS DIR command
- returns -1L on error

int fileno(FILE *stream)

- prototype in stdio.h

- gets file handle for the given stream
- error return undefined

double floor(double x)

- prototype in math.h
- returns largest integer <= x

int flushall(void)

- prototype in stdio.h

- same as fflush() except ALL open file streams are done
- returns integer indicating number of open file streams

double fmod(double x, double y)

- prototype in math.h

- calculates x modulo y, the remainder of x/y, returning the remainder

TC: void fnmerge(char *path, const char *drive, const char *dir, const char *name, const char *ext) - prototype in dir.h - makes a file name (path) from drive, dir, name, and ext - dir can include subdirectories - maximum sizes for these strings are: MAXPATH 80 path MAXDRIVE 3 drive - includes colon (:) MAXDIR 66 dir - includes leading/trailing backslashes MAXFILE 9 name MAXEXT 5 ext, including leading dot (.) - invertible with fnsplit()

TC: void fnsplit(const char *path, char *drive, char *dir, char *name, char *ext) - prototype in dir.h - splits a file name from path into drive, dir, name, and ext - dir can include subdirectories - maximum sizes for these strings are: MAXPATH 80 path MAXDRIVE 3 drive - includes colon (:) MAXDIR 66 dir - includes leading/traing backslashes MAXFILE 9 name MAXEXT 5 ext, including leading dot (.) - invertible with fnmerge()

unsigned FP_OFF(void far *fptr)

- prototype in dos.h

- gets offset of far pointer fptr
- returns unsigned integer value
- not available in earlier versions of MS C; use the following:

```
#define FP_OFF(fptr)  ((unsigned)(fptr))
```

- see FP_SEG()

unsigned FP_SEG(void far *fptr)

- prototype in dos.h

- gets segment of far pointer fptr
- returns unsigned integer value
- not available in some versions of MS C; use the following:

```
  #define FP_SEG(fptr) ((unsigned)((unsigned long)(fptr) >>
16))
```

- see FP_OFF()

int fprintf(FILE *fp, const char *format [, arg1, arg2, ...])

- prototype in stdio.h

- returns number of bytes it attempted to write regardless of success. To check for a successful fprintf call on a buffered stream, use fflush which returns accurate error information.
- using printf or any of the stdio functions results in a default allocation of 512 bytes for the I/O buffer and the inclusion of the standard memory allocation functions
- see printf() and PRINTF SPEC

int vsprintf(char *string, const char *format, va_list param)

- prototype in stdio.h
- sends formatted output to a string

int fputc(int c, FILE *fp)

```
        - prototype in stdio.h

        - puts char c to stream fp
        - using fputc or any of the stdio functions results in a
default
          allocation of 512 bytes for the I/O buffer and the
inclusion of
          the standard memory allocation functions
        - returns c  or EOF
```

int fputs(const char *s, FILE *fp)

- prototype in stdio.h

- using fputs or any of the stdio functions results in a default
  allocation of 512 bytes for the I/O buffer and the inclusion of
  the standard memory allocation functions
- see puts()

int puts(const char *string)

- prototype in stdio.h
- copies string to stdout and appends a newline

size_t fread(void *buf, size_t size, size_t count, FILE *fp)

- prototype in stdio.h

- reads (count * size) bytes from file stream "fp"
- returns the number of blocks actually read
- to get better feedback from this function, the parameters "size"
  and "count" may be swapped. If count is 1, this method allows
  fread() to return the actual number of bytes read
- using fread or any of the stdio functions results in a default
  allocation of 512 bytes for the I/O buffer and the inclusion of
  the standard memory allocation functions
- returns 0 (or short count) on eof or error

- see fopen() fread() setvbuf()

int setvbuf(FILE *stream, char *buf, int type, size_t size)

- prototype in stdio.h

- causes buf to be used for I/O buffering instead of the auto-
  matically allocated buffer; used after given stream is opened
  and before output
- in setvbuf, if buf is NULL, a buffer is allocated via malloc()
- no I/O to the file should occur until after buffering is set

- see setbuf()

void setbuf(FILE *stream, char *buf)

- prototype in stdio.h

- causes "buf" to be used for I/O buffering instead of the automatically allocated buffer, and are used after given stream is opened
- if "buf" is NULL, I/O is unbuffered
- no I/O to the file should occur after opening file until buffering is set

- see setvbuf()

void free(void *pseg)

- prototype in stdlib.h, alloc.h (malloc.h for MS C)

- frees allocated block located at pseg
- MS C ignores a NULL parameter (see _ffree())

FILE *freopen(const char *fname, const char *type, FILE *fp)

- prototype in stdio.h

- substitutes named file in place of open fp and closes original fp
- useful for changing file attached to stdin, stdout, or stderr
- returns fp on success or NULL on error
- see fopen()

double frexp(double value, int eptr)

- prototype in math.h

- calculates mantissa x (a double < 1) and n (integer) such that value = x * 2**n, storing n in word that eptr points

int fscanf(FILE *fp, const char *format, arg1, arg2, ...)

- prototype in stdio.h
- gets formatted input from a stream fp
- see scanf() and SCANF SPEC

int fstat(int handle, struct stat *buff)

- prototype in stat.h
- see stat()

Drive Status Bytes Found in BIOS Data Area

Drive recalibration status (at 40:3E)

```
|7|6|5|4|3|2|1|0| byte at 40:3E
 | | | | | | | `--- 1=recalibrate drive 0
 | | | | | | `---- 1=recalibrate drive 1
 | | | | | `----- 1=recalibrate drive 2
 | | | | `------ 1=recalibrate drive 3
 | | `----------- unused
 `------------ 1=working interrupt flag
```

Diskette motor status

```
|7|6|5|4|3|2|1|0| byte at 40:3F
 | | | | | | | `--- 1=drive 0 motor on
 | | | | | | `---- 1=drive 1 motor on
 | | | | | `----- 1=drive 2 motor on
 | | | | `------ 1=drive 3 motor on
 | | `----------- unused
 `------------ 1=write operation
```

Disk Status Byte

```
|7|6|5|4|3|2|1|0| byte at 40:41
 | | | | | | | `--- invalid diskette command
 | | | | | | `---- diskette address mark not found
 | | | | | `----- sector not found
 | | | | `------ diskette DMA error
 | | | `------- CRC check / data error
 | | `-------- diskette controller failure
 | `--------- seek to track failed
 `---------- diskette time-out
```

Last diskette data rate selected

```
|7|6|5|4|3|2|1|0| byte at 40:8B
 | | | | `--------- step rate time selected (see
below)
 `---------------- data rate selected (see below)
```

Bits
76 Diskette Data Rate Selected

```
 00  500K bps
 01  300K bps
 10  250K bps
 11  reserved
```

Bits
54 Diskette Step Rate Time Selected
00 step rate time of 0C
01 step rate time of 0D
10 step rate time of 0A
11 reserved

Media state for fixed drives (bytes at 40:90-93)

```
|7|6|5|4|3|2|1|0| bytes at 40:90-40:93
 | | | | | `------- reserved (set to 1)
 | | | | `-------- reserved
 | | | `---------- 0=media/drive unestablished
 | | `------------ reserved
 `-------------- data rate (see below)
```

Bits
76 Fixed Disk Data Rate Selected
00 500K bps
01 reserved
10 250K bps
11 reserved

- not all OEM's implemented these bytes identically. Some are also AT specific.

size_t fwrite(const void *buf, size_t size, size_t count, FILE *fp)

- prototype in stdio.h

- writes (count * size) bytes to file stream "fp"
- swapping parameters "size" and "count" can often provide more exact feedback (exactly how many bytes were written)
- returns number of data blocks actually written or a short count on error

- see fopen() fread() setvbuf()

char *gcvt(double value, int ndigit, char *buf)

- prototype in stdlib.h

- converts a floating point number to a string of ndigit digits,
  storing string into buf and returning pointer to that string
- outputs in Fortran-F format if possible, else in Fortran-E format

TC: void geninterrupt(int interrupt_num)

- prototype in dos.h

- generates actual interrupt for "interrupt_num" in code
- not a true function, but inline code generation

int getc(FILE *fp)

- prototype in stdio.h
- macro which returns next character in file stream "fp" or EOF
  on end of file or error

TC: int getcbrk(void)

- prototype in dos.h

- gets control-break setting

- return 0 Ctrl-C is off
        1 Ctrl-C is on

int getch(void)

- prototype in conio.h
- returns next character from console without echoing

int getchar(void)

- prototype in stdio.h

- returns next character in file stream stdin or EOF on end of file or error
- implemented as a macro

int getche(void)

- prototype in conio.h

- function which returns next character from console WITH echoing

MSC: flagType GetColor(LINE line, la *colorlist, PFILE pFile) MSC: void PutColor(LINE line, la *colorlist, PFILE pFile) - prototype in ext.h - GetColor returns nonzero if a color is attached to the line; zero otherwise

TC: int getcurdir(int drive, char *direc)

```
        - prototype in dir.h

        - get current directory for specified drive (0=default,
1=A, etc)
        - direc will contain directory name

        - returns 0 if OK
                -1 on error
```

char *getcwd(char *buf, int n)

- prototype in dir.h

- gets full path name of current working directory up to n bytes,
  placed into buf
- returns buf pointer, else NULL

char *getenv(const char *envvar)

- prototype in stdlib.h

- gets string from environment
- MSDOS environment consists of strings of form envvar=varvalue,...
- returns varvalue or 0 if envvar not found in environment

TC: void getfat(unsigned char drive, struct fatinfo *fatblkp)

- prototype in dos.h

- returns information from the file allocation table for the specified drive (0=default,1=A, etc) into fatblk
- structure fatinfo is defined as:

```
struct fatinfo
    {
    char  fi_sclus;             - sectors per cluster
    char  fi_fatid;             - media descriptor byte
from FAT
    int   fi_nclus;             - cluster on disk
    int   fi_bysec;             - bytes per sector
    };
```

- closest MS C function is _dos_getdiskfree()
- see INT 21,1C

TC: void getfatd(struct fatinfo *fatblkp)

- prototype in dos.h

- returns information from the file allocation table for the DEFAULT drive into fatblk
- structure fatinfo is defined as:

```
struct fatinfo
    {
    char  fi_sclus;             - sectors per cluster
    char  fi_fatid;             - media descriptor byte
from FAT
    int   fi_nclus;             - cluster on disk
    int   fi_bysec;             - bytes per sector
    };
```

- closest MS C function is _dos_getdiskfree()
- see INT 21,1B getfat()

TC: char *getpass(const char *prompt)

- prototype in conio.h

- reads a password from system console after typing prompt, without echoing
- password cannot exceed 8 chars (not counting null terminator)

MSC: int getpid(void)

- prototype in process.h

- returns process ID identifying the calling process (PSP segment)

- Turbo C uses getpsp()

TC: unsigned getpsp(void)

- prototype in dos.h

- returns segment address of the PSP using DOS INT 21,62
- valid only for DOS 3.x
- use global variable _psp to get PSP instead for DOS 2.X
- MS C uses getpid()

char *gets(char *s)

- prototype in stdio.h

- reads string from stdin until newline character is read
- newline character is replaced by \0
- returns string or NULL on end-of-file or error

TC: int getverify(void)

- prototype in dos.h

- returns current state of verify flag (0==off, 1==on)
- see INT 21,54

INT 21,54 - Get Verify Setting

```
AH = 54h


on return:
AL = 00  verify off
   = 01  verify on


- returns value of disk read after write verification flag

- see   INT 21,2E
```

INT 21,2E - Set/Reset Verify Switch

```
AH = 2E
AL = 00 to set off
   = 01 to set verify on
DH = 00 for DOS versions before 3.0


returns nothing


- with the verify setting on, disk I/O is more secure but
  takes longer to complete


- see   INT 21,54
```

int getw(FILE *fp)

```
        - prototype in stdio.h

        - gets integer from file stream fp
        - returns EOF (-1) on eof or error
        - use feof() or ferror() to verify -1 is an integer data
word, and
          not an error return
```

struct tm *gmtime(const time_t *clock)

- prototype in time.h

- clock is a long integer (such as that returned by time())
- returns GMT time in struct tm (see time.h) correcting for time
  zone and any daylight savings time
- global variable timezone is difference in seconds between GMT
  and local standard time

MSC: void huge *halloc(long num, size_t size)

```
        - prototype in malloc.h

        - num = count of elements to allocate
        - size = size of each element;  each element is set to zero;
                 must be a power of 2 if size is over 128K
        - alignment assures compatibility with all data types (para)
        - returns pointer to allocated block on success or NULL on
failure
        - allocates memory directly from DOS

        - see   hfree()
```

MSC: void hfree(void huge *block)

- prototype in malloc.h

- block = pointer to block of allocated memory
- memory is returned to DOS
- freeing an unallocated block will corrupt the DOS's MCB chain

- see halloc()

double hypot(double x, double y)

- prototype in math.h
- returns z where z**2 = x**2 + y**2, or HUGE_VAL on error

MSC: int inp(unsigned port) MSC: unsigned inpw(unsigned port) - prototype in conio.h - port = hardware I/O port 0-3FFh - returns byte or word data read from port

TC: int inport(int port)

- prototype in dos.h

- inport() reads word from input port port
- MS C uses inpw()
- see inportb()

TC: unsigned char inportb(int port)

- prototype in dos.h

- inportb() is macro that reads a byte from hardware port 'port'
- MS C uses inp()
- see inport()

int int86(int intr_num, union REGS *inregs, union REGS *outregs)

- prototype in dos.h

- executes 8086 software interrupt specified by intr_num
- copies register values from inregs into the registers
- if CF is set, an error has occurred
- preserves SP register so calls to INT 25 and INT 26 via this
  function don't require a stack adjustment
- unique to DOS
- see int86x() intdos() intdosx() intr() REGS

int int86x(int intr_num, union REGS *inregs, union REGS *outregs, struct SREGS *segregs) - prototype in dos.h - Executes 8086 software interrupt specified by intr_num - Copies register values from inregs into the registers - also copies segregs->x.ds and segregs->y.es into DS/ES - if CF is set, an error has occurred - preserves SP register so calls to INT 25 and INT 26 via this function don't require a stack adjustment - unique to DOS - see int86() intdos() intdosx() intr() REGS

int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS *segregs) - prototype in dos.h - executes 8086 software interrupt INT 21 - copies register values from inregs into the registers - copies segregs->x.ds and segregs->y.es into DS/ES - if CF is set, an error has occurred - unique to DOS - see intdos() int86() int86x() intr() REGS

TC: void intr(int intr_num, struct REGPACK *preg)

- prototype in dos.h

- same as int86() except that preg contains registers values
  both before & after executing interrupt
- preserves SP register so calls to INT 25 and INT 26 via this
  function don't require a stack adjustment
- see int86x() intdos() intdosx() REGS

int intdos(union REGS *inregs, union REGS *outregs)

- prototype in dos.h

- Executes 8086 software interrupt INT 21
- Copies register values from inregs into the registers
- if CF is set, an error has occurred
- unique to DOS
- see intdosx() int86() int86x() intr() REGS

TC: int ioctl(int handle, int cmd [, void *argdx, int argcx])

```
        - prototype in io.h

        - direct interface to  INT 21,44  (IOCTL)
        - cmd = 0  get device information
                1  set device information (in argdx)
                2  read argcx bytes into addr given by argdx
                3  write argcx bytes from addr given by argdx
                4  same as 2, but handle treated as drive
(0=def.,1=A)
                5  same as 3, but handle treated as drive
(0=def.,1=A)
                6  get input status
                7  get output status
                8  test removability (DOS 3.x)
                11 set sharing conflict retry count (DOS 3.x)
        - cmd = 0,1; returns device information (DX of IOCTL call)
        - cmd = 2-5; returns count of bytes transferred
        - cmd = 6,7; returns device status
        - returns -1 on error & errno = EINVAL, EBADF, or EINVDAT
        - no corresponding function in MS C
```

INT 21,44,F / IOCTL,F - Set Logical Drive

```
AH = 44h
AL = 0F
BL = new drive number (0=default, 1=A:, 2=B:, 3=C:, ...)


on return
AX = error code if CF set  (see DOS ERROR CODES)
AL = logical drive number (should be equal to BL input)
   = 0 if drive accessed by only one drive specifier
   = 1 if drive A:, 2 if drive B:, etc



- changes the logical drive designator of the physical drive
  to be accessed next
- implemented from DOS 3.2
```

Character Test Macros

```
        int isalnum( int c )    - nonzero if letter or digit
        int isalpha( int c )    - nonzero if letter
        int isascii( int c )    - nonzero if in range 0-127
        int iscntrl( int c )    - nonzero if 0x7F, or 0x00-0x1F
        int isdigit( int c )    - nonzero if digit
        int isgraph( int c )    - nonzero if printable 0x21-0x7E
excl space
        int islower( int c )    - nonzero if lowercase
        int isprint( int c )    - nonzero if printable 0x20-0x7E
        int ispunct( int c )    - nonzero if punct char (iscntrl |
isspace)
        int isspace( int c )    - nonzero if space,tab, CR, LF, VT
or FF
        int isupper( int c )    - nonzero if uppercase
        int isxdigit( int c )   - nonzero if hexadecimal digit


        - prototype in ctype.h
```

int isatty(int handle)

- prototype in io.h

- if handle is associated with a character device like tty, console, printer, or serial port returns a non-zero integer
  otherwise returns zero

Character Test Macros

```
        int isalnum( int c )    - nonzero if letter or digit
        int isalpha( int c )    - nonzero if letter
        int isascii( int c )    - nonzero if in range 0-127
        int iscntrl( int c )    - nonzero if 0x7F, or 0x00-0x1F
        int isdigit( int c )    - nonzero if digit
        int isgraph( int c )    - nonzero if printable 0x21-0x7E
excl space
        int islower( int c )    - nonzero if lowercase
        int isprint( int c )    - nonzero if printable 0x20-0x7E
        int ispunct( int c )    - nonzero if punct char (iscntrl |
isspace)
        int isspace( int c )    - nonzero if space,tab, CR, LF, VT
or FF
        int isupper( int c )    - nonzero if uppercase
        int isxdigit( int c )   - nonzero if hexadecimal digit


        - prototype in ctype.h
```

char *itoa(int value, char *string, int radix)

- prototype in stdlib.h
- converts value to string where radix specifies base (2-36)

int kbhit(void)

- prototype in conio.h
- returns nonzero if keystroke available else 0

long labs(long n)

- prototype in stdlib.h
- returns absolute long value of n

double ldexp(double value, int exp)

- prototype in math.h
- returns value x 2**exp

void *lfind(void *key,void *base,int *nelem,int width,int (*fcmp)())

- prototype in stdlib.h

- does linear search for items in an unsorted table;
- base points to 0th element of table
- nelem points to int containing number of entries in table
- width contains number of bytes in each entry
- key points to the search key
- fcmp points to user-written comparison routine, where key and
  elem are passed to it as pointers. fcmp returns:

   integer < 0 if search key < elem
   integer = 0 if equal
   integer > 0 if search key > elem

- returns 0 if no match found, else address of first matching entry

struct tm *localtime(const time_t *clock)

- prototype in time.h

- clock is a long int (such as that returned by time())
- returns time in struct tm (see time.h) correcting for time zone
  and any daylight savings time
- global variable timezone is difference in seconds between GMT
  and local standard time

TC: int lock(int handle, long offset, long length)

- prototype in io.h

- locks arbitrary, non-overlapping regions of any file (DOS 3.X),
  preventing reads/writes to those regions
- returns 0 on success, else -1 on error
- all locks must be released before program termination
- MS C uses locking()
- see unlock()

MSC: int locking(int handle, int mode, long nbytes)

```
        - prototype in io.h, sys\locking.h

        - handle = opened file handle
        - nbytes = bytes to lock beginning with current file
position
        - mode  = locking mode:
                = LK_LOCK       lock region; on failure waits 1 sec
and
                                attempts again;  tries 10 times
                = LK_RLCK       same as LK_LOCK
                = LK_NBLCK      lock region; returns immediately on
error
                = LK_NBRLCK     lock region; returns immediately on
error
                = LK_UNLCK      unlock previously locked region

        - multiple locks may occur in a single file
        - overlapping locked regions are not allowed
        - returns 0 on success or -1 on failure

        - Turbo C uses lock() and unlock()
```

TC: int unlock(int handle, long offset, long length)

- prototype in io.h

- unlock() removes locks placed on a file region by lock()
- returns 0 on success, else -1 on error
- all locks must be released before program termination
- MS C uses locking()
- see lock (C)

double log(double x)

- prototype in math.h
- returns natural logarithm of x, or -HUGE_VAL on error

double log10(double x)

- prototype in math.h
- returns base 10 logarithm of x, or -HUGE_VAL on error

void *lsearch(void *key,void *base,int *nelem,int width,int (*fcmp)())

```
        - prototype in stdlib.h

        - does linear search in unsorted table for key
        - base points to 0th element of table
        - nelem integer pointer to number of entries in table
        - width contains number of bytes in each entry
        - key points to the search key
        - fcmp points to comparison routine, where key and elem are
passed
          to it as pointers.  fcmp returns:

                integer < 0 if search key < elem
                integer = 0 if equal;
                integer > 0 if search key > elem

        - returns 0 if no match found, else address of first
matching entry
```

long lseek(int handle, long offset, int mode)

- prototype in io.h

- moves file position of "handle" to "offset" relative to "mode"

  mode = 0 - SEEK_SET beginning of file
  mode = 1 - SEEK_CUR current position
  mode = 2 - SEEK_END or end of file

- returns -1L on error

- see tell()

long tell(int handle)

- prototype in io.h

- returns offset within file corresponding to handle
- returns -1L on error

- see lseek()

char *ltoa(long value, char *string, int radix)

- prototype in stdlib.h

- converts value to string where radix specifies base (2-36) for conversion

Memory Manipulation Functions

```
        void  *memccpy( void *dest, const void *src, int c, size_t
n )
        void  *memchr( const void *s, int c, size_t n )
        int   memcmp( const void *s1, const void *s2, size_t n )
        void  *memcpy( void *dest, const void *src, size_t n )
        int   memicmp( const void *s1, const void *s2, size_t n )
        void  *memmove( void *dest, const void *src, size_t n )
        void  *memset( void *s, int c, size_t n )
        void  movedata( unsigned srcseg, unsigned srcoff,
                        unsigned dstseg, unsigned dstoff, size_t n )

        - prototypes for the preceding functions are in mem.h,
string.h
        - the following are prototyped in mem.h only (TC):

        void  movmem( void *src, void *dest, unsigned length );
(mem.h only)
        void  setmem( void *dest, unsigned length, char value );
(mem.h only)
```

void *memccpy(void *dest, const void *src, int ch, size_t n)

- prototype in string.h, mem.h

- copies from src to dest until ch is copied or n bytes are copied
  returning a ptr to byte in dest immediately following ch or NULL

void *memchr(const void *s, int ch, size_t n)

- prototype in string.h, mem.h
- searches first n bytes in s for ch, returning pointer to first occurrence or NULL if not found

int memcmp(const void *s1, const void *s2, size_t n)

- prototype in string.h, mem.h

- compares two strings s1 & s2 for a length of n bytes, returning a value:

```
< 0   if s1 < s2
= 0   if s1 = s2
> 0   if s1 > s2
```

void *memcpy(void *dest, const void *src, size_t n)

- prototype in string.h, mem.h

- copies n bytes from src to dest; returns dest
- if overlap occurs, result is undefined (ANSI)

int memicmp(const void *s1, const void *s2, size_t n)

- prototype in string.h, mem.h
- compares first n bytes of s1 & s2, case insensitive

void *memmove(void *dest, const void *src, size_t n)

- prototype in string.h, mem.h

- copies n bytes from src to dest; returns dest
- allows copying overlapped strings (ANSI)

void *memset(void *s, int ch, size_t n)

- prototype in string.h, mem.h

- memset sets all bytes of s to byte ch, with size of s = n; returns value of s

void far *MK_FP(unsigned segment, unsigned offset)

- prototype in dos.h

- returns a far pointer from offset and segment

- not available in some Microsoft C versions; use the following:

```
#define MK_FP(seg,off) \
 ((void far *) (((unsigned long)(seg) << 16) | (unsigned)(off))
```

- see FP_OFF() and FP_SEG()

int mkdir(const char *pathname)

- prototype in dir.h

- takes pathname & creates new directory with that name
- returns 0 if successful, else -1

char *mktemp(char *template)

- prototype in dir.h

- replaces template by unique file name & returns address of template if successful
- template should be string with six trailing Xs

double modf(double value, double *iptr)

- prototype in math.h

- splits value in integer and fraction part, storing integer part
  in area pointed to by iptr, returning the fractional part

void movedata(unsigned segsrc, unsigned offsrc, unsigned segdest, unsigned offdest, size_t n) - prototype in mem.h, string.h - copies n bytes from segsrc:offsrc to segdest:offdest

void movmem(void *src, void *dest, unsigned len)

- prototype in mem.h
- copies len bytes from src to dest

MSC: onexit_t onexit(onexit_t func)

- prototype in stdlib.h

- creates a list of functions to execute on normal program exit
- on exit functions are executed LIFO
- a max of 32 function can be registered
- returns pointer to "func" on success; NULL otherwise
- this is a Microsoft/Lattice extension; ANSI uses atexit()

- see exit() abort()

int open(const char *path, int access [, unsigned permis])

- prototype in io.h, also include stat.h & fcntl.h

- open attribute flags are defined in fcntl.h
- opens file "path" with access and optionally permis
- access is a combination of the following ("permis" follows):

Read/Write Access Flags (mutually exclusive):

O_RDONLY	open read only
O_WRONLY	open write only
O_RDWR	open read/write

Other Access Mode Flags:

O_APPEND	file pointer is placed at EOF before each write
O_CREAT	if file doesn't exist, create with "permis" attributes
O_TRUNC	if exists, truncate length to zero, but leave file attributes unchanged
O_BINARY	binary mode
O_TEXT	text mode
O_EXCL	used with O_CREAT, error occurs if file already exists
O_NDELAY	UNIX only

Permission Attributes (if creating):

S_IWRITE	write permission
S_IREAD	read permission
S_IREAD | S_IWRITE	read/write permission

- if O_BINARY nor O_TEXT is given, file is opened in translation
  mode (O_TEXT) given by global variable _fmode
- returns nonnegative number as file HANDLE, or -1 on error
- see sopen()

MSC: int outp(unsigned port, int datab) MSC: unsigned outpw(unsigned port, unsigned dataw) - prototype in conio.h - write data value to port specified in port - port = hardware I/O port - datab = byte value to write - dataw = word value to write - Turbo C uses outportb() and outport()

TC: void outportb(int port, unsigned char byte)

- prototype in dos.h

- outportb() is macro that writes byte to output port port
- MS C uses outp()
- see also outport()

TC: void outport(int port, int word)

- prototype in dos.h

- outport() writes word to output port port
- MS C uses outpw()
- see also outportb()

TC: char *parsfnm(const char *cmdline, struct fcb *fcbptr, int option)

- prototype in dos.h

- parses command line cmdline for a file name, placing it into a
  FCB as drive/filename/file ext, pointed to by fcbptr
- option is same as AL in MS-DOS function call INT 21,29
- returns pointer to byte beyond end of filename, 0=error

INT 21,29 - Parse a Filename for FCB

```
AH = 29h
AL = bit pattern to control parsing (see bit meanings below)
DS:SI = pointer to a filespec to parse
ES:DI = pointer to a buffer for unopened FCB


Bit patterns for parsing control found in AL:

|7|6|5|4|3|2|1|0|  AL
 | | | | | | | `---- 1 = ignore leading separators
 | | | | | | |       0 = don't ignore leading separators
 | | | | | | `----- 1 = modify drive ID if specified
 | | | | | |        0 = modify drive ID regardless
 | | | | | `------ 1 = modify filename if specified
 | | | | |         0 = modify filename regardless
 | | | | `------- 1 = modify extension if specified
 | | | |          0 = modify extension regardless
 `--------------- unused

on return:
AL = 00 if no wildcard characters present
   = 01 if wildcards present in string
   = FF if drive specifier is invalid
DS:SI = pointer to the first character after parsed filename
ES:DI = pointer to the updated unopened FCB


- retrieves filename from the command line string and places
  the filename components into an unopened FCB for later use
- if no filename is found a pointer is returned in ES:DI
that
  has a blank at ES:DI+1
- this function can be used to detect the existence of
logical DOS
  drives by creating a dummy filespec with a drive letter
and colon
  prepended.  If the drive is invalid, this function will
return FF
  in AL
```

TC: int peek(unsigned seg, unsigned off)

- prototype in dos.h
- returns word found at seg:off

TC: char peekb(unsigned seg, unsigned off)

- prototype in dos.h
- returns byte found at seg:off

void perror(const char *string)

- prototype in stdio.h
- prints error message to stderr, describing most recent call found in system call from current program

TC: void poke(unsigned seg, unsigned off, int value)

- prototype in dos.h

- writes word value to location seg:off

TC: void pokeb(unsigned seg, unsigned off, char value)

- prototype in dos.h

- writes byte value to location seg:off

double poly(double x, int n, double coeff[])

- prototype in math.h

- generates polynominal in x of degree n, with coefficients coeff,
  returning the value of this polynominal evaluated for x

double pow(double x, double y)

- prototype in math.h
- returns p where p = x ** y or +/- HUGE_VAL on error

double pow10(double x)

- prototype in math.h
- returns y where y = 10 ** x or HUGE_VAL on overflow error

int printf(const char *format [, arg1, arg2, ...])

- prototype in stdio.h
- printf formatted string
- see PRINTF SPEC for format specifiers

int putc(int c, FILE *fp)

- prototype in stdio.h

- puts char c to stream fp, returning c on success, else returns EOF on error

int putch(int ch)

- prototype in conio.h

- puts char ch to console with BEL, BS, TAB, LF and BEL characters
  are converted similar to DOS I/O
- TC uses direct video or BIOS depending on value of external
  variable directvideo
- returns nothing

int putchar(char c)

- prototype in stdio.h
- same as putc(c,stdout)

int putenv(const char *envvar)

```
        - prototype in stdlib.h

        - adds string to environment
        - DOS environment consists of strings of form
envvar=varvalue,...
        - returns 0 if OK
                  1 on failure
```

int putw(int n, FILE *fp)

- prototype in stdio.h

- puts integer word n to stream fp
- returns n or EOF on error. Use ferror() to verify
  if EOF is returned, since -1 is a legitimate integer.

void qsort(void *array, size_t n_elem, size_t elem_size, int (*fcmp)(const void *, const void *)) - prototype in stdlib.h - sorts the given array using the ACM quick sort routine - fcmp() is the same as for bsearch()

MSC: int raise(int sig)

- prototype in signal.h

- raises signal to program where 'sig' is:

```
      SIGABRT signal abnormal termination
      SIGILL  signal illegal instruction
      SIGSEGV signal illegal storage access
      SIGFPE  signal floating point error
      SIGINT  signal Ctrl-Break interrupt
      SIGTERM signal program termination (ignored in MS
DOS)
      SIGUSR1 user signal
      SIGUSR2 user signal
      SIGUSR3 user signal
```

- returns 0 on success; nonzero otherwise

- see signal()

int rand(void) void srand(unsigned seed) - prototype in stdlib.h - rand() returns pseudorandom numbers - re-initialize generator with srand(1) or set to new starting point with seed set other than to 1

void srand(unsigned seed)

- prototype in stdlib.h

- re-initializes the random number generator with seed
- see rand()

TC: int randbrd(struct fcb *fcbptr, int reccnt)

- prototype in dos.h

- randbrd() reads reccnt number of records using the open FCB via
  fcbptr, as indicated by the disk record field of the FCB (via
  INT 21,27)
- returns:
  0 all records read(written)
  1 EOF reached & last record read is complete
  2 reading records would have wrapped around address 0xffff
  3 EOF reached & last record is incomplete
- see randbrw()

INT 21,27 - Random Block Read Using FCB

```
AH = 27h
CX = number of records to read
DS:DX = pointer to an opened FCB


on return:
AL = 00 if read was successful
   = 01 if EOF (no data read)
   = 02 if DTA is too small
   = 03 if EOF (partial record read)
CX = actual number of records read
```

- allows random access and sequential reading of a group of records from a file opened with an FCB into the DTA
- FCB must be setup with drive id, filename, extension record length and random record number before call

TC: int randbrw(struct fcb *fcbptr, int reccnt)

- prototype in dos.h

- randbrw() returns 1 if there is not enough disk space to write
  the records (no records are written)
- see randbrd()

int read(int handle, char *buf, int size)

- prototype in io.h

- reads "size" bytes from file "handle" into "buf"
- removes CR's & reports EOF on a Ctrl-Z for text mode file
- returns number of bytes read, 0 on EOF, or -1 on error

void *realloc(void *pseg, unsigned size)

- prototype in stdlib.h & alloc.h (malloc.h for MS C)
- see malloc()

int rename(const char *oldf, const char *newf)

- prototype in stdio.h

- rename file from oldf to newf
- attempts to rename an open file may damage the file system
- returns 0 if OK, else -1 on error

int rmdir(const char *path)

- prototype in dir.h

- takes path & deletes directory with that name
- returns 0 if successful, else -1

MSC: int rmtmp(void)

- prototype in stdio.h

- cleans up all temporary files in the current directory created
  by tmpfile()
- should only be used on files in the current directory
- returns number of temporary files closed and deleted

MSC: FILE *tmpfile(void)

- prototype in stdio.h

- creates and opens a temporary stream file
- returns stream pointer or NULL on error
- file is automatically deleted when closed or rmtmp() is called
- rmtmp() works only if in original directory
- file is opened in binary write mode; except MS C version 4.0
  opened file in write translate mode

void *sbrk(int incr)

- prototype in alloc.h (malloc.h for MS C)

- dynamically changes the amount of space allocated to the calling
  programs data segment. Amount of allocated space is increased by
  amount incr (can be negative).

- returns 0 if OK
  -1 & errno is set to ENOMEM

int scanf(const char *format, arg1, arg2, ...)

- prototype in stdio.h
- accepts input from stdin and converts to format specs
- see SCANF SPEC

TC: int setcbrk(int value)

- prototype in dos.h

- sets control-break setting (value = 0 sets Ctrl-C checking off
  except for during console and printer I/O; value = 1 always
  check during DOS calls)

TC: void setdate(struct date *dateblk)

- prototype in dos.h

- sets DOS date via INT 21,2B
- DOS 3.3+ also sets CMOS clock

- MS C uses _dos_setdate()

void setmem(void *addr, int len, char value)

- prototype in mem.h
- sets len bytes in addr to value

int setmode(int handle, int mode)

```
        - prototype in io.h

        - sets mode of file associated with handle to binary
(O_BINARY) or
          text (O_TEXT) but not both
        - returns 0 if successful, else -1 on error
```

TC: int setverify(int value)

- prototype in dos.h

- sets state of verify flag (0==off, 1==on) to value
- no corresponding MS C function
- see INT 21,2E

double sin(double x)

- prototype in math.h

- returns sine of x
- x must be in radians

double sinh(double x)

- prototype in math.h

- returns hyperbolic sine of x
- x must be in radians

TC: void sleep(unsigned seconds)

- prototype in dos.h

- suspended program for seconds of time
- accurate in seconds to limit of MS-DOS clock
- no corresponding MS C function

int sprintf(char *s, const char *format [, arg1, arg2, ...])

- prototype in stdio.h
- prints formatted data into string s
- see PRINTF SPEC for format specifiers

double sqrt(double x)

- prototype in math.h
- returns square root of x, or 0 on error

int sscanf(const char *s, const char *format, arg1, arg2, ...)

- prototype in stdio.h
- see scanf() and SCANF SPEC

int stat(char *pathname, struct stat *buff) int fstat(char *handle, struct stat *buff) - prototype in stat.h - stat() and fstat() store information about a given open file (or directory) in the stat structure (in stat.h) - stat() gets information about the open file or directory, while fstat() gets info about the open file associated with handle - returns 0 if successful, else -1 on error

int stime(time_t *tp)

- prototype in time.h

- sets system time & date, where tp to value of time in seconds since 00:00:00 GMT Jan 1, 1970

char *stpcpy(char *dest, const char *src)

- prototype in string.h

- copies src into dest
- returns dest+strlen(src)

size_t strlen(const char *str)

- prototype in string.h

- returns number of characters in str, not counting the null terminating character

String Manipulation Functions

- prototype in string.h (a few also exist in stdlib.h)

Conversion:

```
char  *strlwr( char *s )
char  *strrev( char *s )
double strtod( const char *s, char **endptr )
long  strtol( const char *s, char **endptr, int radix )
unsigned long strtoul( const char *s, char **endptr, int
radix )
char  *strupr( char *s )
char  *strdup( const char *s )
```

Search:

```
char  *strchr( char *s, int c )
size_t strcspn( const char *s1, const char *s2 )
char  *strpbrk( const char *s1, const char *s2 )
char  *strrchr( const char *s, int c )
size_t strspn( const char *s1, const char *s2 )
char  *strstr( const char *s1, const char *s2 )
char  *strtok( char *s1, const char *s2 )
```

Comparison:

```
int strcmp( const char *s1, const char *s2 )
int strcmpi( const char *s1, const char *s2 )
int stricmp( const char *s1, const char *s2 )
int strncmp( const char *s1, const char *s2, size_t maxlen )
int strncmpi( const char *s1, const char *s2, size_t
maxlen )
int strnicmp( const char *s1, const char *s2, size_t
maxlen )
```

Copy:

```
char *stpcpy( char *dest, const char *src )
char *strcpy( char *dest, const char *src )
char *strcat( char *dest, const char *src )
```

```
char *strncat( char *dest, const char *src, size_t maxlen )
char *strncpy( char *dest, const char *src, size_t maxlen )
```

Miscellaneous:

```
size_t strlen( const char *s )
char  *strnset( char *s, int ch, size_t n )
char  *strset( char *s, int ch )
char  *strerror( int errnum )
char  *_strerror( const char *s )
```

char *strlwr(char *str)

- prototype in string.h
- converts str to all lower case

char *strrev(char *str)

- prototype in string.h
- reverses all characters in str, except null terminator

double strtod(const char *str, char **endptr)

```
        - prototype in string.h

        - converts string str to a double value
        - stops reading at tfirst character that cannot be
interpreted
          as part of a double value, returning in *endptr
        - str must have format:

          [ws][sn][ddd][.][ddd][fmt[sn]ddd]

          where [ws]  = optional whitespace
                [sn]  = optional sign (+-)
                [ddd] = optional digits
                [fmt] = option e or E
                [.]   = optional decimal point
```

long strtol(const char *str, char **endptr, int base) unsigned long strtoul(const char *s, char **endptr, int radix) - prototype in string.h - converts string str to long (or unsigned long) value - stops reading at first character that cannot be interpreted as part of a long value, returning in *endptr - str must have format: [ws][sn][0][x][ddd] where [ws] = optional whitespace [sn] = optional sign (+-) [0] = optional zero [x] = optional x or X [ddd] = optional digits - base is between 2 and 36; if 0, first few characters of str determines base to be used (oct,hex or dec) any other value of base is illegal

char *strupr(char *str)

- prototype in string.h
- converts to all uppercase

char *strchr(const char *str, int c)

- prototype in string.h

- scans str for first occurrence of c
- returns pointer to c in str, or NULL if not found

size_t strcspn(const char *str1, const char *str2)

- prototype in string.h

- returns length of initial segment of str1 that consists entirely of characters NOT from str2

char *strpbrk(const char *str1, const char *str2)

- prototype in string.h

- scans str1 for first occurrence of any character appearing in str2
- returns pointer to first occurrence; NULL if not found

char *strrchr(const char *str, int c)

- prototype in string.h

- scans a string in reverse direction for specified character c
- strrchr finds the LAST occurrence of c in str
- returns pointer to LAST occurrence, or NULL if not found

size_t strspn(const char *str1, const char *str2)

- prototype in string.h

- returns length of initial segment of str1 that consists entirely of characters from str2

char *strstr(const char *anchor, const char *testpattern)

- prototype in string.h

- returns a pointer within anchor where testpattern is found,
  or NULL if testpattern is not found in anchor
- may be incompatible with Microsoft C

char *strtok(char *str1, const char *str2)

- prototype in string.h

- parses str1 for tokens and separators, where separators are
  defined in str2
- first call returns a pointer to the first character of the first
  token in str1 and writes a null character into str1 immediately
  following the returned token; subsequent calls with NULL for
  the first argument will work through the string str1 in this way
  until no tokens remain
- when tokens are exhausted, returns NULL

int strcmp(const char *str1, const char *str2)

- prototype in string.h

- compares str1 to str2
- returns < 0 if str1 < str2, = 0 if str1 = str2, or > 0 if str1 > str2, using a signed comparison

int stricmp(const char *str1, const char *str2)

- prototype in string.h

- compares str1 to str2 without case sensitivity

- returns < 0 if str1 < str2
  = 0 if str1 = str2
  > 0 if str1 > str2

  using a signed comparison

int strncmp(const char *str1, const char *str2, size_t maxlen)

- prototype in string.h

- compares str1 to str2 up to maxlen characters

- returns < 0 if str1 < str2
  = 0 if str1 = str2
  > 0 if str1 > str2

  using a signed comparison

int strnicmp(const char *str1, const char *str2, size_t maxlen)

- prototype in string.h

- compares str1 to str2 up to maxlen chars ignoring case

- returns < 0 if str1 < str2
  = 0 if str1 = str2
  > 0 if str1 > str2

  using a signed comparison

char *strcpy(char *dest, const char *src)

- prototype in string.h

- copies src into dest
- returns dest

char *strcat(char *dest, const char *src)

- prototype in string.h
- appends src to dest

char *strncat(char *dest, const char *src, size_t maxlen)

- prototype in string.h

- appends up to maxlen characters of src to dest and then appends a null character

char *strncpy(char *dest, const char *src, size_t maxlen)

- prototype in string.h

- copies exactly maxlen characters from src to dest, truncating
  or null-padding dest
- dest might NOT be null-terminated if length of src >= maxlen

char *strnset(char *str, int ch, size_t n)

- prototype in string.h

- sets up to first n bytes of str to ch
- if n > strlen(str) then strlen(str) replaces n

char *strset(char *str, int ch)

- prototype in string.h
- sets all characters in str to ch

char *strerror(char *str)

- prototype in string.h

- str should contain your customized error message to which a system
  error message will be appended
- allows you to generate customized error messages, returning a
  pointer to a null-terminated string containing an error message
- if str is NULL, the return value contains the most recently
  generated system error message; this string is null-terminated
- if str is not NULL, the return value contains, a colon, a space,
  the most recently generated system error message, and a newline;
  with length of str being 94 characters or less
- this function generates the error string but does NOT print it
- for accurate error-handling, strerror should be called as soon
  as a library routine generates an error return

void swab(char *src, char *dest, int nbytes)

- prototype in stdlib.h

- copies nbytes bytes from src into dest, with adjacent even and
  odd-byte positions being swapped
- nbytes should therefore be an even number
- a standard way of performing this is to use the XCHG instruction

int system(const char *command)

- prototype in stdlib.h

- invokes MS-DOS COMMAND.COM to execute "command", returning exit
  status of COMMAND.COM; the COMSPEC environment variable is
  automatically searched if necessary
- a CR/LF pair in the command string will result in an error
- see exec... spawn...

double tan(double x)

- prototype in math.h

- returns tangent of x
- x must be in radians

double tanh(double x)

- prototype in math.h

- returns hyperbolic tangent of x
- x must be in radians

MSC: char *tempnam(char *dir, char *prefix) MSC: char *tmpnam(char *string) - prototype in stdio.h - string = buffer to receive temporary filename - tempnam() creates a temporary file in the specified directory "dir" using the "prefix" as the beginning of the filename; memory for the resulting filename is allocate via malloc() and must be freed with free() - tmpnam() creates the temporary file in the current directory - return a pointer to the new name or NULL on error

time_t time(time_t *tloc)

- prototype in time.h

- gives current time in seconds elapsed since 00:00:00 GMT, January 1, 1970, and stores it into tloc

int toascii(int c)

- prototype in ctype.h
- clears all but lower 7 bits in c, converting to ASCII

void tzset(void)

- prototype in time.h
- included for UNIX compatibility but does nothing in DOS

char *ultoa(unsigned long value, char *string, int radix)

- prototype in stdlib.h

- converts value to string where radix specifies base (2-36) for conversion

MSC: int umask(int permission)

```
        - prototype in io.h, sys\types.h, sys\stat.h

        - changes permision in which a file may be accessed
        - permission    = S_IWRITE      write is forbidden
                        = S_IREAD       read is forbidden  (ignored
by DOS)
                        = S_IWRITE | S_IREAD  no reading or writing

        - returns previous permission value
```

int ungetc(int c, FILE *fp)

- prototype in stdio.h
- pushes character "c" back into file stream "fp"

TC: void unixtodos(long utime, struct date *dateptr, struct time *timeptr) - prototype in dos.h - converts date & time to DOS format - no corresponding MS C function

MSC: int utime(char *path, struct utimbuf *times)

```
        - prototype in sys\types.h, sys\utime.h

        - path = any valid DOS filename
        - times = time values in structure (modtime should be set)
        - if write access to the file is available the file time is
updated
          from the modtime field
        - if times is NULL the file date is set to current time
        - returns 0 on success or -1 on error

        struct utimbuf
            {
            time_t actime;       - access time (not used in DOS)
            time_t modtime;      - modification time
            };
```

int vfprintf(FILE *stream, const char *format, va_list param)

- prototype in stdio.h
- sends formatted output to a stream

int vfscanf(FILE *stream, const char *format, va_list argp)

- prototype in stdio.h

- like fscanf(), except arguments accepted from va_arg array from
  va_list parm

int vprintf(const char *format, va_list param)

- prototype in stdio.h
- sends formatted output to stdout

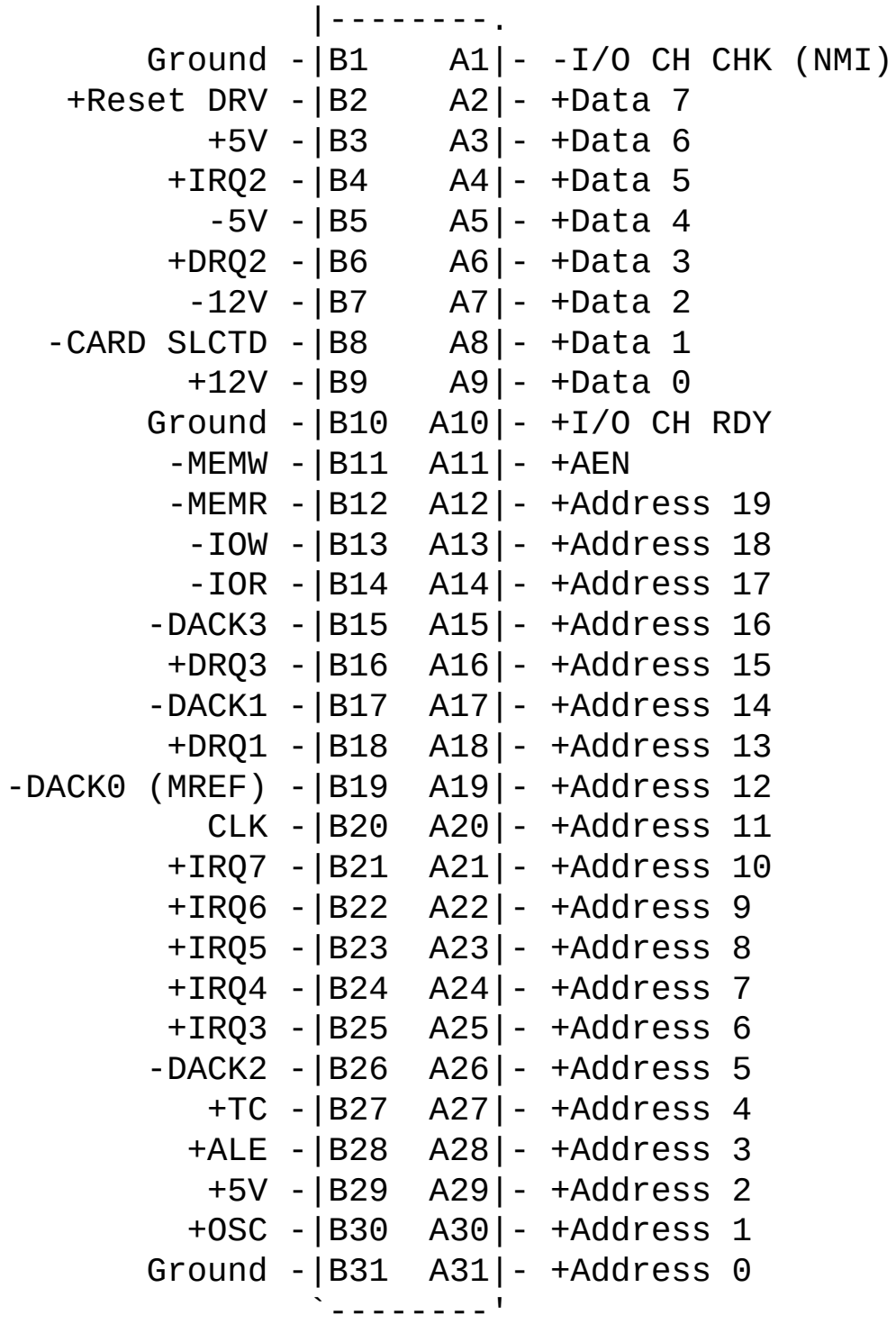
int vscanf(const char *format, va_list argp)

- prototype in stdio.h
- like scanf(), except arguments accepted from va_arg array from
  va_list parm

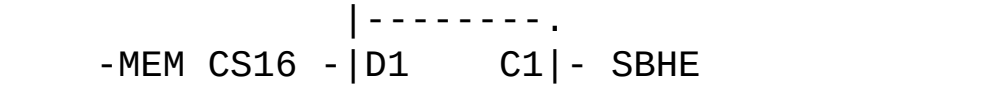
Hardware Data and Specifications

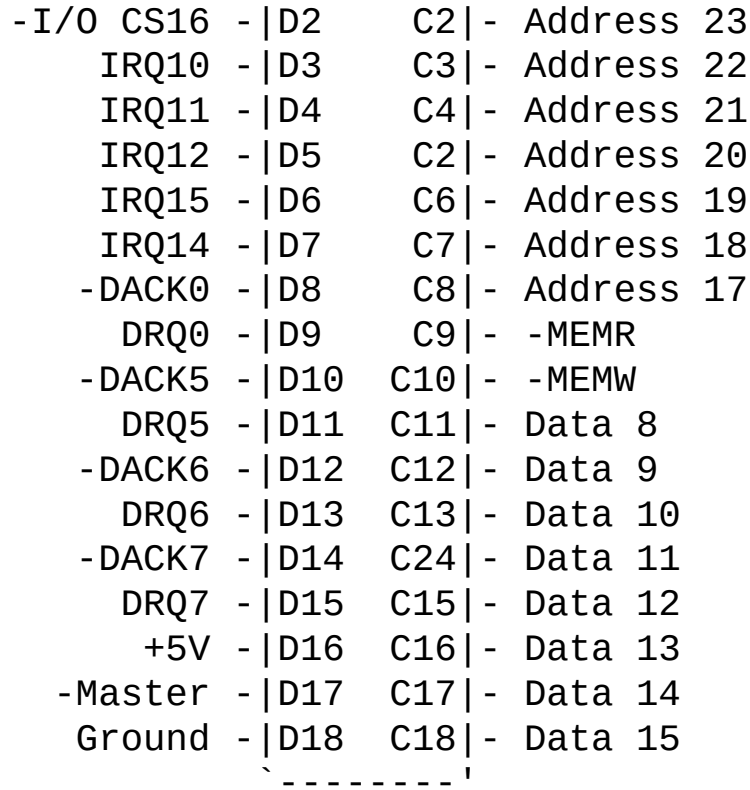
16450	16550	6845	765	8042
8250	8253	8254	8259	8272
BUS	capacitor values	CMOS clock	CMOS RAM	CRTC
diagnostic codes	disk drives	drive specifications	Epson printer codes	FDC
floppy controller	game port	hard disks drives	hard drives	hardware ports
Hayes modem info	HP Laserjet	joystick	keyboard commands	keyboard controller
Laserjet	modem commands	NEC 765	parallel port	PIC
PIT	port addresses	ports	POST errors	printer port
resistor values	RS232 pins	RTC	serial ports	timer
UART				

PC, XT and AT 8 bit BUS Structure



AT 16 Bit BUS Extension





- pin numbering starts from the rear of the machine

Signal	Description
A0-A19	Address Bits 0-19 allow access to 1Mb memory and 64K of port addresses.
A17-A23	Address Bits 17-23 allow access from 1Mb memory to 16Mb
AEN	Address Enable; When active the DMA controller has control of the Address and Data BUS as well as the MEMR/MEMW lines. When inactive the CPU has control of these lines
ALE	Address Latch Enable (output); used to latch addresses from the CPU. Forced active during DMA cycles.
CARD SLCTD	Card Selected; activated by cards in the XT's slot 8
CLK	System clock signal (actual BUS speed)
D0-D7	Data bits 0-7 for I/O to memory and I/O
DACK0-DACK3	DMA Acknowledge for channels 0-3; used by the controller to acknowledge DMA requested. DACK0 is used

for memory
refresh (MREF)
DRQ0-DRQ3 DMA Request 0-3; used by peripherals to get service from
the DMA controller; Held active until the corresponding
DACKx signal becomes active.
I/O CH CHK I/O Channel Check; Generates a Non Maskable Interrupt
I/O CH RDY I/O Channel Ready; pulled inactive my memory or I/O
devices to lengthen memory or I/O cycles. Usually used
by slower devices to add wait states. Should not be
held inactive for more than 17 cycles.
I/O CS16 I/O Chip Select 16 Bit; 16 bit I/O cycle
IOR I/O Read; instructs an I/O device to drive its data
onto the system BUS
IOW I/O Write; instructs an I/O device to read data from
the BUS
IRQ2-IRQ7 Interrupt Requests 2-7; signals the CPU that an I/O
device needs service (see 8259)
MASTER Used with DRQ to gain control of system
MEM CS16 Memory Chip Select 16 Bit; 16 Bit memory cycle
MEMR Memory Read; this signal is driven by the CPU or DMA
controller and instructs memory to drive its data onto
the system BUS. Present on both PC and AT extension BUS
MEMW Memory Write; this signal is driven by the CPU or DMA
controller and instructs memory to read and store data
from the system BUS. Present on both PC and AT extension BUS
OSC Oscillator; 14.31818 MHz clock (70ns period); 50% duty
cycle
RESET DRV Reset Drive; used to reset system logic
SBHE System BUS High Enable; activates data bits 8-15 on AT

extension BUS

TC Terminal Count; pulses when the terminal count for a DMA channel is reached

- all ISA BUS signals use standard TTL levels
- input and output are relative to the CPU

Ceramic File Capacitor Color Code Chart

Color	Digit	Multiplier		
Black	0	1		
Brown	1	10		
Red	2	100	Gold	ñ 5% tolerance
Orange	3	1,000	Silver	ñ10% tolerance
Yellow	4	10,000	White	ñ10% tolerance
Green	5	100,000		
Blue	6	1,000,000		
Violet	7			
Gray	8			
White	9			

ÕÍÛÛÛÛ°ÛÛ±ÛÛ²ÛÛ°ÛÛ±ÛÛÛÛÍ¸

```
-                         -
      |  |  |  |  |
      |  |  |  `---------- tolerance/temperature dependance
      |  |  `------------ picofarad value multiplier
      `---------------- first 2 digits of picofarad value
```

Hard Drive Specifications

Manufacturer	MB	Cyl	Hds	Access	Avg	Miscellaneous Information
Alloy ID-160		125			30	
Atasi AT3020			635	3		Wedge servo
Atasi AT3033			635	5		Wedge servo
Atasi AT3046			635	7		
Atasi AT3051			703	7		
Atasi AT3085			1024	8		
Bull D530			987	3		
Bull D550			987	5		
Bull D570			987	7		
Bull D585			987	7		
Bull D530			1166	3		
CDC Wren I 9415-21			697	3		STnn, MFM, Voice coil
CDC Wren I 9415-36			697	5		STnn, MFM, Voice coil
CDC Wren II 9415-25			615	4		STnn, MFM, Voice coil
CDC Wren II 9415-30			989	3		STnn, MFM, Voice coil
CDC Wren II 9415-38			733	5		STnn, MFM, Voice coil
CDC Wren II 9415-48			925	5		STnn, MFM, Voice coil
CDC Wren II 9415-51		42	989	5	28	STnn, MFM, Voice coil
CDC Wren II 9415-67			925	7		STnn, MFM, Voice coil
CDC Wren II 9415-86		69	925	9		STnn, MFM, Voice coil
CDC Wren II 9416-182		144	969	9		STnn, MFM, Voice coil
CDC Wren V			380	15	16	ESDI/SCSI, FH
CDC Wren V			702	15	16	ESDI/SCSI, FH
CDC Wren VI			766	15	16.5	ESDI/SCSI, FH
CDC Wren VII		1.2G		15	16.5	SCSI, FH
CMI CM3426			612	4		
CMI CM5205			256	4		
CMI CM5410			256	4		

CMI CM5616		256	4		
CMI CM6426		615	4		
CMI CM6426S		640	4		
CMI CM6640		640	4		
CMS F40-K	42	1024	5	22	STnn, MFM, Voice coil
CMS F60-K	42	1024	7		STnn, MFM, Voice coil
CMS F70-K	42	1024	8		STnn, MFM, Voice coil
CMS F80-K	42	1024	9		STnn, MFM, Voice coil
Columbia SCSI	42	834	3	19	SCSI, RLL, Band stepper
Core AT40F	40	564	4	10	ESDI, MFM, Voice coil
Core AT43	43	988	5	20	STnn, MFM, Voice coil
Core HC150	150	1024+		18	ESDI
Emerald DOS 150-3000	150	1024+			ESDI
Emulex ATS-170	142	1022			SCSI & ESDI
Emulex ATS-380	310	1222			SCSI & ESDI
FlashCard 49	49	615	6	28	STnn, RLL, Voice coil
Fuji 302-13	10	612	2		
Fuji 302-26	20	612	4		
Fujitsu M2230AS		320	2		
Fujitsu M2233AS	10	320	4		
Fujitsu M2234AS		320	6		
Fujitsu M2235AS	21	320	8	56	
Fujitsu M2241AS		754	4		
Fujitsu M2242AS	43	754	7	33	STnn, MFM, Voice coil
Fujitsu M2243AS	72	754	11	33	STnn, MFM, Voice coil
Fujitsu M2263	778u		14	16	ESDI/SCSI, FH
Hitachi DK511-5	699		7		
Hitachi DK511-8	823		10		
Hitachi DK515C-78	780u		8	16	SCSI, FH
Hitachi DK515-78	780u		8	16	ESDI, FH
IMI 5006H		306	2		
IMI 5012H		306	4		
IMI 5018H		306	6		
Imprimis (see CDC)					
Irwin 416		819	2		
Irwin 510		628	2		HD/tape

Irwin 516		819	2		HD/tape
Irwin 561		180	4		
Kalok KL320	26u	615	4	40	STnn, MFM, Band stepper
Kalok KL330	38u	615	4	40	STnn, RLL, Band stepper
Maxtor XT1065	52	918	7	28	STnn, Voice coil
Maxtor XT1085	68	1024	8	28	STnn, Voice coil
Maxtor XT1105	83	918	11	28	STnn, Voice coil
Maxtor XT1140	112	918	15	28	STnn, Voice coil
Maxtor XT1190	150	1224		28	STnn, Voice coil
Maxtor XT2085		1224	7	28	STnn, Voice coil
Maxtor XT2140		1224	11		STnn, Voice coil
Maxtor XT2190	160	1224	15		STnn, Voice coil
Maxtor XT4170E	179u	1224	7	14	ESDI, MFM, FH Voice coil
Maxtor XT4230E	231u	1224	9	16	ESDI, MFM, FH Voice coil
Maxtor XT4380E	384u	1224	15	16	ESDI, MFM, FH Voice coil
Maxtor XT4380S	384u	1224		16	SCSI, MFM, FH Voice coil
Maxtor XT8380E	410u			14.5	ESDI, FH
Maxtor XT8380S	410u			14.5	SCSI, FH
Maxtor XT8760E	768u			16.5	ESDI, FH
Maxtor XT8760S	768u			16.5	SCSI, FH
Micropolis 1302		830	3		
Micropolis 1303		830	5		
Micropolis 1304		830	6		
Micropolis 1323		1024	4		
Micropolis 1323A		1024	5		
Micropolis 1324		1024	6		
Micropolis 1324A		1024	7		
Micropolis 1325		1024	8		
Micropolis 1333A	44	1024	5	28	STnn, MFM, Voice coil
Micropolis 1335	71	1024	8	30	
Micropolis 1355	142	1022			
Micropolis 1518	1.2Gu		15	14	ESDI, FH
Micropolis 1558	382u		15	18	ESDI, FH
Micropolis 1568	765u		15	16	ESDI, FH
Micropolis 1578	382u		15	18	SCSI, FH
Micropolis 1588	765u		15	16	SCSI, FH
Micropolis 1598	1.2Gu		15	14	SCSI, FH
Microscience HH312	10	306	4		
Microscience HH325	20	612	4		

Microscience HH612	10	306	4		
Microscience HH725	21	612	4		Band stepper
Microscience HH1050	44	1024	5	28	STnn, MFM, Voice coil
Miniscribe 1006		306	2		
Miniscribe 1012		306	4		
Miniscribe 2006		306	2		
Miniscribe 3012	10	612	4		
Miniscribe 3053	44	1024	6	25	STnn, MFM, Voice coil
Miniscribe 3212	10	612	4		
Miniscribe 3412		306	4		
Miniscribe 3425	20	615	4		
Miniscribe 3438	31.2	615	4		
Miniscribe 3650	50u	809	6	61	STnn, MFM, linear drive
Miniscribe 3650F	50u	809	6	46	STnn, MFM, linear drive
Miniscribe 3675	75u	809	6	61	STnn, MFM, linear drive
Miniscribe 4010		480	2		
Miniscribe 4020		480	4		
Miniscribe 6032	26	1024	3		
Miniscribe 6053	43	1024	5	28	STnn, MFM, FH Voice coil
Miniscribe 6085	71	1024	8		
Miniscribe 6128	104	1024	8		
Miniscribe 8051A	51u	745	4	28	STnn, MFM, Voice coil
Miniscribe 8212		615	2		
Miniscribe 8425	20	615	4		STnn, MFM, 3«
Miniscribe 8438	31.2	615	4		
Miniscribe 9380E	382u	1224	15	16	ESDI, RLL, FH Voice coil
Miniscribe 9380S	382u	1224	15	16	SCSI, RLL, FH Voice coil
Miniscribe 9780E	781u		15	17	ESDI, FH Voice coil
Mitsubisi MR535	42	981	6		STnn, MFM, 5¬ Voice coil
NEC D5662	385u			18	ESDI, FH
NEC D5682	765u		18	16	ESDI, FH
NEC D5862	385u			18	SCSI, FH
NEC D5882	765u		18	16	SCSI, FH
NEC 5124		310	4		
NEC 5126		612	4		

NEC 5146		615	8		
Newberry Penny 340		615	8		
Northgate Turbo	42	809	6	36	STnn, RLL, Voice coil
Plus Hardcard 40	42	612	4	40	STnn, RLL, Voice coil
Priam ID45ATD2	42	1166	5	20	STnn, MFM, Voice coil
Priam ID130	130	1224			STnn
Priam ID40		987	5		
Priam ID60		987	7		
Priam V130		987	3		STnn
Priam V185		1166	7		
Priam 638	382u		15	18	ESDI, 5¬
Priam 676	765u		15	14	ESDI, 5¬
Priam 738	382u		15	18	SCSI, 5¬
Priam 776	765u		15	14	SCSI, 5¬
Quadram Q520		512	4		
Quadram Q530		512	6		
Quadram Q540		512	8		
Rodime RO101		192	2		
Rodime RO102		192	4		
Rodime RO103		192	6		
Rodime RO104		192	8		
Rodime RO201		320	2		
Rodime RO202		320	4		
Rodime RO203		320	6		
Rodime RO204		320	8		
Rodime RO201E		640	2		
Rodime RO202E		640	4		
Rodime RO203E	33	640	6		
Rodime RO204E		640	8		
Rodime RO252	10	306	4		
Rodime RO352	10	306	4		
Rodime RO3055	45	872	7		STnn, MFM, Voice coil
RMS 503		153	2		
RMS 506		153	4		
RMS 512		153	8		
Seagate ST125	21	615	4	28	STnn, MFM, Band stepper
Seagate ST125N	22	407	4		SCSI, RLL, Band stepper
Seagate ST138	32	615	6	28	STnn, MFM, 3« Band stepper
Seagate ST138N	31	615	4	28	SCSI, RLL, Band

stepper					
Seagate ST138R	33	615	4	28	STnn, RLL, Band
stepper					
Seagate ST151	43	977	5		STnn, MFM, Voice
Coil					
Seagate ST157A	49	560	6	28	IDE, 3«
Seagate ST157N	49	615	6	28	SCSI, 3« RLL,
Band stepper					
Seagate ST157R	49	615	6	28	STnn, 3« RLL,
Voice Coil					
Seagate ST177N	61	921	5		SCSI, RLL, Voice
Coil					
Seagate ST206		306	2		STnn, MFM, Band
stepper					
Seagate ST213	11	615	2		STnn, MFM, Band
stepper					
Seagate ST225	21	615	4	65	STnn, MFM, 5¬
Band stepper					
Seagate ST225N	21	615	4	65	SCSI, MFM, 5¬
Band stepper					
Seagate ST225R	21	667	2		STnn, RLL, 5¬
Band stepper					
Seagate ST238R	31	615	4	65	STnn, RLL, Band
Stepper					
Seagate ST250R	42	667	4		STnn, RLL, Band
stepper					
Seagate ST251	42	820	6	40	STnn, MFM, 5¬
Band stepper					
Seagate ST251-1	43	820	6	28	STnn, MFM, 5¬
Band stepper					
Seagate ST251N0	43	820	4	40	SCSI, RLL, 5¬
Band stepper					
Seagate ST251N1	43	630	4		SCSI, RLL, 5¬
Band stepper					
Seagate ST251R	43	820	6	40	STnn, RLL, 5¬
Band stepper					
Seagate ST277	66	820	6		STnn, MFM, 5¬
Band stepper					
Seagate ST277N0	65	820	6	40	SCSI, RLL, 5¬
Band stepper					
Seagate ST277N1	65	628	6		SCSI, RLL, 5¬
Band stepper					
Seagate ST277R	65	820	6	40	STnn, RLL, 5¬
Band stepper					
Seagate ST280A	72			29	IDE, 5¬
Seagate ST296N	85	820	6	28	SCSI, RLL, 5¬

Band stepper					
Seagate ST406	5	306	2		STnn, MFM, Band stepper
Seagate ST412	10	306	4		STnn, MFM, FH Band stepper
Seagate ST419	15	306	6		STnn, MFM, Band stepper
Seagate ST425	20	306	8		STnn, MFM, Band stepper
Seagate ST506	5	153	4		STnn, MFM, FH Band stepper
Seagate ST1096N	84	906	7		SCSI, RLL, Voice coil
Seagate ST1102A	89			29	IDE, 3«
Seagate ST1144	125			19	IDE, 3«
Seagate ST1239	211			15	IDE, 3«
Seagate ST4026	21	615	4		STnn, MFM, Voice coil
Seagate ST4038	32	733	5	40	STnn, MFM, FH Voice coil
Seagate ST4051	43	977	5	40	STnn, MFM, Voice coil
Seagate ST4053	44	1024	5	28	STnn, MFM, Voice coil
Seagate ST4077R	65	1024	5	28	STnn, MFM, Voice coil
Seagate ST4096	80	1024	9	28	STnn, MFM, FH Voice coil
Seagate ST4144R	122	1024	9	28	STnn, RLL, FH Voice coil
Shugart SA 604		160	4		
Shugart SA 606		160	6		
Shugart SA 612		306	4		
Shugart SA 712		320	4		
SPC Scorecard 44	44	753	7		STnn, MFM, Voice coil
Syquest SQ306RD		306	2		
Syquest SQ312RD		615	2		
Syquest SQ325F		612	4		
Syquest SQ338F		612	6		
Tandon TM252	10	306	4		
Tandon TM262	20	615	4		
Tandon TAN501		306	2		
Tandon TAN502		306	4		
Tandon TAN503		306	6		
Tandon TM602S		153	4		

Tandon TM603S		153	6		
Tandon TM603SE		230	6		
Tandon TM702AT		615	4		
Tandon TM703		695	5		
Tandon TM703AT		733	5		
Tandon TM755		981	5		
Tandy SCSI	80	823	6	28	SCSI, RLL, Band stepper
Toshiba MK53F		830	5		
Toshiba MK54F	49	830	7	25	STnn, MFM, 3« Voice coil
Toshiba MK56F	72	830	10	25	
Toshiba MK134FA		733	7	25	STnn
Toshiba MK358FA	765u		15	16	ESDI, FH
Toshiba MK358FB	765u		15	16	SCSI, FH
Tulin 226		640	4		
Tulin 240		640	6		
Tulin 326	20	640	4		
Tulin 340	30	640	6		
WD93024A	21.6			28	STnn, MFM, 3« Band stepper
WD93024X	21.6			39	STnn, MFM, 3« Band stepper
WD93028A	21.6			69	STnn, MFM, 3« Band stepper
WD93028X	21.6			70	STnn, MFM, 3« Band stepper
WD93034X	32.4			39	STnn, MFM, 3« Band stepper
WD93038X	32.4			70	STnn, MFM, 3« Band stepper
WD93044A	43.2			28	STnn, MFM, 3« Band stepper
WD93044X	43.2			39	STnn, MFM, 3« Band stepper
WD93048A	43.2			69	STnn, MFM, 3« Band stepper
WD93048X	43.2			70	STnn, MFM, 3« Band stepper
WD95024A	20			28	STnn, MFM, 5¬ Band stepper
WD95028AD	20			69	STnn, MFM, 5¬ Band stepper
WD95028X	20			70	STnn, MFM, 5¬ Band stepper
WD95038X	30			70	STnn, MFM, 5¬

Band stepper			
WD95044A	40	28	STnn, MFM, 5¬
Band stepper			
WD95048AD	40	69	STnn, MFM, 5¬
Band stepper			
WD95048X	40	70	STnn, MFM, 5¬
Band stepper			
WDAC140	42.5	19	STnn, MFM, 3«
Voice coil			
WDAC280	85.1	19	STnn, MFM, 3«
Voice coil			
WDSC320	320		SCSI, 3«
WDSC8320	320		SCSI, 3«

- STnn indicates Seagate ST412/ST506 compatible
- 3« indicates 3.5 inch half height drive
- 5¬ indicates 5.25 inch half height drive
- FH indicates Full Height 5.25 inch drive
- 'u' in filesize indicates unformatted spec
- formatted capacity is related to BIOS and may vary with different BIOS and OEM versions
- read/write heads should normally never touch hard disk media
- standard AT hard disk controllers do not use DMA for disk I/O due to the slow speed of the 8 Bit DMA chips used; Port I/O is used instead for an actual increase in performance
- with some XT and ESDI systems it is necessary to use debug to execute the ROM disk format code; The usual method is to set AH=Drive, AL=Interleave and jump to the code at C800:5 via the G =C800:5 command (some systems locate this code at C600:5, CA00:5 and CC00:5)
- DOS version and disk size determine cluster size; DOS 2.x always has cluster size of 8K, while DOS 3.x+ has variable cluster sizes:

2 - 16 Mb partitions have a cluster size of 8K
16 - 128 Mb partitions have a cluster size of 2K

```
128 - 256 Mb partitions have a cluster size of 4K
256 - 512 Mb partitions have a cluster size of 8K
```

Epson FX Printer Codes

Printer Operation:

Decimal	ASCII	Description
7	BEL	Beeper
17	DC1	Select printer
19	DC3	Deselect printer
27 25 48	ESC EM 0	Turn cut sheet feeder control off
27 25 52	ESC EM 4	Turn cut sheet feeder control on
27 56	ESC 8	Disable paper out sensor
27 57	ESC 9	Enable paper out sensor
27 60	ESC <	Select unidirectional mode for one line
27 64	ESC @	Initialize printer
27 85 48	ESC U 0	Cancel unidirectional mode
27 85 49	ESC U 1	Select unidirectional mode
27 115 48	ESC s 0	Turn half speed mode off
27 115 49	ESC s 1	Turn half speed mode on

Vertical/Horizontal Motion:

Decimal	ASCII	Description
8	BS	Backspace
9	HT	Horizontal tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
27 47 c	ESC / c	Select vertical tab channel (c=0..7)
27 48	ESC 0	Select 8 lines per inch
27 49	ESC 1	Select 7/72 inch line spacing
27 50	ESC 2	Select 6 lines per inch
27 51 n	ESC 3 n	Select n/216 inch line spacing (n=0..255)
27 65 n	ESC A n	Select n/72 inch line spacing (n=0..85)
27 66 0	ESC B NUL	Clear Vertical tabs
27 66 tabs	ESC B tabs	Select up to 16 vertical tabs where tabs are ascending values from 1..255 ending with NUL
27 67 n	ESC C n	Select page length in lines (n=1..127)
27 67 48 n	ESC C 0 n	Select page length in inches

(n=1..22)
27 68 0 ESC D NUL Clears all horizontal tables
27 68 tabs 0 ESC D tabs NUL Sets up to 32 horizontal tabs with
ascending values 1-137. NUL or a value
less than previous tab ends command.
27 74 n ESC J n Immediate n/216 inch line feed (n=0..255)
27 78 n ESC N n Select skip over perforation (n=1..127)
27 79 ESC O Cancel skip over perforation
27 81 n ESC Q n Set right margin (n=column)
27 98 b c 0 ESC b c NUL Clear vertical tabs in channel (c=0..7)
27 98 c tabs ESC b c tabs Select up to 16 vertical tabs in channels
(c=0..7) where tabs are ascending values
from 1..255 ending with NUL
27 101 48 s ESC e 0 s Set horizontal tab to increments of 's'
27 101 49 s ESC e 1 s Set vertical tab to increments of 's'
27 102 48 s ESC f 0 s Set horizontal skip to increments of 's'
27 102 49 s ESC f 1 s Set vertical skip to increments of 's'
27 106 n ESC j n Reverse linefeed (n/216 inch after buffer)
27 108 n ESC l n Set left margin (n=column)

Printing Style:

Decimal	**ASCII**	**Description**
27 33 n	ESC ! n	Master select where n is a combination of:

0	Pica	16	Double Strike
1	Elite	32	Double Wide
4	Condensed	64	Italic
8	Emphasized	128	Underline

Pica & Elite and Condensed/ Emphasized are

			mutually exclusive
	27 107 48	ESC k 0	Select NLQ Roman font
	27 107 49	ESC k 1	Select NLQ Sans Serif font
	27 120 48	ESC x 0	Select draft mode
	27 120 49	ESC x 1	Select NLQ mode

Print Size and Character Width:

Decimal	ASCII	Description
14	SO	Select double width for one line
15	SI	Select condensed mode
18	DC2	Cancel condensed mode
20	DC4	Cancel one line double width mode
27 14	ESC SO	Double width for one line (duplicate)
27 15	ESC SI	Select condensed mode (duplicate)
27 77	ESC M	Select elite width (12 cpi)
27 80	ESC P	Select pica width (10 cpi)
27 87 48	ESC W 0	Cancel double width mode
27 87 49	ESC W 1	Select double width mode

Print Enhancement:

Decimal	ASCII	Description
27 45 48	ESC - 0	Cancel underlining
27 45 49	ESC - 1	Select underlining
27 69	ESC E	Select emphasized mode
27 70	ESC F	Cancel emphasized mode
27 71	ESC G	Select double strike mode
27 72	ESC H	Cancel double strike mode
27 83 48	ESC S 0	Select superscript
27 83 49	ESC S 1	Select subscript
27 84	ESC T	Cancel superscript/subscript

Character Sets:

Decimal	ASCII	Description
27 52	ESC 4	Select italic mode
27 53	ESC 5	Cancel italic mode
27 54	ESC 6	Enable printing of characters (128-159,255)
27 55	ESC 7	Cancel [ESC 6] command
27 82 n	ESC R n	Select International character set where

numeric 'n' is:

0	USA	7	Spain I
1	France	8	Japan
2	Germany	9	Norway
3	United Kingdom	10	

Denmark II
4 Denmark I 11 Spain II
5 Sweden 12 Latin America
6 Italy

27 116 0 ESC t NUL Select italic character set
27 116 1 ESC t SOH Select Epson character set

User Defined Characters:

Decimal	ASCII	Description
27 37 0	ESC % NUL	Selects normal character set
27 37 1	ESC % SOH	Selects user defined set
27 38 0	ESC & NUL ?	Select user defined chars (see manual)
27 58 0 0 0	ESC : NUL NUL NUL	Copy ROM into RAM

Graphics Character Sets:

Decimal	ASCII	Description
27 42 0 n1 n2	ESC * NUL n1 n2	Select single density graphics
27 42 1 n1 n2	ESC * SOH n1 n2	Select double density graphics
27 63 s n	ESC ? s n	Reassign graphics mode 's'=(K,L,Y or Z) to mode 'n'=(0..6)
27 75 n1 n2	ESC K n1 n2	Single density graphics (60 dpi)
27 76 n1 n2	ESC L n1 n2	Double density graphics (120 dpi)
27 89 n1 n2	ESC Y n1 n2	Hi-speed double den graphics (120 dpi)
27 90 n1 n2	ESC Z n1 n2	Quad density graphics (240 dpi)
27 94 m n1 n2	ESC ^ m n1 n2	Select 9 pin graphics mode

number of columns = n1 + (n2 * 256)

Other:

Decimal	ASCII	Description
13	CR	Carriage Return
24	CAN	Cancel text in line (but not control codes)
127	DEL	Delete character (but not control codes)
27 32 n	ESC SP n	Space in n/72 inch following each

NLQ char

27 35	ESC #	MSB control sequence cancel
27 36	ESC $	Select absolute dot position
27 61	ESC =	MSB = 0
27 62	ESC >	MSB = 1
27 73 48	ESC I 0	Cancel above [ESC I 1]
27 73 49	ESC I 1	Printable codes expansion (0-31,128-159)
27 92	ESC \	Select relative dot position
27 97 n	ESC a n	NLQ justification where numeric 'n' is: 0 left justification (default) 1 center 2 right justification 3 full justification
27 112	ESC p	Select/cancel proportional mode

- the codes listed are relative to the Epson LX 800
- in several situations where a numeric value of zero or one is required, the ASCII value of the number can be substituted

PORTS Common I/O Port Addresses

Port addresses are not always constant across PC, AT and PS/2
Unless marked, port addresses are relative to PC and XT only

000-00F 8237 DMA controller
000 Channel 0 address register
001 Channel 0 word count
002 Channel 1 address register
003 Channel 1 word count
004 Channel 2 address register
005 Channel 2 word count
006 Channel 3 address register
007 Channel 3 word count
008 Status/command register
009 Request register
00A Mask register
00B Mode register
00C Clear MSB/LSB flip flop
00D Master clear temp register
00E Clear mask register
00F Multiple mask register

010-01F 8237 DMA Controller (PS/2 model 60 & 80), reserved (AT)

020-02F 8259A Master Programmable Interrupt Controller
020 8259 Command port (see 8259)
021 8259 Interrupt mask register (see 8259)

030-03F 8259A Slave Programmable Interrupt Controller (AT, PS/2)

040-05F 8253 or 8254 Programmable Interval Timer (PIT, see ~8253~)
040 8253 channel 0, counter divisor
041 8253 channel 1, RAM refresh counter
042 8253 channel 2, Cassette and speaker functions
043 8253 mode control (see 8253)
044 8254 PS/2 extended timer
047 8254 Channel 3 control byte

060-067 8255 Programmable Peripheral Interface (PC,XT, PCjr)
060 8255 Port A keyboard input/output buffer (output PCjr)
061 8255 Port B output
062 8255 Port C input
063 8255 Command/Mode control register

060-06F 8042 Keyboard Controller (AT,PS/2)
060 8042 Keyboard input/output buffer register
061 8042 system control port (for compatability with 8255)
064 8042 Keyboard command/status register

070 CMOS RAM/RTC, also NMI enable/disable (AT,PS/2, see RTC)
071 CMOS RAM data (AT,PS/2)

080 Manufacturer systems checkpoint port (used during POST)
080-090 DMA Page Registers
081 High order 4 bits of DMA channel 2 address
082 High order 4 bits of DMA channel 3 address
083 High order 4 bits of DMA channel 1 address

090-097 POS/Programmable Option Select (PS/2)
090 Central arbitration control Port
091 Card selection feedback
092 System control and status register
094 System board enable/setup register
095 Reserved
096 Adapter enable/setup register
097 Reserved

0A0 NMI Mask Register (PC,XT) (write 80h to enable NMI, 00h disable)
0A0-0BF Second 8259 Programmable Interrupt Controller (AT, PS/2)
0A0 Second 8259 Command port (see 8259)
0A1 Second 8259 Interrupt mask register (see 8259)

0C0 TI SN76496 Programmable Tone/Noise Generator (PCjr)
0C0-0DF 8237 DMA controller 2 (AT)
0C2 DMA channel 3 selector (see ports 6 & 82)

0E0-0EF Reserved

0F0-0FF Math coprocessor (AT, PS/2)
0F0-0F5 PCjr Disk Controller
0F0 Disk Controller

0F2 Disk Controller control port
0F4 Disk Controller status register
0F5 Disk Controller data port

0F8-0FF Reserved for future microprocessor extensions

100-10F POS Programmable Option Select (PS/2)
100 POS Register 0, Adapter ID byte (LSB)
101 POS Register 1, Adapter ID byte (MSB)
102 POS Register 2, Option select data byte 1
    Bit 0 is card enable (CDEN)
103 POS Register 3, Option select data byte 2
104 POS Register 4, Option select data byte 3
105 POS Register 5, Option select data byte 4
    Bit 7 is (-CHCK)
    Bit 6 is reserved
106 POS Register 6, subaddress extension (LSB)
107 POS Register 7, subaddress extension (MSB)

110-1EF System I/O channel

170-17F Fixed disk 1 (AT)
170 disk 1 data
171 disk 1 error
172 disk 1 sector count
173 disk 1 sector number
174 disk 1 cylinder low
175 disk 1 cylinder high
176 disk 1 drive/head
177 disk 1 status

1F0-1FF Fixed disk 0 (AT)
1F0 disk 0 data
1F1 disk 0 error
1F2 disk 0 sector count
1F3 disk 0 sector number
1F4 disk 0 cylinder low
1F5 disk 0 cylinder high
1F6 disk 0 drive/head
1F7 disk 0 status

200-20F Game Adapter (see GAME PORT or ~JOYSTICK~)

210-217 Expansion Card Ports (XT)
210 Write: latch expansion bus data
    read: verify expansion bus data

211 Write: clear wait,test latch
    Read: MSB of data address
212 Read: LSB of data address
213 Write: 0=enable, 1=/disable expansion unit
214-215 Receiver Card Ports
214 write: latch data, read: data
215 read: MSB of address, next read: LSB of address

21F Reserved

220-26F Reserved for I/O channel

270-27F Third parallel port (see ~PARALLEL PORT~)
278 data port
279 status port
27A control port

280-2AF Reserved for I/O channel

2A2-2A3 MSM58321RS clock

2B0-2DF Alternate EGA, or 3270 PC video (XT, AT)

2E0 Alternate EGA/VGA
2E1 GPIB Adapter (AT)

2E2-2E3 Data acquisition adapter (AT)

2E8-2EF COM4 non PS/2 UART (Reserved by IBM) (see ~UART~)

2F0-2F7 Reserved

2F8-2FF COM2 Second Asynchronous Adapter (see UART)
         Primary Asynchronous Adapter for PCjr

300-31F Prototype Experimentation Card (except PCjr)
         Periscope hardware debugger
320-32F Hard Disk Controller (XT)
320 Read from/Write to controller
321 Read: Controller Status, Write: controller reset
322 Write: generate controller select pulse
323 Write: Pattern to DMA and interrupt mask register
    (see ports 0F,21,C2)
324 disk attention/status

330-33F Reserved for XT/370

340-35F Reserved for I/O channel

360-36F PC Network

370-377 Floppy disk controller (except PCjr)
372 Diskette digital output
374 Diskette controller status
375 Diskette controller data
376 Diskette controller data
377 Diskette digital input

378-37F Second Parallel Printer (see ~PARALLEL PORT~)
First Parallel Printer (see PARALLEL PORT)
378 data port
379 status port
37A control port

380-38F Secondary Binary Synchronous Data Link Control (SDLC) adapter
380 On board 8255 port A, internal/external sense
381 On board 8255 port B, external modem interface
382 On board 8255 port C, internal control and gating
383 On board 8255 mode register
384 On board 8253 channel square wave generator
385 On board 8253 channel 1 inactivity time-out
386 On board 8253 channel 2 inactivity time-out
387 On board 8253 mode register
388 On board 8273 read: status; Write: Command
389 On board 8273 write: parameter; read: response
38A On board 8273 transmit interrupt status
38B On board 8273 receiver interrupt status
38C On board 8273 data

390-39F Cluster Adapter

3A0-3AF Primary Binary Synchronous Data Link Control (SDLC) adapter
3A0 On board 8255 port A, internal/external sense
3A1 On board 8255 port B, external modem interface
3A2 On board 8255 port C, internal control and gating
3A3 On board 8255 mode register
3A4 On board 8253 counter 0 unused
3A5 On board 8253 counter 1 inactivity time-outs
3A6 On board 8253 counter 2 inactivity time-outs
3A7 On board 8253 mode register

3A8 On board 8251 data
3A9 On board 8251 command/mode/status register

3B0-3BF Monochrome Display Adapter (write only, see ~6845~)
3B0 port address decodes to 3B4
3B1 port address decodes to 3B5
3B2 port address decodes to 3B4
3B3 port address decodes to 3B5
3B4 6845 index register, selects which register [0-11h]
    is to be accessed through port 3B5
3B5 6845 data register [0-11h] selected by port 3B4,
    registers 0C-0F may be read. If a read occurs without
    the adapter installed, FFh is returned. (see 6845)
3B6 port address decodes to 3B4
3B7 port address decodes to 3B5
3B8 6845 Mode control register
3B9 reserved for color select register on color adapter
3BA status register (read only)
3BB reserved for light pen strobe reset

3BC-3BF Primary Parallel Printer Adapter (see ~PARALLEL PORT~)
3BC parallel 1, data port
3BD parallel 1, status port
3BE parallel 1, control port

3C0-3CF EGA/VGA
3C0 VGA attribute and sequencer register
3C1 Other video attributes
3C2 EGA, VGA, CGA input status 0
3C3 Video subsystem enable
3C4 CGA, EGA, VGA sequencer index
3C5 CGA, EGA, VGA sequencer
3C6 VGA video DAC PEL mask
3C7 VGA video DAC state
3C8 VGA video DAC PEL address
3C9 VGA video DAC
3CA VGA graphics 2 position
3CC VGA graphics 1 position
3CD VGA feature control
3CE VGA graphics index
3CF Other VGA graphics

3D0-3DF Color Graphics Monitor Adapter (ports 3D0-3DB are
        write only, see 6845)
3D0 port address decodes to 3D4

3D1 port address decodes to 3D5
3D2 port address decodes to 3D4
3D3 port address decodes to 3D5
3D4 6845 index register, selects which register [0-11h]
    is to be accessed through port 3D5
3D5 6845 data register [0-11h] selected by port 3D4,
    registers 0C-0F may be read. If a read occurs without
    the adapter installed, FFh is returned. (see 6845)
3D6 port address decodes to 3D4
3D7 port address decodes to 3D5
3D8 6845 Mode control register (CGA, EGA, VGA, except PCjr)
3D9 color select palette register (CGA, EGA, VGA, see 6845)
3DA status register (read only, see 6845, PCjr VGA access)
3DB Clear light pen latch (any write)
3DC Preset Light pen latch
3DF CRT/CPU page register (PCjr only)

3E8-3EF COM3 non PS/2 UART (Reserved by IBM) (see ~UART~)

3F0-3F7 Floppy disk controller (except PCjr)
3F0 Diskette controller status A
3F1 Diskette controller status B
3F2 controller control port
3F4 controller status register
3F5 data register (write 1-9 byte command, see INT 13)
3F6 Diskette controller data
3F7 Diskette digital input

3F8-3FF COM1 Primary Asynchronous Adapter (see ~UART~)

3220-3227 PS/2 COM3 (see UART)
3228-322F PS/2 COM4 (see UART)
4220-4227 PS/2 COM5 (see UART)
4228-422F PS/2 COM6 (see UART)
5220-5227 PS/2 COM7 (see UART)
5228-522F PS/2 COM8 (see UART)

- many cards designed for the ISA BUS only uses the lower 10 bits
  of the port address but some ISA adapters use addresses beyond
  3FF. Any address that matches in the lower 10 bits will decode
  to the same card. It is up to the adapters to resolve or ignore
  the high bits of the port addresses. An example would

be the
Cluster adapter that has a port address of 390h. The
second
cluster adapter has a port address of 790h which resolves
to
the same port address with the cards determining which one
actually gets the data.

Hayes Command Set / Register Formats

```
+++     standard escape sequence (see S2 below)
Comma   standard pause character (see S8 below)
AT      standard attention sequence

ATA     force immediate answer
ATC0    transmitter off
ATC1    transmitter on

ATD     dial (ATD number)
ATDP    pulse dial (ATDP number)
ATDT    tone dial (ATDT number)

ATE0    disable local character echo
ATE1    enable local character echo
ATF0    Half Duplex (modem echoes characters)
ATF1    Full Duplex (modem does not echo characters)

ATH0    force line on hook
ATH1    force line off hook
ATH2    force line special off hook (used for HAM radio)

ATI0    request product code, formatted PPR, PP=prod., R=rev.
ATI1    request ROM check sum

ATL1    speaker volume low  (modem specific)
ATL2    speaker volume medium  (modem specific)
ATL3    speaker volume high  (modem specific)
ATM0    speaker always off
ATM1    speaker on except while carrier present
ATM2    speaker always on
ATM3    speaker on except while dialing and carrier present

ATO     force modem into on-line state
ATP     sets modem to default pulse dial

ATQ0    modem returns result codes
ATQ1    modem does not return result codes

ATR     sets modem to answer mode after dialing out
ATSn    where Sn is S register number 'n' (see registers
below)
```

```
        ATT    sets modem to default tone dial

        ATV0   send numeric codes
        ATV1   send word result codes

        ATX0   basic result code set "CONNECT", no dial tone detect
        ATX1   extended result code set "CONNECT 1200", no dialtone
detect
        ATX2   wait for dial tone, extended result codes
        ATX3   detect busy signal, extended result codes
        ATX4   wait for dial tone, detect busy, extended result
codes

        ATZ    reset to power up condition
```

Hayes 2400 Compatible Modem Commands

```
        ATB0    CCITT V.22 at 1200 bps
        ATB1    BELL 212A at 1200 bps

        AT&C    DCD always on
        AT&C1   DCD on while carrier present

        AT&D    DTR ignored
        AT&D1   DTR fail disconnect enabled
        AT&D2   DTR fail disconnect enabled (auto answer off)
        AT&D3   DTR fail disconnect enabled (reset modem)

        AT&F    restore to factory configuration

        AT&G    no guard tone
        AT&G1   550 Hz guard tone
        AT&G2   1800 Hz guard tone

        AT&J    RJ11/RJ41S/RJ45 jack
        AT&J2   RJ12/RJ13 jack

        AT&L    regular phone line
        AT&L1   leased line

        AT&M    async operation
        AT&M1   async/sync operation
        AT&M2   sync auto dial
```

```
AT&M3   sync manual dial

AT&P    39/61 pulse make/break ratio
AT&P1   33/67 pulse make/break ratio

AT&T4   grant RDL test request
AT&T5   deny RDL test request

AT&W    write current registers to non-volatile memory

AT&X    sync clock internal

AT&X1   sync clock external
AT&X2   sync clock slaved

AT&Z    store dial command line
```

Hayes Compatible Modem Registers

Not all modems support all of these registers and some modems have registers other than those listed here. Also note, some are read only.

to set a register use: AT Sr=## (cr)
to read the register: AT Sr? (cr)

Where "r" is the register and (cr) is a carriage return

The following represent conventions used in the tables below:

() indicates defaults for Smartmodem 1200
** indicates possible inconsistencies across modems

Reg	Values	Register function
S0	0-255	ring to answer on (0=don't answer)
S1	0-255	ring count (clear after 8 sec) (read only)
S2	0-255	escape char, normally "+", 128-255 disable escape

```
        S3   0-127    end of line character (0x0D/CR)
        S4   0-127    line feed character (0x0A/LF)
        S5   0-32,127 backspace character (0x08/BS)
        S6   2-255    pause before dialing in seconds (2)
        S7   1-255    wait for carrier in seconds (30)
        S8   0-255    pause for comma in dial string in seconds (2)
        S9   1-255    carrier detect response time in 1/10 second
(6)
        S10  1-255    carrier loss delay in 1/10 second, 255=ignore
CD (7)
        S11 50-255    touch tone dial speed, in milliseconds (70)
        S12 20-255    escape guard time, in 1/50 second,0=no delay
(50)

        |7|6|5|4|3|2|1|0|  S13   power up async data format
         | | | | | | | `---- unused
         | | | | | | `----- result code, 0=basic, 1=extended
         | | | | | `------ parity, 0=disabled, 1=enabled
         | | | | `------- parity, 0=odd ,1=even
         | | | `-------- data bits, 0=7 bits, 1=8 bits
         | | `--------- undefined
         | `---------- buffer ovfw flag, 0=disabled,1=enabled
         `----------- 8th bit, 0=space,1=mark (8 bit only)

        |7|6|5|4|3|2|1|0|  S14   option register status data format
         | | | | | | | `---- unused **
         | | | | | | `----- local echo, 0=disabled, 1=enabled
         | | | | | `------ result codes, 0=enabled,1=disabled
         | | | | `------- result codes, 0=numeric,1=word
         | | | `-------- command recognition, 0=enable,1=disable
         | | `--------- dial method, 0=touch tone, 1=pulse
         | `---------- unused **
         `----------- 0=answer, 1=originate **

        S15           flag register **

        |7|6|5|4|3|2|1|0|  S16   test status
         | | | | | | | `---- analog loop, 0=inactive, 1=active
         | | | | | | `----- unused
         | | | | | `------ digital loop, 0=inactive, 1=active
         | | | | `------- remote req digital loop,0=inact.,1=active
         | | | `-------- remote digital loop,0=inact.,1=active
         | | `--------- self test RDL, 0=inactive, 1=active
         | `---------- self test analog loop,0=inactive,1=active
         `----------- unused
```

S18 0-255 remote test timer in seconds

```
|7|6|5|4|3|2|1|0|  S21   option status
 | | | | | | | `---- jack type, 0=RJ11,RJ41,RJ45,1=RJ12,RJ13
 | | | | | | `----- unused
 | | | | | `------ CTS state,  0=follows RTS, 1=forced on
 | | | `--------- DTR modem behavior (see below)
 | | `---------- DCD, 0=forced on, 1=follows carrier
 | `----------- DSR, 0=forced on, 1=behaves normally
 `------------ long space disconnect,0=disabled,1=enabled
```

Bits
43 DTR Modem Behavior Bits
00 ignore DTR
01 assume command state when DTR drops
10 assume command state and disable auto-answer when DTR drops
11 reset when DTR drops

```
|7|6|5|4|3|2|1|0|  S22   option status
 | | | | | | `------ speaker volume, 01=low,10=med,11=high
 | | | | `--------- speaker setting (see below)
 | `-------------- X command in effect
 `--------------- make/break ratio, 0=39/61, 1=33/67
```

Bits
32 S22 Speaker Setting
00 speaker always off
01 speaker on until carrier detected
10 speaker always on
11 speaker off during dialing, then on until CD

Bits
654 S22 X Command in Effect
100 X1 command in effect
101 X2 command in effect
110 X3 command in effect
111 X4 command in effect

```
|7|6|5|4|3|2|1|0|  S23   option status
 | | | | | | | `---- remote dig.loop request,0=denied,1=ok
 | | | | | `------- baud rate (see table)
 | | | | `-------- unused
 | | `----------- parity (see table)
```

```
 `-------------- guard tone setting
```

Bits
21 S23 Baud Rate Table
00 0-300 bps
01 600 bps
10 1200 bps
11 2400 bps

Bits
54 S23 Parity Setting
00 even parity
01 parity bit always 0
10 odd parity
11 parity bit always 1

Bits
76 S23 Guard Tone Setting
00 no guard tones
01 550 hz guard tone
10 1800 hz guard tone

```
S25 0-255     DTR detect delay in seconds (5)
S26 0-255     RTS to CTS delay in milliseconds (1)

|7|6|5|4|3|2|1|0|  S27   option status
 | | | | | | `------ synch/async mode (see table)
 | | | | | `------- phone line, 0= public, 1=leased
 | | | | `-------- unused
 | | `------------ clock type (see table)
 | `------------- protocol, 0=CCITT V.22, 1=Bell 212A
 `-------------- unused
```

Bits
10 S27 Sync/Async mode
00 asynchronous mode
01 async/sync mode
10 sync auto-dial
11 sync manual dial

Bits
45 S27 Clock Type
00 internal clock used
01 external clock used
10 slaved clock used

Parallel Printer Port

Port 3BC printer data output (readable)

```
 |7|6|5|4|3|2|1|0|  ports 278, 378, 3BC
  | | | | | | | `---- data bit 0, hardware pin 2
  | | | | | | `----- data bit 1, hardware pin 3
  | | | | | `------ data bit 2, hardware pin 4
  | | | | `------- data bit 3, hardware pin 5
  | | | `-------- data bit 4, hardware pin 6
  | | `--------- data bit 5, hardware pin 7
  | `---------- data bit 6, hardware pin 8
  `----------- data bit 7, hardware pin 9
```

Port 3BD printer status register   (Parallel Printer Port)

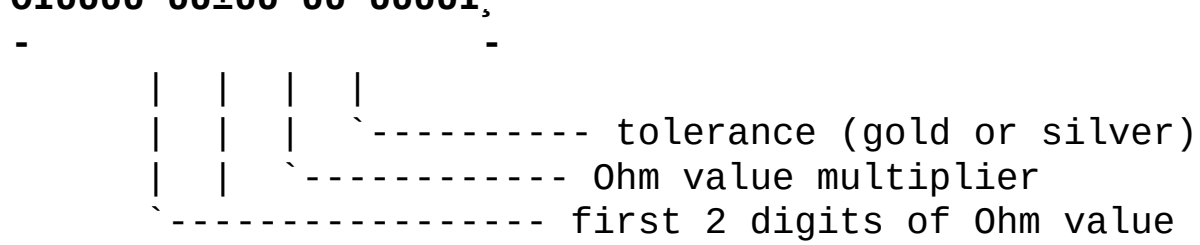
```
 |7|6|5|4|3|2|1|0|  ports 279, 379, 3BD
  | | | | | | | `---- 1 = time-out
  | | | | | | `------- unused
  | | | | `-------- 1 = error,  pin 15
  | | | `--------- 1 = on-line,  pin 13
  | | `---------- 1 = out of paper,  pin 12
  | `----------- 0 = Acknowledge,  pin 10
  `------------ 0 = busy,  pin 11
```

Port 3BE printer control register   (Parallel Printer Port)

```
 |7|6|5|4|3|2|1|0|  ports 27A, 37A, 3BE
  | | | | | | | `---- 1 = output data to printer,  (pin 1)
  | | | | | | `----- 1 = auto line feed,  (pin 14)
  | | | | | `------ 0 = initialize printer,  (pin 16)
  | | | | `------- 1 = printer reads output,  (pin 17)
  | | | `-------- 0 = IRQ disable,1=IRQ enable for ACK
  `------------- unused
```

Resistor Color Code Chart

Color	Digit	Multiplier		
Black	0	1		
Brown	1	10		
Red	2	100	Gold	ñ 5% tolerance
Orange	3	1,000	Silver	ñ10% tolerance
Yellow	4	10,000		
Green	5	100,000		
Blue	6	1,000,000		
Violet	7	10,000,000		
Gray	8	100,000,000		
White	9	1,000,000,000		



Interrupt Services DOS/BIOS/EMS/Mouse

BIOS break interrupt	BIOS clock services	BIOS communications	BIOS disk services	BIOS keyboard services
BIOS printer services	BIOS serial services	BIOS system services	BIOS timer interrupt	BIOS video services
critical error handler	Ctrl-Break	ctrl-c	DOS functions	DOS interrupts
DOS multiplex	DOS scheduler	DOS services	EMS services	EMS status
equipment flags	exceptions	exec function	INDOS	int 10
int 10,0	int 10,1	int 10,10	int 10,11	int 10,12
int 10,13	int 10,14	int 10,15	int 10,1a	int 10,1b
int 10,1c	int 10,2	int 10,3	int 10,4	int 10,5
int 10,6	int 10,7	int 10,8	int 10,9	int 10,a
int 10,b	int 10,c	int 10,d	int 10,e	int 10,f
int 10,fe	int 10,ff	int 11	int 12	int 13
int 13,0	int 13,1	int 13,10	int 13,11	int 13,12
int 13,13	int 13,14	int 13,15	int 13,16	int 13,17
int 13,18	int 13,19	int 13,1a	int 13,2	int 13,3
int 13,4	int 13,5	int 13,6	int 13,7	int 13,8
int 13,9	int 13,a	int 13,b	int 13,c	int 13,d
int 13,e	int 13,f	int 13,status	int 14	int 14,0
int 14,1	int 14,2	int 14,3	int 14,4	int 14,5
int 14,status	int 15	int 15,0	int 15,1	int 15,2
int 15,20	int 15,21	int 15,3	int 15,40	int 15,41
int 15,42	int 15,43	int 15,44	int 15,4f	int 15,80
int 15,81	int 15,82	int 15,83	int 15,84	int 15,85
int 15,86	int 15,87	int 15,88	int 15,89	int 15,90
int 15,91	int 15,c0	int 15,c1	int 15,c2	int 15,c3
int 15,c4	int 15,f	int 16	int 16,0	int 16,1
int 16,10	int 16,11	int 16,12	int 16,2	int 16,3
int 16,4	int 16,5	int 17	int 17,0	int 17,1
int 17,2	int 17,status	int 19	int 1a	int 1a,0

int 1a,1	int 1a,2	int 1a,3	int 1a,4	int 1a,5
int 1a,6	int 1a,7	int 1a,8	int 1a,80	int 1a,9
int 1a,a	int 1a,b	int 1b	int 1c	int 1d
int 1e	int 1f	int 20	int 21	int 21,0
int 21,1	int 21,10	int 21,11	int 21,12	int 21,13
int 21,14	int 21,15	int 21,16	int 21,17	int 21,19
int 21,1a	int 21,1b	int 21,1c	int 21,1f	int 21,2
int 21,21	int 21,22	int 21,23	int 21,24	int 21,25
int 21,26	int 21,27	int 21,28	int 21,29	int 21,2a
int 21,2b	int 21,2c	int 21,2d	int 21,2e	int 21,2f
int 21,3	int 21,30	int 21,31	int 21,32	int 21,33
int 21,34	int 21,35	int 21,36	int 21,37	int 21,38
int 21,39	int 21,3a	int 21,3b	int 21,3c	int 21,3d
int 21,3e	int 21,3f	int 21,4	int 21,40	int 21,41
int 21,42	int 21,43	int 21,44	int 21,44,0	int 21,44,1
int 21,44,2	int 21,44,3	int 21,44,4	int 21,44,5	int 21,44,6
int 21,44,7	int 21,44,8	int 21,44,9	int 21,44,a	int 21,44,b
int 21,44,c	int 21,44,d	int 21,44,e	int 21,44,f	int 21,45
int 21,46	int 21,47	int 21,48	int 21,49	int 21,4a
int 21,4b	int 21,4c	int 21,4d	int 21,4e	int 21,4f
int 21,5	int 21,50	int 21,51	int 21,52	int 21,53
int 21,54	int 21,55	int 21,56	int 21,57	int 21,58
int 21,59	int 21,5a	int 21,5b	int 21,5c	int 21,5d
int 21,5e	int 21,5f	int 21,6	int 21,60	int 21,62
int 21,63	int 21,64	int 21,65	int 21,66	int 21,67
int 21,68	int 21,69	int 21,6c	int 21,7	int 21,8
int 21,9	int 21,a	int 21,b	int 21,c	int 21,d
int 21,e	int 21,f	int 21,f8	int 22	int 23
int 24	int 25	int 26	int 27	int 28
int 29	int 2a	int 2a,0	int 2a,1	int 2a,3
int 2a,4	int 2a,5	int 2a,6	int 2a,80	int 2a,81
int 2a,82	int 2a,84	int 2a,87	int 2e	int 2f
int 2f,0	int 2f,1	int 2f,2	int 2f,2e	int 2f,3
int 2f,4	int 2f,5	int 2f,6	int 2f,8	int 2f,80

int 33	int 33,0	int 33,1	int 33,10	int 33,13
int 33,14	int 33,15	int 33,16	int 33,17	int 33,18
int 33,19	int 33,1a	int 33,1b	int 33,1c	int 33,1d
int 33,1e	int 33,1f	int 33,2	int 33,20	int 33,21
int 33,22	int 33,23	int 33,24	int 33,3	int 33,4
int 33,5	int 33,6	int 33,7	int 33,8	int 33,9
int 33,a	int 33,b	int 33,c	int 33,d	int 33,e
int 33,f	int 4a	int 5	int 67	int 67,40
int 67,41	int 67,42	int 67,43	int 67,44	int 67,45
int 67,46	int 67,47	int 67,48	int 67,49	int 67,4a
int 67,4b	int 67,4c	int 67,4d	int 67,4e	int 67,4f
int 67,50	int 67,51	int 67,52	int 67,53	int 67,54
int 67,55	int 67,56	int 67,57	int 67,58	int 67,59
int 67,5a	int 67,5b	int 67,5c	int 67,5d	int 67,60
int 70	int 8	int 9	int table	interrupt table
INVARS	IOCTL,0	IOCTL,1	IOCTL,2	IOCTL,3
IOCTL,4	IOCTL,5	IOCTL,6	IOCTL,7	IOCTL,8
IOCTL,9	IOCTL,a	IOCTL,b	IOCTL,c	IOCTL,d
IOCTL,e	IOCTL,f	IRQ	keyboard interrupt	mouse interrupt
mouse services	multiplex interrupt	printer status	system interrupts	user timer routine
video interrupt	video modes			

INT 1A - System and Real Time Clock BIOS Services

For more informations, see the following topics:

INT 1A,0 Read system clock counter
INT 1A,1 Set system clock counter
INT 1A,2 Read real time clock time (AT,PS/2)
INT 1A,3 Set real time clock time (AT,PS/2)
INT 1A,4 Read real time clock date (AT,PS/2)
INT 1A,5 Set real time clock date (AT,PS/2)
INT 1A,6 Set real time clock alarm (AT,PS/2)
INT 1A,7 Reset real time clock alarm (PS/2)
INT 1A,8 Set RTC activated power on mode (convertible, PS/2)
INT 1A,9 Read RTC alarm time and status (convertible,PS/2)
INT 1A,A Read system day counter (PS/2)
INT 1A,B Set system day counter (PS/2)
INT 1A,80 Set up sound multiplexer (PCjr only)

- function is specified in register AH
- see RTC

INT 1A,0 - Read System Clock Counter

```
AH = 00


on return:
AL = midnight flag, 1 if 24 hours passed since reset
CX = high order word of tick count
DX = low order word of tick count


- incremented approximately 18.206 times per second
- at midnight CX:DX is zero
- this function can be called in a program to assure the
date is
  updated after midnight;  this will avoid the passing two
midnights
  date problem
```

INT 1A,1 - Set System Clock Counter

```
AH = 01
CX = high order word of tick count
DX = low order word of tick count


returns nothing


- CX:DX should be set to the number of seconds past
  midnight multiplied by approximately 18.206
```

INT 1A,2 - Read Time From Real Time Clock (XT 286,AT,PS/2)

```
AH = 02


on return:
CF = 0 if successful
   = 1 if error, RTC not operating
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = 1 if daylight savings time option


- on AT with BIOS before 6/10/85, DL is not returned
```

INT 1A,3 - Set Time on Real Time Clock (XT 286, AT,PS/2)

```
AH = 03
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
DL = 1 if daylight savings time option
   = 0 if standard time


returns nothing


- clock values must be in BCD
- see   RTC
```

INT 1A,4 - Read Real Time Clock Date (XT 286, AT,PS/2)

```
        AH = 04


        on return:
        CH = century in BCD (decimal 19 or 20)
        CL = year in BCD
        DH = month in BCD
        DL = day in BCD
        CF = 0 if successful
           = 1 if error or clock not operating

        - calling this interrupt will update the DOS maintained
date and
          reset the BIOS Data Area date rollover flag at 40:70
        - see   RTC
```

INT 1A,5 - Set Real Time Clock Date (XT 286,AT, PS/2)

```
AH = 05
CH = century in BCD (decimal 19 or 20)
CL = year in BCD
DH = month in BCD
DL = day in BCD


returns nothing


- all values must be in BCD
- see   RTC
```

INT 1A,9 - Read RTC Alarm Time and Status (convertible,PS/2)

```
        AH = 09


        on return:
        CH = hours in BCD
        CL = minutes in BCD
        DH = seconds in BCD
        DL = alarm status:
           = 00 if alarm not enabled (AIE=0)
           = 01 if alarm enabled but will not power up system
(AIE=1)
           = 02 if alarm will power up system (AIE=1)
```

- see RTC

INT 1A,A - Read System Day Counter (PS/2)

```
AH = 0Ah


on return:
CX = count of days since 1-1-1980
```

INT 1A,B - Set System Day Counter (PS/2)

```
AH = 0Bh
CX = count of days since 1-1-1980
```

INT 13 - Diskette BIOS Services

For more information see the following topics:

INT 13,0 Reset disk system
INT 13,1 Get disk status
INT 13,2 Read disk sectors
INT 13,3 Write disk sectors
INT 13,4 Verify disk sectors
INT 13,5 Format disk track
INT 13,6 Format track and set bad sector flag (XT & portable)
INT 13,7 Format the drive starting at track (XT & portable)
INT 13,8 Get current drive parameters (XT & newer, see note Ø)
INT 13,9 Initialize 2 fixed disk base tables (XT & newer, see note Ø)
INT 13,A Read long sector (XT & newer, see note Ø)
INT 13,B Write long sector (XT & newer, see note Ø)
INT 13,C Seek to cylinder (XT & newer, see note Ø)
INT 13,D Alternate disk reset (XT & newer, see note Ø)
INT 13,E Read sector buffer (XT & portable only)
INT 13,F Write sector buffer (XT & portable only)
INT 13,10 Test for drive ready (XT & newer, see note Ø)
INT 13,11 Recalibrate drive (XT & newer, see note Ø)
INT 13,12 Controller ram diagnostic (XT & portable only)
INT 13,13 Drive diagnostic (XT & portable only)
INT 13,14 Controller internal diagnostic (XT & newer, see note Ø)
INT 13,15 Read disk type/DASD type (XT BIOS from 1/10/86 & newer)
INT 13,16 Disk change line status (XT BIOS from 1/10/86 & newer)
INT 13,17 Set dasd type for format (XT BIOS from 1/10/86 & newer)
INT 13,18 Set media type for format (BIOS date specific)
INT 13,19 Park fixed disk heads (AT & newer)
INT 13,1A Format ESDI drive unit (PS/2 50+)

Ø The 1983 version of the "IBM XT BIOS Technical Reference" shows these functions are available for the XT but many other programming references say they aren't available until the AT. This is probably due to misunderstanding the design of the disk BIOS. Upon bootup the hard disk BIOS replaces the default INT 13h diskette handler with the hard disk INT 13h handler.

Most disk BIOS calls use the following parameter scheme:

AH = function request number
AL = number of sectors (1-128 dec.)
CH = cylinder number (0-1023 dec.)
CL = sector number (1-17 dec.)
DH = head number (0-15 dec.)
DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0, 81h=drive 1)
DL = drive number (0=A:, 1=2nd floppy, 80h=C:, 81h=D:)
Note that some programming references use (0-3) as the drive number which represents diskettes only.
ES:BX = address of user buffer

and return with:
CF = 0 if successful
= 1 if error
AH = status of operation (see INT 13,STATUS)

- INT 13 diskette read functions should be retried at least 3 times to assure the disk motor has time to spin up to speed
- physical sector numbers can be converted to and from DOS sector numbers with the following formulas:

dos_sector = (sector - 1) + (head * sectors_per_track) + (track * sectors_per_track * num_heads)

```
          physical_sector = 1 + (dos_sector  MOD  sectors_per_track)
          physical_head = (dos_sector / sectors_per_track)  MOD
num_heads
          physical_track = dos_sector / (sectors_per_track *
num_heads)
```

- registers DS, BX, CX and DX are preserved
- see INT 13,STATUS

INT 13,0 - Reset Disk System

```
        AH = 00
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = disk operation status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - clears reset flag in controller and pulls heads to track 0
        - setting the controller reset flag causes the disk to
recalibrate
          on the next disk operation
        - if bit 7 is set, the diskette drive indicated by the
lower 7 bits
          will reset then the hard disk will follow; return code in
AH is
          for the drive requested
```

INT 13,5 - Format Disk Track

```
        AH = 05
        AL = interleave value (XT only)
        CX = track/cylinder number (see below for format)
        DH = head number  (0-15 dec.)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)
        ES:BX = pointer to block of "track address fields"
containing
                four byte fields for each sector to be formatted of
the form:

                1 byte  track number
                1 byte  head number           Size      #
                1 byte  sector number         Codes   Bytes
                1 byte  sector size code        0       128
                                                1       256
                                                2       512
                                                3      1024
        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - BIOS disk write attempts should reset the controller on
error
        - INT 13,17 should be called to set the DASD type
        - this function is capable of doing great damage if the
parameters
          are incorrectly specified; only the drive number is
checked
        - initializes disk address fields and data sectors
        - interleave is specified by ordering of track address
fields
        - after INT 13 disk format, if the disk is to be used with
DOS the
          DOS data structure must be written
        - only the disk number is checked for validity
        - the parameters in CX change depending on the number of
cylinders;
          the track/cylinder number is a 10 bit value taken from
```

the 2 high
order bits of CL and the 8 bits in CH (low order 8 bits of track):

```
|F|E|D|C|B|A|9|8|7|6|5-0|  CX (cylinder value 0-1023 dec.)
 | | | | | | | | | |  `-----  unused
 | | | | | | | | `---------    high order 2 bits of track/cylinder
 `------------------------  low order 8 bits of track/cyl number
```

INT 13,17 - Set DASD Type for Format (XT BIOS from 1/10/86 & newer)

```
        AH = 17h
        AL = 00 no disk
           = 01  320k/360k diskette in 320k/360k drive
           = 02  320k/360k diskette in 1.2Mb drive
           = 03  1.2Mb diskette in 1.2Mb drive
           = 04  720k diskette in 720k drive  (BIOS 6/10/85 & newer)
                 720K diskette in 1.44Mb drive (PS/2)
                 1.44Mb diskette in 1.44Mb drive (PS/2)
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        returns nothing


        - only the disk number is checked for validity
        - tells BIOS format routine about the disk type
```

INT 13,6 - Format Track and Set Bad Sector Flags (XT & portable)

```
AH = 06
AL = Interleave value (XT only)
BX = format buffer, size = 512 bytes;  the first
     2*(sectors/track) bytes contain F,N for each sector
     F = 00h for good sector,
     F = 80h for bad sector
     N = sector number


on return:
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error


- BIOS disk write attempts should reset the controller on
error
- only the disk number is checked for validity
```

INT 13,7 - Format Drive Starting at Specified Track (XT & portable)

```
        AH = 07
        AL = interleave value (XT only)
        BX = format buffer, size = 512 bytes;  the first 2*(sectors/
track)
             bytes contain F, N for each sector where:

                F = 00h for good sector
                F = 80h for bad sector
                N = sector number

        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - BIOS disk write attempts should reset the controller on
error
        - only the disk number is checked for validity
```

INT 13,8 - Get Current Drive Parameters (XT & newer)

```
        AH = 08
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = status  (see INT 13,STATUS)
        BL = CMOS drive type
             01 - 5¬  360K          03 - 3«  720K
             02 - 5¬  1.2Mb         04 - 3« 1.44Mb
        CH = cylinders (0-1023 dec. see below)
        CL = sectors per track  (see below)
        DH = number of sides (0 based)
        DL = number of drives attached
        ES:DI = pointer to 11 byte Disk Base Table (DBT)
        CF = 0 if successful
           = 1 if error
```

Cylinder and Sectors Per Track Format

```
        |F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  CX
         | | | | | | | | | | `------------  sectors per track
         | | | | | | | | `------------  high order 2 bits of
cylinder count
         `------------------------  low order 8 bits of cylinder
count

        - the track/cylinder number is a 10 bit value taken from
the 2 high
          order bits of CL and the 8 bits in CH (low order 8 bits
of track)
        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
        - only the disk number is checked for validity
```

INT 13,C - Seek to Cylinder (XT & newer)

```
AH = 0C
CH = low order byte of cylinder number (see below)
CL = high order byte of cylinder number (see below)
DH = head number (0-15)
DL = fixed drive number (80h=drive 0, 81h=drive 1)


on return:
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error


- many good programming references indicate this function
is only
  available on the AT, PS/2 and later systems, but all hard
disk
  systems since the XT have this function available
- notice that CX is in reverse byte format similar to the
way data
  is stored in memory and reverse of normal register storage
- only the disk number is checked for validity
- the parameters in CX change depending on the number of
cylinders;
  the track/cylinder number is a 10 bit value taken from
the 2 high
  order bits of CL and the 8 bits in CH (low order 8 bits
of track):

  |F|E|D|C|B|A|9|8|7|6|5-0|  CX
   | | | | | | | | | |  `-----  unused
   | | | | | | | | `---------  high order 2 bits of track
number
   `-----------------------  low order 8 bits of track
number
```

INT 13,D - Alternate Disk Reset (XT & newer)

```
        AH = 0D
        DL = fixed drive number  (80h=drive 0, 81h=drive 1)


        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
        - used to force drive recalibration similar to INT 13,0
        - drive heads are positioned at track zero
```

INT 13,E - Read Sector Buffer (XT & portable only)

```
AH = 0E


on return:
AL = number of sectors actually transferred
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error
```

INT 13,F - Write Sector Buffer (XT & portable only)

```
AH = 0F


on return:
AL = number of sectors actually transferred
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error
```

INT 13,10 - Test for Drive Ready (XT & newer)

```
        AH = 10h
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
```

INT 13,11 - Recalibrate Drive (XT & newer)

```
        AH = 11h
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
```

INT 13,12 - Controller RAM Diagnostic (XT & portable only)

```
AH = 12h


on return:
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error
```

INT 13,13 - Drive Diagnostic (XT & portable only)

```
AH = 13h


on return:
AH = status  (see INT 13,STATUS)
CF = 0 if successful
   = 1 if error
```

INT 13,14 - Controller Internal Diagnostic (XT & newer)

```
        AH = 14h


        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - many good programming references indicate this function
is only
          available on the AT, PS/2 and later systems, but all hard
disk
          systems since the XT have this function available
        - not valid on PS/2 model 30
```

INT 13,15 - Read DASD Type (XT BIOS from 1/10/86 & newer)

```
        AH = 15h
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = 00 drive not present
           = 01 diskette, no change detection present
           = 02 diskette, change detection present
           = 03 fixed disk present
        CX:DX = number of fixed disk sectors; if 3 is returned in AH
        CF = 0 if successful
           = 1 if error


        - XT's must have a BIOS date 1/10/86 or newer
        - used to determine if INT 13,16 can detect disk change

        - see   INT 13,STATUS
```

INT 13,16 - Change of Disk Status (XT BIOS from 1/10/86 & newer)

```
        AH = 16h
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = 00 no disk change
           = 01 disk changed
        CF = set if disk has been removed or an error occurred


        - used to detect if a disk change has occurred
        - see   INT 13,STATUS    INT 13,15
```

INT 13,19 - Park Fixed Disk Heads (AT & newer)

```
        AH = 19h
        DL = drive number (0=A:, 1=2nd floppy, 80h=drive 0,
81h=drive 1)


        on return:
        AH = status  (see INT 13,STATUS)
        CF = 0 if successful
           = 1 if error


        - available only on AT, XT 283 and PS/2 machines
```

INT 16 - Keyboard BIOS Services

For more information, see the following topics:

INT 16,0 Wait for keystroke and read
INT 16,1 Get keystroke status
INT 16,2 Get shift status
INT 16,3 Set keyboard typematic rate (AT+)
INT 16,4 Keyboard click adjustment (AT+)
INT 16,5 Keyboard buffer write (AT,PS/2 enhanced keyboards)
INT 16,10 Wait for keystroke and read (AT,PS/2 enhanced keyboards)
INT 16,11 Get keystroke status (AT,PS/2 enhanced keyboards)
INT 16,12 Get shift status (AT,PS/2 enhanced keyboards)

- with IBM BIOS's, INT 16 functions do not restore the flags to
  the pre-interrupt state to allow returning of information via
  the flags register
- functions 3 through 12h are not available on all AT machines
  unless the extended keyboard BIOS is present
- all registers are preserved except AX and FLAGS
- see SCAN CODES

INT 16,4 - Keyboard Click Adjustment (AT+)

```
        AH = 04
        AL = 1 for click on
           = 0 for click off


        - available only on AT and later machines that support the
extended
          keyboard BIOS
```

INT 16,11 - Extended Get Keyboard Status (AT+)

```
        AH = 11h


        on return:
        ZF = 0 if key pressed (data waiting)
        AX = 0 if no scan code is available
        AH = scan code
        AL = ASCII character or zero if special function key


        - available on AT and PS/2 machines with extended keyboard
support
        - data is not removed from buffer
        - similar to INT 16,1
```

INT 17 - Printer BIOS Services

For more information, see the following topics:

INT 17,0 Print character
INT 17,1 Initialize printer port
INT 17,2 Read printer port status

Status flags returned in register AH

```
|7|6|5|4|3|2|1|0|  AH (status)
 | | | | | | | `----  time-out
 | | | | | | `-------  unused
 | | | | | `--------  1 = I/O error  (parallel pin 15)
 | | | | `---------  1 = printer selected/on-line  (parallel
pin 13)
 | | | `----------  1 = out of paper  (parallel pin 12)
 | | `-----------  1 = printer acknowledgment  (parallel pin
10)
 | `------------  1 = printer not busy  (parallel pin 11)
 `-------------  1 = timeout
```

- on the AT, XT 286, and PS/2 when the BIOS determines the printer is busy, INT 15,90 is executed
- all printers do not return reliable status information; bit 3 and bit 7 are usually reliable.
- all registers are preserved except AH

INT 17,0 - Print Character

```
AH = 00
AL = character to print
DX = printer to be used (0-2)


on return:
AH = printer status, see INT 17,STATUS


- writes character and returns status
```

INT 17,1 - Initialize Printer Port

```
AH = 01
DX = printer port to initialize (0-2)


on return:
AH = status, see INT 17,STATUS
```

- initializes printer port and returns status
- outputs characters 0x08 and 0x0C to printer port which reset Epson and IBM printers, but may produce undesirable effects on other printers

INT 15,90 - Device Busy

```
        AH = 90h
        AL = type code:
           = 00  disk
           = 01  diskette
           = 02  keyboard
           = 03  pointing device
           = 80  network (ES:BX = NCB)
           = FC  fixed disk reset (PS/2)
           = FD  diskette motor start
           = FE  printer
        ES:BX = pointer to network control block if waiting for
network

        on return:
        CF = 0 if wait not satisfied
           = 1 if wait time satisfied
        AH = when CF set, 80h for PC & PCjr, 86h for XT (BIOS
11/8/82)

        - tells the OS a a program is about to wait for a device
        - used for multitasking OS development
```

INT 15 - System BIOS Services

For more information see the following topics:

INT 15,0	Turn cassette motor on (PC,PCjr only)
INT 15,1	Turn cassette motor off (PC,PCjr only)
INT 15,2	Read blocks from cassette (PC,PCjr only)
INT 15,3	Write blocks to cassette (PC,PCjr only)
INT 15,F	PS/2 format periodic interrupt (PS/2 only)
INT 15,20	PRINT.COM Critical region flag (undocumented DOS 3.x+)
INT 15,21	PS/2 power on self test (PS/2 except 30)
INT 15,40	Read / modify profiles (convertible only)
INT 15,41	Wait on external event (convertible only)
INT 15,42	Request system power off (convertible only)
INT 15,43	Read system status (convertible only)
INT 15,44	(De)activate internal modem power (convertible)
INT 15,4F	Keyboard intercept (BIOS date specific)
INT 15,80	Device open
INT 15,81	Device close
INT 15,82	Program termination
INT 15,83	Event wait
INT 15,84	Joy-Stick support
INT 15,85	System request key pressed
INT 15,86	Elapsed time wait (AT and after)
INT 15,87	Move block to/from extended memory
INT 15,88	Extended memory size determination
INT 15,89	Switch processor to protected mode
INT 15,90	Device busy
INT 15,91	Interrupt complete
INT 15,C0	Return system configuration parameters (PS/2 only)
INT 15,C1	Return extended BIOS data address (PS/2 only)
INT 15,C2	Pointing device BIOS interface (PS/2 only)
INT 15,C3	Enable/Disable watchdog timer
INT 15,C4	Programmable option select

- CF is set and AX is left unchanged if a function requested in AH
  isn't supported
- some Tandy BIOS's fail to restore interrupts on return from INT 15

INT 15,0 - Turn Cassette Motor On (PC,PCjr only)

```
AH = 00
```

- all programs using the cassette must actually turn on the device before use
- there is a noticeable delay between turn-on and device ready
- a noticeable clicking noise can be heard by switching the cassette on and off

- see INT 15,1

INT 15,1 - Turn Cassette Motor Off (PC,PCjr only)

```
AH = 01
```

- all programs using the cassette should turn off
  the device after use
- a noticeable clicking noise can be heard by switching the
  cassette on and off

- see INT 15,0

INT 15,21 - Power On Self Test (POST) Error Log (PS/2 except 30)

```
        AH = 21h
        AL = 00 read POST error log
           = 01 write error code to POST error log
        BH = Device code  (if write)
        BL = Device error  (if write)

        on return
        CF = 0 if successful
           = 1 if in error (AH contains code)

        if reading POST error log:
           AH = 00h if successful
              = 80h (PCjr and PC)        = 86h for all other
machines
           BX = number of POST error codes stored
           ES:DI pointer to POST error log

        if writing POST error log:
           AH = 00h if successful        = 01 POST error log full
              = 80h (PCjr and PC)        = 86h for all other
machines
```

INT 15,3 - Write Blocks to Cassette (PC,PCjr only)

```
AH = 03
CX = count of bytes to write
ES:BX = pointer to data buffer


on return:

CX = zero
ES:BX = pointer to byte following last byte written


- data is actually written in 256 byte blocks
- if CX is less than 256, then the block is padded
- no error detection/correction is available for writing
```

INT 15,F - Format Periodic Interrupt (PS/2 only)

```
        AH = 0F
        AL = phase code
           = 00  reserved
           = 01  surface analysis
           = 02  formatting

        on return:
        CF = 0 continue formatting/scanning
           = 1 end of format/scanning

        - after formatting/scanning each cylinder the formatting
          routine calls this interrupt
        - may be used by a program to gain control after formatting
          a cylinder
        - if invoked from a non-PS/2 CF=1 and AH contains the
following
          AH = 80h  for PC and PCjr
             = 86h  on most other machines
```

INT 15,40 - Read / Modify Profiles (convertible only)

```
AH = 40h

AL = 00 - return system profile in CX,BX
     01 - modify system profile
     CX,BX = profile info


AL = 02 - return internal modem profile in BX


AL = 03 - modify internal modem profile
     BX = profile info
```

INT 15,41 - Wait on External Event (convertible only)

```
AH = 41h
AL = condition type:

        |7|6|5|4|3|2|1|0|  AL
         | | | | | `-------- conditions (codes) to wait for
         | | | | `--------- reserved
         | | | `---------- 1=port address/0=user byte
         `---------------- reserved
```

Condition codes:

```
     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

BH = condition compare or mask value
BL = time-out value times 55 ms, 0 if no time limit
DX = I/O port address (if AL bit 4 = 1)
ES:DI = pointer to user byte (if AL bit 4 = 0)
```

INT 15,42 - Request System Power Off (convertible only)

```
AH = 42h

AL = 00 to use system profile
   = 01 to force suspend regardless of profile
```

INT 15,43 - Read System Status (convertible only)

```
AH = 43h


on return
AL = STATUS:

        |7|6|5|4|3|2|1|0|  AL
         | | | | | | | `---- LCD detached
         | | | | | | `----- reserved
         | | | | | `------ RS232/parallel powered on
         | | | | `------- internal modem powered on
         | | | `-------- power activated by alarm
         | | `--------- standby power lost
         | `---------- external power in use
         `----------- battery low
```

INT 15,44 - (De)activate Internal Modem Power (convertible)

```
AH = 44h

AL = 00 to power off
   = 01 to power on
```

INT 15,80 - Device Open

```
AH = 80h
BX = device id
CX = process id


on return:
CF = 0 if successful
   = 1 if error
AH = 80h for PC & PCjr
   = 86h for XT (BIOS after 11/8/82)


- available on the AT, XT 286, PS/2 and XT with BIOS after
11/8/82
- designed for use in simple multitasking
```

INT 15,81 - Device Close

```
AH = 81h
BX = device id
CX = process id


on return:
CF = 0 if successful
   = 1 if error
AH = 80h for PC & PCjr
   = 86h for XT (BIOS after 11/8/82)


- available on the AT, XT 286, PS/2 and XT with BIOS after
11/8/82
- designed for use in simple multitasking
```

INT 15,82 - Program Termination

```
AH = 82h
BX = device id


on return:
CF = 0 if successful
   = 1 if error
AH = 80h for PC & PCjr
   = 86h for XT (BIOS after 11/8/82)


- available on the AT, XT 286, PS/2 and XT with BIOS after
11/8/82
- designed for use in simple multitasking
```

INT 15,84 - Joy-Stick Support

```
AH = 84h
DX = 0 to read the current switch settings
   = 1 to read the joystick position (resistive inputs)


on return (DX=0, read switch setting):
CF = 0 if successful
   = 1 if error
AH = 80h error code if CF set on PC
   = 86h error code if CF set on XT before 11/8/82
AL = switch settings in bits 7-4 (if read switch function)

on return (DX=1, read joystick position):
AX = A(X)
BX = A(Y)
CX = B(X)
DX = B(Y)
```

INT 15,87 - Move Block to/from Extended Memory

```
AH = 87h
CX = word count of block to be moved
ES:SI = pointer to Global Descriptor Table (GDT)


on return:
CF = 0  if successful (AH contains return code)
   = 1  if error detected
AH = 0  operation successful (ZF also set)
   = 1  RAM parity error  (if CF set)
   = 2  other exception (if CF set)
   = 3  gate address on line 20h failed (if CF set)
   = 80h on PC and PCjr (if CF set)
   = 86h on XT and newer 808x machines (if CF set)

- transfers data blocks to and from extended memory on 80286
  and 80386 machines by switching from real to protected
mode
  for the duration of the transfer
- all real mode interrupts are disabled
- processor shuts down during the switch from protected mode
  to real mode on 286 processors
```

INT 15,88 - Extended Memory Size Determination

```
        AH = 88h


        on return:
        CF = 80h for PC, PCjr
           = 86h for XT and Model 30
           = other machines, set for error, clear for success
        AX = number of contiguous 1k blocks of memory starting
             at address 1024k (100000h)


        - works only on 80286 and 80386 machines
        - retrieves bytes 30 and 31 from the CMOS RAM (this data is
set
          by the boot memory scan
```

INT 15,89 - Switch Processor to Protected Mode

```
AH = 89h
BH = IRQ8 interrupt vector offset
BL = IRQ0 interrupt vector offset
CX = offset into protected mode CS to jump
ES:SI = pointer to Global Descriptor Table (GDT)


on return:
CF = 0 if successful
   = 1 if error

- switches 286 and 386 machines into protected mode to take
  advantage of advanced feature and extended memory
- Global descriptor table must be setup before interrupt
- normal BIOS functions are not available after the switch
```

INT 15,91 - Interrupt Complete

```
        AH = 91h

        on return:
        CF = 0 if successful
           = 1 if error
        AH = when CF set, 80h for PC & PCjr, 86h for XT (BIOS
11/8/82)
        AL = type code:
           = 00  disk
           = 01  diskette
           = 02  keyboard
           = 03  pointing device
           = 80  network (ES:BX = NCB)
           = FC  fixed disk reset (PS/2)
           = FD  diskette motor start
           = FE  printer

        - should not be used by applications software
        - used by the BIOS to indicate device interrupt is complete
        - used for multitasking OS development
```

INT 15,C1 - Return Extended BIOS Data Area Segment (PS/2 only)

```
        AH = C1


        on return:
        CF = 0 if successful
           = 1 if error
        AH = when CF set, 80h for PC & PCjr, 86h for XT (BIOS
11/8/82)
        ES = EBDA segment address if PS/2


        - the EBDA is used internally by the PS/2 BIOS routines
        - the EBDA is allocated at the top of user memory by the
POST
          routine
```

INT 15,C2 - Pointing Device BIOS Interface (PS/2 only)

```
AH = C2
AL = 0  enable/disable pointing device (BH contains flag)
   = 1  reset pointing device
   = 2  set sample rate
   = 3  set resolution
   = 4  read device type
   = 5  pointing device initialization
   = 6  extended commands
   = 7  pointing device far call initialization
BH = 0 indicates enable for AL = 0
   = 1 indicates disable for AL = 0


on return:
CF = 0 if successful
   = 1 if error
AH = when CF set, 80h for PC & PCjr, 86h for XT, AT
   = 01  invalid function call
   = 02  invalid input
   = 03  interface error
   = 04  resend
   = 05  no far call installed
```

INT 15,C3 - Enable/Disable Watchdog Timer (PS/2)

```
        AH = C3
        AL = 0  disable watchdog time-out
           = 1  enable watchdog time-out
        BX = watchdog timer count (1-255)

        on return:
        CF = 0 if successful
           = 1 if error
        AH = when CF set, 80h for PC & PCjr, 86h for XT, AT & Model
30


        - the watchdog timer is available only on PS/2 with 80286
and
          80386 processors.
        - the timer uses channel timer channel 3 and IRQ0.
Activated
          when IRQ0 is active for more than one channel 0 timer
cycle,
          which causes the timer to be decremented.   When the
watch-
          dog timer reaches 0, and NMI is generated.
```

INT 10 - Video BIOS Services

For more information, see the following topics:

INT 10,0 - Set video mode
INT 10,1 - Set cursor type
INT 10,2 - Set cursor position
INT 10,3 - Read cursor position
INT 10,4 - Read light pen
INT 10,5 - Select active display page
INT 10,6 - Scroll active page up
INT 10,7 - Scroll active page down
INT 10,8 - Read character and attribute at cursor
INT 10,9 - Write character and attribute at cursor
INT 10,A - Write character at current cursor
INT 10,B - Set color palette
INT 10,C - Write graphics pixel at coordinate
INT 10,D - Read graphics pixel at coordinate
INT 10,E - Write text in teletype mode
INT 10,F - Get current video state
INT 10,10 - Set/get palette registers (EGA/VGA)
INT 10,11 - Character generator routine (EGA/VGA)
INT 10,12 - Video subsystem configuration (EGA/VGA)
INT 10,13 - Write string (BIOS after 1/10/86)
INT 10,14 - Load LCD char font (convertible)
INT 10,15 - Return physical display parms (convertible)
INT 10,1A - Video Display Combination (VGA)
INT 10,1B - Video BIOS Functionality/State Information (MCGA/VGA)
INT 10,1C - Save/Restore Video State (VGA only)
INT 10,FE - Get DESQView/TopView Virtual Screen Regen Buffer
INT 10,FF - Update DESQView/TopView Virtual Screen Regen Buffer

Warning: Some BIOS implementations have a bug that causes register
BP to be destroyed. It is advisable to save BP before a call to
Video BIOS routines on these systems.

- registers CS, DS, ES, SS, BX, CX, DX are preserved unless explicitly changed
- see INT 1F INT 1D INT 29 INT 21,2 INT 21,6 INT 21,9

INT 10,1C - Save/Restore Video State (VGA only)

```
AH = 1C

AL = 0  get save buffer size
   CX = requested states
        bit 0: video hardware state
        bit 1: video BIOS data areas
        bit 2: video DAC state

on return:
AL = 1C
BX = buffer size in 64 byte blocks

AL = 1  save requested state
   CX = requested states (see AL = 0)
   ES:BX = pointer to buffer

returns nothing

AL = 2  restore requested states
   CX = requested states (see AL = 0)
   ES:BX = pointer to buffer

returns nothing
```

INT 10,2 - Set Cursor Position

```
AH = 02
BH = page number (0 for graphics modes)
DH = row
DL = column


returns nothing


- positions relative to 0,0 origin
- 80x25 uses coordinates 0,0 to 24,79;  40x25 uses 0,0 to
24,39
- the 6845 can also be used to perform this function
- setting the data in the BIOS Data Area at location 40:50
does not
  take immediate effect and is not recommended
- see   VIDEO PAGES   6845   BDA
```

INT 10,3 - Read Cursor Position and Size

```
AH = 03
BH = video page


on return:
CH = cursor starting scan line (low order 5 bits)
CL = cursor ending scan line (low order 5 bits)
DH = row
DL = column
```

- returns data from BIOS DATA AREA locations 40:50, 40:60 and 40:61
- the 6845 can also be used to read the cursor position
- the return data can be circumvented by direct port I/O to the 6845
  CRT Controller since this function returns the data found in the
  BIOS Data Area without actually checking the controller

INT 10,4 - Read Light Pen Position

```
AH = 04


on return:
AH = 0  light pen switch not triggered
   = 1  light pen triggered
BX = pixel column (0-319 or 0-639, mode dependent)
CH = raster line (0-199) (CGA and EGA modes 4, 5 and 6)
CX = raster line (EGA modes except 4, 5 and 6)
DH = row (0-24)
DL = column (0-79 or 0-79 mode dependent)


- data returned as a byte coordinate, leaving horizontal
  accuracy to within 2 pixels (320) or 4 pixels (640)
- vertical accuracy within 2 lines
- PS/2's don't support the light pen interface
```

INT 10,5 - Select Active Display Page

```
AH = 05
AL = new page number, see VIDEO PAGES

for PCjr only:
AL = 80h to read CRT/CPU page registers
     81h to set CPU page register
         BL = CPU page register
     82h to set CRT page register
         BH = CRT page register
     83h to set CPU and page registers
         BH = CRT page register
         BL = CPU page register


on return: (PCjr only)
BH = CRT page register
BL = CPU page register
```

INT 10,6 - Scroll Window Up

```
AH = 06
AL = number of lines to scroll, previous lines are
     blanked, if 0 or AL > screen size, window is blanked
BH = attribute to be used on blank line
CH = row of upper left corner of scroll window
CL = column of upper left corner of scroll window
DH = row of lower right corner of scroll window
DL = column of lower right corner of scroll window
```

```
returns nothing
```

```
- in video mode 4 (300x200 4 color) on the EGA, MCGA and VGA
  this function scrolls page 0 regardless of the current
page
- can be used to scroll graphics screens, using character
coords
- on CGA's this function disables video adapter, causing
flitter
```

INT 10,7 - Scroll Window Down

```
AH = 07
AL = number of lines to scroll, previous lines are
     blanked, if 0 or AL > screen size, window is blanked
BH = attribute to be used on blank line
CH = row of upper left corner of scroll window
CL = column of upper left corner of scroll window
DH = row of lower right corner of scroll window
DL = column of lower right corner of scroll window


returns nothing


- in video mode 4 (300x200 4 color) on the EGA, MCGA and VGA
  this function scrolls page 0 regardless of the current
page
- can be used to scroll graphics screens, using character
coords
- on CGA's this function disables video adapter, causing
flitter
```

INT 10,8 - Read Character and Attribute at Cursor Position

```
AH = 08
BH = display page


on return:
AH = attribute of character (alpha modes only)
AL = character at cursor position


- in video mode 4 (300x200 4 color) on the EGA, MCGA and VGA
  this function works only on page zero
```

INT 10,9 - Write Character and Attribute at Cursor Position

```
AH = 09
AL = ASCII character to write
BH = display page  (or mode 13h, background pixel value)
BL = character attribute (text) foreground color (graphics)
CX = count of characters to write (CX >= 1)


returns nothing


- does not move the cursor
- in graphics mode (except mode 13h), if BL bit 7=1 then
  value of BL is XOR'ed with the background color
```

INT 10,A - Write Character Only at Current Cursor Position

```
AH = 0A
AL = ASCII character to write
BH = display page  (or mode 13h, background pixel value)
BL = foreground color (graphics mode only)
CX = count of characters to write (CX >= 1)


return nothing


- similar to INT 10,9 except color ignored in text modes
```

INT 10,B - Set Color Palette

```
AH = 0B
BH = palette color ID
   = 0  to set background and border color
   = 1  to select 4 color palette
BL = color value (when BH = 0)
   = palette value (when BH = 1)

Palette    Pixel          Color
   0         0      current background color
             1      green (2)
             2      red (4)
             3      brown (6)
   1         0      current background color
             1      cyan (3)
             2      magenta (5)
             3      white (7)

- does not work for all EGA and VGA video modes
- sets border color in text mode (BH = 0)
```

INT 10,C - Write Graphics Pixel at Coordinate

```
AH = 0C
AL = color value (XOR'ED with current pixel if bit 7=1)
BH = page number, see VIDEO PAGES
CX = column number (zero based)
DX = row number (zero based)

returns nothing

- if bit 7 is 1, color specified is XOR'ed with current
pixel
- page number in BH ignored for 320x200 4 color graphics
mode
- this function is known to destroy AX and possibly SI and
DI on
  on some PS/2 VGA systems
```

INT 10,D - Read Graphics Pixel at Coordinate

```
AH = 0D
BH = page number, see VIDEO PAGES
CX = column number (zero based)
DX = row number (zero based)


on return:
AL = color of pixel read


- 64K IBM EGAs with BIOS dated 9/13/84 in 350 line video
  modes,  return invalid data in AL
- page number in BH ignored for 320x200 4 color graphics
mode
```

INT 10,10 - Set/Get Palette Registers (EGA/VGA)

```
AH = 10h

AL = 00  set individual palette register
   BH = color value
   BL = palette register


AL = 01  set border color (overscan register)
   BH = color value


AL = 02  set all palette registers and border
   ES:DX = pointer to 17 byte table representing 16 palette
           registers and border color register


AL = 03  toggle intensity/blinking (EGA)
   BL = 0  enable intensity
        1  enable blinking


AL = 07  read palette register (PS/2)
   BL = palette register to read (0-15)

on return:
  BH = value of palette register


AL = 08  read border color (overscan register, PS/2)

on return:
  BH = value of border color (overscan register)


AL = 09  read palette registers and border (PS/2)
   ES:DX = pointer to 17 byte table representing 16 palette
           registers and border color register

on return:
ES:DX = pointer to table provided as input
```

AL = 10 set DAC color register
   BX = color register to set
   CH = green value
   CL = blue value
   DH = red value

AL = 12 set block of DAC color registers
   BX = first color register to set
   CX = number of color registers to set
   ES:DX = pointer to table of color values to set

AL = 13 set attribute controller color select state
   BL = 0 set Mode Control register bit 7
      BH = value for bit 7
   BL = 1 set color select register
      BH = value for color select register

AL = 15 read DAC color register (PS/2)
   BX = color register to read

on return:
CH = green value
CL = blue value
DH = red value

AL = 17 read block of DAC color registers
   BX = first color register to read
   CX = number of color registers to read
   ES:DX = pointer to buffer for color registers

on return:
ES:DX = pointer to color table provided as input

AL = 18 update video DAC mask register
   BL = new mask

AL = 19 read video DAC mask register

on return:

BL = value read from video DAC mask register

AL = 1A read color page state
   BL = bit 7 of Mode Control Register
   BH = bits 2 thru 3 of Color select register if BL = 0
      = bits 0 thru 3 of Color select register if BL = 1

on return:
BL = current paging mode
CX = current page

AL = 1B sum color values to shades of gray
   BX = first color register to sum
   CX = number of color registers to sum

- controls the pixel color mapping bit values
- BIOS extension to EGA/VGA systems

INT 10,13 - Write String (BIOS versions from 1/10/86)

```
        AH = 13h
        AL = write mode (see bit settings below)
           = 0 string is chars only, attribute in BL, cursor not
moved
           = 1 string is chard only, attribute in BL, cursor moved
           = 2 string contains chars and attributes, cursor not
moved
           = 3 string contains chars and attributes, cursor moved
        BH = video page number
        BL = attribute if mode 0 or 1 (AL bit 1=0)
        CX = length of string (ignoring attributes)
        DH = row coordinate
        DL = column coordinate
        ES:BP = pointer to string


        Bit settings for write mode (register AL):

        |7|6|5|4|3|2|1|0|  AL
         | | | | | | | `---- 0=don't move cursor, 1=move cursor
         | | | | | | `----- 0=BL has attributes, 1=string has
attributes
         `---------------- unused


        returns nothing


        - BEL, BS, CR, LF are treated as ASCII control codes
        - wraps data and scrolls if unable to fit data on one line
```

INT 10,14 - Load LCD Character Font (convertible only)

```
AH = 14h

AL = 0 - load user specified font
   ES:DI = pointer to character font
   CX = number of characters to store
   DX = char offset into ram font area
   BH = number of bytes per character
   BL = 0  load main font (block 0)
      = 1  load alternate font (block 1)

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 attribute
   BL = 0  ignore high intensity attribute
      = 1  map high intensity to underscore
      = 2  map high intensity to reverse video
      = 3  map high intensity to select alternate font
```

INT 10,15 - Return Physical Display Parms (convertible)

```
AH = 15h

on return:
AX = alternate display adapter type
ES:DI = pointer to parameter table:
```

Offset	Size	Description
01	word	monitor model number
02	word	vertical pels per meter
03	word	horizontal pels per meter
04	word	total number of vertical pels
05	word	total number of horizontal pels
06	word	horizontal pel separation in micrometers
07	word	vertical pel separation in micrometers

INT 1D - Video Initialization Parameter Table Vector

- not a true interrupt, but a far pointer to a video initialization parameter table for video controllers

INT 21 - DOS Function Dispatcher

For more information, see the following topics:

INT 21,0 Program terminate
INT 21,1 Keyboard input with echo
INT 21,2 Display output
INT 21,3 Wait for auxiliary device input
INT 21,4 Auxiliary output
INT 21,5 Printer output
INT 21,6 Direct console I/O
INT 21,7 Wait for direct console input without echo
INT 21,8 Wait for console input without echo
INT 21,9 Print string
INT 21,A Buffered keyboard input
INT 21,B Check standard input status
INT 21,C Clear keyboard buffer, invoke keyboard function
INT 21,D Disk reset
INT 21,E Select disk
INT 21,F Open file using FCB
INT 21,10 Close file using FCB
INT 21,11 Search for first entry using FCB
INT 21,12 Search for next entry using FCB
INT 21,13 Delete file using FCB
INT 21,14 Sequential read using FCB
INT 21,15 Sequential write using FCB
INT 21,16 Create a file using FCB
INT 21,17 Rename file using FCB
INT 21,18 DOS dummy function (CP/M) (not used/listed)
INT 21,19 Get current default drive
INT 21,1A Set disk transfer address
INT 21,1B Get allocation table information
INT 21,1C Get allocation table info for specific device
INT 21,1D DOS dummy function (CP/M) (not used/listed)
INT 21,1E DOS dummy function (CP/M) (not used/listed)
INT 21,1F Get pointer to default drive parameter table (undocumented)

INT 21,20 DOS dummy function (CP/M) (not used/listed)
INT 21,21 Random read using FCB
INT 21,22 Random write using FCB
INT 21,23 Get file size using FCB
INT 21,24 Set relative record field for FCB
INT 21,25 Set interrupt vector
INT 21,26 Create new program segment
INT 21,27 Random block read using FCB
INT 21,28 Random block write using FCB
INT 21,29 Parse filename for FCB
INT 21,2A Get date
INT 21,2B Set date
INT 21,2C Get time
INT 21,2D Set time
INT 21,2E Set/reset verify switch
INT 21,2F Get disk transfer address
INT 21,30 Get DOS version number
INT 21,31 Terminate process and remain resident
INT 21,32 Get pointer to drive parameter table (undocumented)
INT 21,33 Get/set Ctrl-Break check state & get boot drive
INT 21,34 Get address to DOS critical flag (undocumented)
INT 21,35 Get vector
INT 21,36 Get disk free space
INT 21,37 Get/set switch character (undocumented)
INT 21,38 Get/set country dependent information
INT 21,39 Create subdirectory (mkdir)
INT 21,3A Remove subdirectory (rmdir)
INT 21,3B Change current subdirectory (chdir)
INT 21,3C Create file using handle
INT 21,3D Open file using handle
INT 21,3E Close file using handle
INT 21,3F Read file or device using handle
INT 21,40 Write file or device using handle
INT 21,41 Delete file
INT 21,42 Move file pointer using handle
INT 21,43 Change file mode
INT 21,44 I/O control for devices (IOCTL)
INT 21,45 Duplicate file handle
INT 21,46 Force duplicate file handle
INT 21,47 Get current directory

INT 21,48 Allocate memory blocks
INT 21,49 Free allocated memory blocks
INT 21,4A Modify allocated memory blocks
INT 21,4B EXEC load and execute program (func 1 undocumented)
INT 21,4C Terminate process with return code
INT 21,4D Get return code of a sub-process
INT 21,4E Find first matching file
INT 21,4F Find next matching file
INT 21,50 Set current process id (undocumented)
INT 21,51 Get current process id (undocumented)
INT 21,52 Get pointer to DOS "INVARS" (undocumented)
INT 21,53 Generate drive parameter table (undocumented)
INT 21,54 Get verify setting
INT 21,55 Create PSP (undocumented)
INT 21,56 Rename file
INT 21,57 Get/set file date and time using handle
INT 21,58 Get/set memory allocation strategy (3.x+, undocumented)
INT 21,59 Get extended error information (3.x+)
INT 21,5A Create temporary file (3.x+)
INT 21,5B Create new file (3.x+)
INT 21,5C Lock/unlock file access (3.x+)
INT 21,5D Critical error information (undocumented 3.x+)
INT 21,5E Network services (3.1+)
INT 21,5F Network redirection (3.1+)
INT 21,60 Get fully qualified file name (undocumented 3.x+)
INT 21,62 Get address of program segment prefix (3.x+)
INT 21,63 Get system lead byte table (MSDOS 2.25 only)
INT 21,64 Set device driver look ahead (undocumented 3.3+)
INT 21,65 Get extended country information (3.3+)
INT 21,66 Get/set global code page (3.3+)
INT 21,67 Set handle count (3.3+)
INT 21,68 Flush buffer (3.3+)
INT 21,69 Get/set disk serial number (undocumented DOS 4.0+)
INT 21,6A DOS reserved (DOS 4.0+)
INT 21,6B DOS reserved
INT 21,6C Extended open/create (4.x+)
INT 21,F8 Set OEM INT 21 handler (functions F9-FF) (undocumented)

- int 21 functions are called with the function number in AH
- register AX may be altered, its contents are not guaranteed
- if an error occurs, CF is set to 1 and AX contains a simple
  error code; INT 21,59 can be used to determine cause.
- most INT 21 functions do not restore the flags to pre-interrupt
  state to allow returning of information via the flags register

INT 21,5F AL=00 Get Redirection Mode (DOS 3.1 +)

```
AH = 5F
AL = 00
BL = redirection type
   = 03  printer
   = 04  disk drive


on return:
AX = error code if CF is set
BH = device redirection state
   = 00  redirection off
   = 01  redirection on


- determines if disk or printer redirection is enabled
- fails if file sharing is not loaded
```

INT 21,5F AL=01 Set Redirection Mode (DOS 3.1 +)

```
AH = 5F
AL = 01
BH = device redirection state
   = 00  turn redirection off
   = 01  turn redirection on
BL = redirection type
   = 03  printer
   = 04  disk drive


on return:
AX = error code if CF is set
```

```
- enables or disables disk or printer redirection
- local devices are used when redirection is off
- fails if file sharing is not loaded
```

INT 21,5F AL=02 Get Redirection List Entry (DOS 3.1+)

```
AH = 5F
AL = 02
BX = redirection list index
DS:SI = far pointer to a 16 byte buffer to hold device name
ES:DI = far pointer to a 128 byte buffer to hold network
name

on return:
AX = error code if CF is set
   = 1  invalid function code
   = 12 no more files
BH = device status flag,
   bit 0 = 0  device valid
   bit 0 = 1  device invalid
BL = device type
   = 3  printer
   = 4  drive
CX = stored parameter value
DS:SI = far pointer to ASCIIZ local device name
ES:DI = far pointer to ASCIIZ network name

- value returned in CX is the value previously passed to
  INT 21,5F with subfunction 3 in register CX
- registers DX and BP are destroyed
- fails if file sharing is not loaded
```

INT 21,5F AL=03 Redirect Device (DOS 3.1+)

```
AH = 5F
AL = 03
BL = device type
   = 3 if printer
   = 4 if drive
CX = parameter to save for caller
DS:SI = far pointer to ASCIIZ local device name
ES:DI = far pointer to ASCIIZ network name followed
        by ASCIIZ password

on return
AX = error code if CF is set
   = 1  invalid function code or
        invalid source or destination string format or
        source device already redirected
   = 3  path not found
   = 5  access denied
   = 8  insufficient memory

- parameter passed in CX can later be retrieved by calls
  to INT 21,5F subfunction 2
- local device name in DS:SI can be a drive designator
  (D:), a printer name (PRN or LPTx), or a null string
- when a null string is followed by a password is used, DOS
  tries granting access to the network directory using the
  given password
- fails if file sharing is not loaded
```

INT 21,5F AL=04 Cancel Device Redirection (DOS 3.1+)

```
AH = 5F
AL = 04
DS:SI = far pointer to ASCIIZ local device name


on return
AX = error code if CF set
   = 01  function code invalid
         ASCIIZ string is not an existing source device
```

```
           = 0F  redirection paused on server


        - local device name in DS:SI can be a drive designator
          (D:), a printer name (PRN or LPTx), or a string beginning
          with 2 backslashes "\\"
        - when two backslashes are used for the string the
connection
          between the local machine and network directory terminates
        - fails if file sharing is not loaded
```

INT 21,5F AL=05 Get Redirection List Extended Entry (DOS 4.0+)

```
        AH = 5F
        AL = 05
        BX = redirection list index
        DS:SI = pointer to buffer to receive ASCIIZ source device
name
        ES:DI = pointer to buffer to receive ASCIIZ network path

        on return
        AX = error code if CF set
        BH = device status flag (bit 0 clear if valid)
        BL = device type
           = 03  printer device
           = 04  disk device
        CX = stored user parameter value
        BP = NETBIOS local session number
        DS:SI = pointer to buffer containing ASCIIZ source device
name
        ES:DI = pointer to buffer containing ASCIIZ network path


        - retrieves the status, type, source and target of a given
redirection
        - fails if file sharing is not loaded
```

INT 21,5F AL=06 Get Redirection List (DOS 4.0+)

```
AH = 5F
AL = 06

- no information is currently available for this subfunction
```

INT 21,B - Check Standard Input Status

```
AH = 0B


on return:
AL = 00 if no character available
   = FF if character available


- checks STDIN for available characters
- character is not returned
- if Ctrl-Break is detected INT 23 is executed
```

INT 21,D - Disk Reset

```
AH = 0D


returns nothing


- all file buffers are flushed to disk
- does NOT update directory entry
```

INT 21,10 - Close a File Using FCB

```
AH = 10h
DS:DX = pointer to opened FCB


on return:
AL = 00  if file closed
   = FF  if file not closed


- closes a previously opened file opened with an FCB
- FCB must be setup with drive id, filename, and extension
  before call
```

INT 21,11 - Search for First Entry Using FCB

```
        AH = 11h
        DS:DX = pointer to unopened FCB

        on return:
        AL = 00 if matching file found
           = FF if file not found

        - searches for first occurrence of filename specified in FCB
        - FCB must have drive id, filename, and extension before
call
        - extended FCB can be used to specify a search criteria
based
          on attributes; hidden, system, label, and directory
attributes
          can be used to narrow the search  (see FILE ATTRIBUTES)
        - after successful call DTA holds an unopened FCB/XFCB for
          the requested file.   Using any of the other FCB functions
          destroys this DTA copy of the FCB/XFCB
        - searching can be continued with the FCB find-next function
        - "?" wildcard supported after DOS 2.1, "*" supported in
DOS 3.x
        - DOS 2.x can't find . and .. entries, DOS 3.x can (unless
in root)
        - see   INT 21,12
```

INT 21,12 - Search for Next Entry Using FCB

```
AH = 12h
DS:DX = pointer to unopened FCB returned from
        INT 21,11  or  INT 21,12


on return:
AL = 00 if file found
   = FF if file not found
```

- finds next matching file after calls to INT 21,11 and INT 21,12
- FCB should be the same across calls to INT 21,11 and 12
- after successful call DTA holds an unopened FCB/XFCB for the requested file. Using any of the other FCB functions destroys this DTA copy of the FCB/XFCB

INT 21,13 - Delete File Using FCB

```
        AH = 13h
        DS:DX = pointer to an unopened FCB


        on return:
        AL = 00 if file deleted
           = FF if file not found


        - deletes unopened file with normal attributes
        - FCB must contain drive id, filename, and extension before
call
        - "?" wildcard supported after DOS 2.1, "*" supported by
DOS 3.x+
        - DOS 2.x allowed deletion of a subdirectory if XFCB was
provided,
          even if files existed, causing lost clusters.  DOS 3.x
does not
```

INT 21,14 - Sequential Read Using FCB

```
AH = 14h
DS:DX = pointer to an opened FCB


on return:
AL = 00 if successful read
   = 01 if end of file (no data read)
   = 02 if DTA is too small
   = 03 if end of file or partial record read


- reads a record from file pointed to by FCB at the location
  specified in current block and current record number
- data record is read into the DTA
- FCB record number is updated
```

INT 21,15 - Sequential Write Using FCB

```
AH = 15h
DS:DX = pointer to an opened FCB


on return:
AL = 00 if write was successful
   = 01 if diskette is full or read only
   = 02 if DTA is too small


- writes a record from the DTA to the current record position
  in file specified by the opened FCB
- record size and output location are maintained in the FCB
```

INT 21,16 - Create a File Using FCB

```
AH = 16h
DS:DX = pointer to an unopened FCB


on return:
AL = 00 if file created
   = FF if file creation failed


- creates file using FCB and leaves open for later output
- FCB must be setup with drive id, filename, and extension
  before call
- an extended FCB can be used to also set file attributes
```

INT 21,17 - Rename a File Using FCB

```
AH = 17h
DS:DX = pointer to a modified FCB of the format:

  Offset     Description
    00     drive designator
    01     original file name
    09     original file extension
    11     new file name
    19     new extension


on return:
AL = 00 if file renamed
   = FF if file not renamed


- allows renaming of files with normal attributes
- "?" wildcard supported after DOS 2.1, "*" supported by
  DOS 3.x+
```

INT 21,21 - Random Read Using FCB

```
AH = 21h
DS:DX = pointer to an opened FCB


on return:
AL = 00 if read successful
   = 01 if EOF (no data read)
   = 02 if DTA is too small
   = 03 if EOF (partial record read)


- reads random records from a file opened with an FCB
  to the DTA
- FCB must be setup with drive id, filename, extension,
  record position and record length before call
- current record position field in FCB is not updated
```

INT 21,22 - Random Write Using FCB

```
AH = 22h
DS:DX = far pointer to an opened FCB


on return:
AL = 00 if write successful
   = 01 if diskette full or read only
   = 02 if DTA is too small


- write records to random location in file opened with FCB
- FCB must be setup with drive id, filename, extension,
  record position and record length before call
- current record position field in FCB is not updated
```

INT 21,23 - Get File Size Using FCB

```
AH = 23h
DS:DX = pointer to an unopened FCB


on return:
AL = 00 if successful
   = FF if file not found


- determines the number of records in a file
- FCB must be setup with drive id, complete filename and
  extension plus record length before call
- updates random record position in FCB located at DS:DX
  with file record count
```

INT 21,24 - Set Relative Record Field in FCB

```
AH = 24h
DS:DX = pointer to an opened FCB


returns nothing


- modifies opened FCB for random operation
- sets FCB random record field to current sequential block
  and record fields
```

INT 21,28 - Random Block Write Using FCB

```
AH = 28h
CX = number of records to write
DS:DX = pointer to an opened FCB


on return:
AL = 00 if write successful
   = 01 if diskette full or read only
   = 02 if DTA is too small
CX = number of records written


- allows random access and sequential writing of a group
  of records from a file opened with an FCB into the DTA
- FCB must be setup with random record number and record
size
- updates random record number, current block and current
  record fields
```

INT 21,2C - Get Time

```
AH = 2C


on return:
CH = hour (0-23)
CL = minutes (0-59)
DH = seconds (0-59)
DL = hundredths (0-99)


- retrieves DOS maintained clock time
```

INT 21,30 - Get DOS Version Number

```
        AH = 30h


        on return:
        AL = major version number (2-5)
        AH = minor version number (in hundredths decimal)
        BH = FF  indicates MS-DOS, only if OEM vendor chooses to
identify
           = 00  indicates PC-DOS
        BL:CX = 24 bit OEM serial number if BH is FF


        - for an example DOS version 2.1 returns AL=2 and AH=10
        - DOS versions prior to DOS 2.0 return zero in AH and AL
        - DOS version 4.0 and 4.1 usually return the same value of
4.00
        - the OEM serial number is a rarity, though some older OEM
DOS
          versions implemented this feature
        - the OS/2 compatibility box returns 10.10 for OS/2 1.1,
10.20
          for OS/2 1.2, etc...
        - when testing for version, a specific test can often cause
your
          code to not work in following versions of DOS.  It is
often better
          to test for a version number greater or equal to the
minimum rather
          than a specific version number where possible
        - see DOS Versions
```

VERSION - Versions of DOS

Version	Date	Changes
PC-DOS 1.0	Oct 1981	original release, single sided drive
PC-DOS 1.1	Jun 1982	bugfix, double sided drive support
MS-DOS 1.25	Jun 1982	for early compatibles
PC-DOS 2.0	Mar 1983	PC/XT, added (hard drive &UNIX features)
PC-DOS 2.1	Oct 1983	PCjr & portable mods, fixes for 2.0
MS-DOS 2.11	Oct 1983	compatible equivalent to 2.1
PC-DOS 3.0	Aug 1984	support for 1.2 Mb drive (AT)
PC-DOS 3.1	Nov 1984	added network support, fixes for 3.0
MS-DOS 2.25	Oct 1985	compatible; foreign language support
PC-DOS 3.2	Jul 1986	720k 3«" drive support for Convertible
MS-DOS 3.2	Jul 1986	
MS-DOS 3.21	May 1987	
MS-DOS 3.3	Jul 1987	
PC-DOS 3.3	Apr 1987	PS/2, 1.44 disk support, mult. partitions
MS-DOS 3.30a	Feb 1988	
CPQ-DOS 3.31	Oct 1988	Compaq DOS for disk partitions > 32MB
PC-DOS 4.00	Aug 1988	Larger DOS partitions, EMS support (bugs)
MS-DOS 4.00	Oct 1988	
MS-DOS 4.01	Nov 1988	
MS-DOS 4.01a	Apr 1989	
PC-DOS 4.01	??? ????	Fixes for major bugs in 4.0

- release dates vary between OEM versions
- IBM was supposedly responsible for most changes in DOS 4.x
- DOS 4.01 reports version 4.0, except in some vendor versions
- DOS 5.0 is primarily a Microsoft developed version

INT 21,38 - Get/Set Country Dependent Information

```
        AH = 38h
        AL = 00 to get current country information
           = 00-FE country codes  (DOS 3.x+)
           = FF for country codes >= FF, country codes are in BX
        BX = country code if AL = FF (country code > 255)
        DX = FFFF to set country information
        DS:DX = pointer to buffer to contain country data (if get
data)

        on return:
        AX = error code if CF set
           = 02 invalid country
        BX = country code (DOS 3.x+)
        DS:DX = pointer to returned country data (see COUNTRY CODES)


        - returns a pointer to country specific data, for DOS 3.x+
this
          function can be used to also set this information
```

DOS Country Codes (DOS 2.x)

Offset	Size	Description
00	word	Date and time format 0 = month day year, hh:mm:ss (USA) 1 = day month year, hh:mm:ss (Europe) 2 = year month day, hh:mm:ss (Japan)
02	2bytes	ASCIIZ currency symbol
04	2bytes	ASCIIZ thousands separator
06	2bytes	ASCIIZ decimal separator
08	18bytes	Reserved

DOS Country Codes (DOS 3.0+)

Offset	Size	Description
00	word	Date and time format 0 = month day year, hh:mm:ss (USA) 1 = day month year, hh:mm:ss (Europe) 2 = year month day, hh:mm:ss (Japan)
02	5bytes	ASCIIZ currency symbol
07	2bytes	ASCIIZ thousands separator
09	2bytes	ASCIIZ decimal separator
0B	2bytes	ASCIIZ date separator
0D	2bytes	ASCIIZ time separator
0F	byte	Currency symbol format 0 = symbol leads, without space 1 = symbol follows, without space 2 = symbol leads, one space 3 = symbol follows, one space 4 = symbol replace decimal separator
10	byte	Number of digits after decimal
11	byte	Time format Bit 0 = 0 12 hour clock = 1 24 hour clock
12	dword	Case map call address
16	2bytes	ASCIIZ data list separator
18	10bytes	Reserved

- see also INT 21,38

INT 21,39 - Create Subdirectory (mkdir)

```
AH = 39h
DS:DX = pointer to ASCIIZ path name


on return:
CF = 0 if successful
   = 1 if error
AX = error code  (see DOS ERROR CODES)


- creates specified subdirectory
- returns error if directory already exists, element of the path
  is not found, directory full or write protected disk
```

INT 21,3A - Remove Subdirectory (rmdir)

```
AH = 3A
DS:DX = pointer to ASCIIZ path name


on return:
CF = 0 if successful
   = 1 if error
AX = error code  (see DOS ERROR CODES)


- allows deletion of a subdirectory as long as it exists, is empty
  and not the current directory
```

INT 21,3B - Change Current Directory (chdir)

```
AH = 3B
DS:DX = pointer to ASCIIZ path name


on return:
CF = 0 if successful
   = 1 if error
AX = error code if CF set  (see DOS ERROR CODES)


- changes the current directory to the directory specified
  by pointer DS:DX
```

INT 21,47 - Get Current Directory

```
AH = 47h
DL = drive number (0 = default, 1 = A:)
DS:SI = pointer to a 64 byte user buffer


on return:
DS:SI = pointer ASCIIZ directory path string
AX = error code if CF set  (see DOS ERROR CODES)


- returns the current directory relative to the root
directory
- the leading slash "\" and drive designator are omitted
```

INT 21,4D - Get Return Code of Sub-process

```
AH = 4D


on return:
AH = system exit code  (indicates normal termination)
   = 00 for normal termination
   = 01 if terminated by ctl-break
   = 02 if critical device error
   = 03 if terminated by INT 21,31
AL = child exit code


- retrieve child process and system exit codes
- this function can only be used to retrieve the exit code
once
  multiple attempts to read exit codes will cause errors
```

INT 21,5C - Lock/Unlock File Access (DOS 3.0+)

```
AH = 5C
AL = 00  lock file
   = 01  unlock file
BX = file handle
CX = most significant word of region offset
DX = least significant word of region offset
SI = most significant word of region length
DI = least significant word of region length


on return:
AX = error code if CF set  (see DOS ERROR CODES)


- used for networking and multi-tasking system to preserve
  data integrity
- duplicated handles inherit access to locked regions
- EXEC'd programs do not inherit access to locked regions
```

INT 21,5E AL=0 Get Machine Name (DOS 3.1+)

```
AH = 5E
AL = 00 get machine name
DS:DX = far pointer to buffer receiving name string


on return:
AX = error code if CF set, (invalid function)
CH = 0  if name not defined
   > 0  if name defined
CL = NETBIOS name number, if CH not 0
DS:DX = far pointer to buffer containing string if CH not 0


- returns pointer to an ASCIIZ string identifying the
  computer on a Microsoft network
- output string is a 15 bytes long, padded with blanks and
  null terminated
- unpredictable if called without file sharing loaded
```

INT 21,5E AL=01 Set Machine Name (DOS 3.1+)

```
AH = 5E
AL = 01 set machine name
CH = 00 undefine name
   = any other value means to define name
CL = name number
DS:DX = pointer to 15 byte, blank padded ASCIIZ name string


on return:
AX = error code if CF set, (invalid function)


- specifies the network machine name on a Microsoft network
- unpredictable if called without file sharing loaded
```

INT 21,5E AL=02 Set Printer Setup (DOS 3.1+)

```
AH = 5E
AL = 02 set printer setup
BX = redirection list index (see INT 21,5F AL=2)
CX = length of setup string
DS:SI = far pointer to printer setup string

on return:
AX = error code if CF set, (invalid function)

- see INT 21,5F subfunction 2 for information on obtaining
the
  redirection list index
- fails if file sharing is not loaded
```

INT 21,5E AL=03 Get Printer Setup (DOS 3.1+)

```
AH = 5E
AL = 03 get printer setup
BX = redirection list index (see INT 21,5F AL=2)
ES:DI = far pointer to buffer to receive setup string

on return:
AX = error code if CF set

- see INT 21,5F subfunction 2 for information on obtaining
the
  redirection list index
- fails if file sharing is not loaded
```

INT 21,5E AL=04 Set Printer Mode (DOS 3.1+)

```
        AH = 5E
        AL = 04  set printer setup
        BX = redirection list index (see INT 21,5F AL=2)
        DX = printer mode to set (bit zero is set for binary mode
and clear
             for translated/text mode)


        on return:
        AX = error code if CF set


        - see INT 21,5F subfunction 2 for information on obtaining
the
          redirection list index
        - fails if file sharing is not loaded
        - issues INT 2F,11  and  INT 2F,1F
```

INT 21,5E AL=05 Get Printer Mode (DOS 3.1+)

```
        AH = 5E
        AL = 05  set printer setup
        BX = redirection list index (see INT 21,5F AL=2)


        on return:
        AX = error code if CF set
        DX = printer mode where bit zero is set for binary mode and
             clear for translated mode



        - see INT 21,5F subfunction 2 for information on obtaining
the
          redirection list index
        - fails if file sharing is not loaded
```

- issues INT 2F,11 and INT 2F,1F

INT 21,63 - Get Lead Byte Table (MSDOS 2.25 only)

See below for information on the Asian INT 21,63 specifications

```
        AH = 63h
        AL = 00  get address of system lead byte table
           = 01  set or clear interim console flag
           = 02  get value of interim console flag
        DL = 00  clear interim console flag (when AL=01)
           = 01  set interim console flag (when AL=01)

        on return
        AX = error code if CF set
        DS:SI = far pointer to lead byte table (AL=00)
        DL = value of interim console flag (AL=02)

        - available only in MSDOS 2.25
        - provides multiple byte character support
        - function 00, returns the address of system table
containing
          legal lead byte ranges
        - function 01 & 02 set and get the interim console flag
which
          determines whether interim bytes are returned on some
console
          functions
        - all registers except CS:IP and SS:SP may be destroyed
```

INT 21,63 - Get Lead Byte Table (Asian DOS 3.2 +)

```
        AH = 63h
        AL = 00  get address of Double Byte Character Set (DBCS)
lead table
           = 01  set Hongeul (Korean) input mode
```

```
   = 02  get Hongeul (Korean) input mode
DL = new keyboard input mode
   = 00  return completely formed characters only
   = 01  return partially formed characters also


on return
AL = 00 on success
   = FF if function not supported
DS:SI = pointer to DBCS table  (function 00)
DL = current keyboard input mode (function 02)
   = 00  only completely formed characters are accepted
   = 01  partially formed characters are accepted


- available only in Asian MSDOS 3.2+
- provides Asian multiple byte character support
- all registers except CS:IP and SS:SP may be destroyed
- DOS 4.0 returns an empty list
```

INT 21,65 - Get Extended Country Information (DOS 3.3+)

```
        AH = 65h
        AL = 01  get extended country information
           = 02  get pointer to character translation table
           = 04  get pointer to filename character translation table
           = 05  get pointer to filename terminator table
           = 06  get pointer to collating sequence
           = 07  get segment of DCBS vector (DOS 4.x)
           = 20  country dependant character capitalization (DOS 4+)
           = 21  country dependant string capitalization (DOS 4+)
           = 22  country dependant ASCIIZ string capitalization
(DOS 4+)
           = 23  determine if character represents country relative
                 Yes or No response (DOS 4+)
           = A0  country dependant filename character
capitalization (DOS 4+)
           = A1  country dependant filename string capitalization
(DOS 4+)
           = A2  country dependant ASCIIZ filename capitalization
(DOS 4+)
        BX = code page (-1 = current global code page, AL < 20h)
        CX = amount of data to return
           = length of string (AL=21h, A1h)
        DX = country ID (-1 = current country, AL < 20h)
        DL = character to capitalize or test (AL=20h, 23h, A0h)
        DH = second character if double byte character (AL=23h)
        ES:DI = pointer to output buffer (AL < 20h, see tables
below)
        DS:DX = pointer to string to capitalize (AL=21h, 22h, A1h,
A2h)


        on return
        AX = error code if CF is set  (see DOS ERROR CODES)
           = 00  NO response if (AL was 23h)
           = 01  YES response if (AL was 23h)
           = 02  not a yes or no response (AL was 23h)
        CX = amount of data returned
        DL = capitalize character (AL=20)
        ES:DI pointer to returned table of the forms:
```

Offset Size Table for Function 01

00	byte	country id = 01
01	word	size (38 or less)
03	word	country id
05	word	code Page
07	word	date Format
09	5bytes	currency symbol string (ASCIIZ)
0E	2bytes	thousands separator string (ASCIIZ)
10	2bytes	decimal separator string (ASCIIZ)
12	2bytes	date separator string (ASCIIZ)
14	2bytes	time Separator string (ASCIIZ)
16	byte	currency symbol location
17	byte	currency decimal places
18	byte	time format
19	dword	extended ASCII map call address
1D	2bytes	list separator string (ASCIIZ)
1F	10bytes	reserved

Offset Size Table for Function 2

00	byte	country Id = 02
01	dword	pointer to uppercase table

Offset Size Table for Function 4

00	byte	country Id = 04
01	dword	pointer to filename character translation table

Offset Size Table for Function 5

00	byte	country Id = 05
01	dword	pointer to filename uppercase table

File Terminator Table for Function 5

00	word	table size
02	7bytes	unknown
09	byte	length of following data
0A	nbytes	list of filename termination characters, ex: < > . " / \ [ ] : | + = ; ,

Offset Size Table for Function 6

```
  00   byte   country Id = 06
  01   dword  pointer to collating table


Offset Size     Table for Function 7
  00   byte   country Id = 07
  01   dword  pointer to byte pair table
```

- functions A0, A1, A2 are implemented in DOS 4.x but don't work

INT 21,66 - Get/Set Global Code Page (DOS 3.3+)

```
AH = 66h
AL = 01  get global code page
   = 02  set global code page
BX = active code page if setting
DX = system code page if setting


on return
AX = error code if CF set  (see DOS ERROR CODES)
BX = active code page  (for AL=1 only)
DX = system code page  (for AL=1 only)


- moves country data from COUNTRY.SYS to country memory
table
```

DOS Interrupt Summary

See the following topics for more information:

INT 20 Program terminate
INT 21 DOS Function Dispatcher
INT 22 Program Terminate
INT 23 Ctl-break exit address
INT 24 Critical error handler address
INT 25 Absolute disk read
INT 26 Absolute disk write
INT 27 Terminate but stay resident
INT 28 DOS idle loop/scheduler (undocumented)
INT 29 Fast character output (undocumented)
INT 2E Execute command using base level COMMAND.COM (undoc.)
INT 2F Multiplex interrupt (DOS 3.x+)

INT 2F,1 - Multiplex Interrupt - Submit file

```
        AL = 01
        AH = 01  resident portion of PRINT
             02  resident portion of ASSIGN
             06  get pointer to ASSIGN drive translation table
(undoc DOS 3+)
             10  resident portion of SHARE
             B7  resident portion of APPEND
             C0-FF reserved for user applications
        DS:DX = pointer to a 5 byte packet containing an priority
byte
                and a DWORD pointer to an ASCIIZ filename
        DS:DI

        on return:
        ES = segment of ASSIGN work area and assignment table (AH=6
only)
```

INT 67 - Expanded Memory Specification

For more information, see the following topics:

INT 67,40 Get EMM Status
INT 67,41 Get Page Frame Base Address
INT 67,42 Get Page Counts
INT 67,43 Get Handle and Allocate Pages
INT 67,44 Map Logical Page Into Physical Page Window
INT 67,45 Release Handle and Memory Pages
INT 67,46 Get EMM Version
INT 67,47 Save Page Map Context
INT 67,48 Restore Page Map Context
INT 67,49 Get I/O Port Addresses
INT 67,4A Get Logical to Physical Page Mapping
INT 67,4B Get Handle Count
INT 67,4C Get Page Count for Handle
INT 67,4D Get Page Count for All Handles
INT 67,4E Get/Set Page Map Context
INT 67,4F Get/Set Partial Page Map
INT 67,50 Map/Unmap Multiple Handle Pages
INT 67,51 Reallocate Pages
INT 67,52 Get/Set Handle Attributes
INT 67,53 Get/Set Handle Name
INT 67,54 Get Handle Directory
INT 67,55 Alter Page Map and Jump
INT 67,56 Alter Page Map and Call
INT 67,57 Move/Exchange Memory Region
INT 67,58 Get Mappable Physical Address Array
INT 67,59 Get Expanded Memory Hardware Information
INT 67,5A Allocate Standard/Raw Pages
INT 67,5B Alternate Map Register Set DMA Registers
INT 67,5C Prepare Expanded Memory for Warm Boot
INT 67,5D Enable/Disable OS Functions
INT 67,60 Get Physical Window Array

- function number is specified in AH
- see EMS STATUS

INT 67,41 - Get Page Frame Base Address (LIM EMS 3.0+)

```
AH = 41h


on return:
AH = 00 success
   = error code (see EMS STATUS)
BX = segment of page frame (PFBA)


- use this to determine where in the 1Mb memory address the page
  frame will be mapped
- registers not listed are preserved
```

INT 67,42 - Get Page Counts (LIM EMS 3.2+)

```
AH = 42h


on return:
AH = 00  success
   = error code (see EMS STATUS)
BX = number of unallocated or available pages
DX = total number of pages in EMM system


- registers not listed are preserved
```

INT 67,43 - Get Handle and Allocate Pages (LIM EMS 3.2+)

```
AH = 43h
BX = number of logical pages to allocate


on return:
AH = 00  success
   = error code (see EMS STATUS)
DX = EMM handle


- handles not explicitly closed by the application are not
  recoverable on exit from the program
- registers not listed are preserved
```

INT 67,44 - Map Logical Page Into Physical Page Window (LIM EMS)

```
AH = 44h
AL = physical page number (0-3)
BX = logical page number (0 to total allocated minus 1)
DX = EMM handle


on return:
AH = 00  success
   = error code (see EMS STATUS)


- registers not listed are preserved
```

INT 67,45 - Release Handle and Memory Pages (LIM EMS)

```
        AH = 45h
        DX = EMM handle


        on return:
        AH = 00 success
           = error code (see EMS STATUS)


        - handles must be explicitly freed by an application or the
memory
          will not be available to the following applications
        - if unsuccessful, the operation should be retried
        - registers not listed are preserved
```

INT 67,47 - Save Page Map Context (LIM EMS 3.0 +)

```
        AH = 47h
        DX = EMM handle assigned to the interrupt service routine
             servicing the interrupt


        on return:
        AH = 00  success
           = error code (see EMS STATUS)


        - any TSR, ISR or device driver using EMS should save
contexts
          before manipulating EMS memory and restore contexts
afterwards
        - registers not listed are preserved
        - see INT 67,48
```

INT 67,48 - Restore Page Map Context (LIM EMS 3.0+)

```
        AH = 48h
        DX = EMM handle assigned to the interrupt service routine
             servicing the interrupt


        on return:
        AH = 00  success
           = error code (see EMS STATUS)


        - any TSR, ISR or device driver using EMS should save
contexts
          before manipulating EMS memory and restore contexts
afterwards
        - registers not listed are preserved
```

INT 67,49 - Get I/O Port Addresses (LIM EMS ó 3.0)

```
        AH = 49h
        ES:DI = far pointer to storage array


        on return:
        AL = board count  (0 < AL <= 4)
        AH = 00 success
           = error code (see EMS STATUS)


        - defined in EMS 3.0, but undocumented in EMS 3.2;  new
software
          should not use this function, but old software still works
        - registers not listed are preserved
```

INT 67,4A - Get Logical to Physical Page Mapping (LIM EMS ó 3.0)

```
AH = 4A
DX = EMS handle
ES:DI far pointer to storage array


on return:
AH = 00  success
   = error code (see EMS STATUS)
BX = contains count of entries placed in array at ES:DI


- defined in EMS 3.0, but undocumented in EMS 3.2;  new software
  should not use this function, but old software still works
- registers not listed are preserved
```

INT 67,4B - Get Handle Count (LIM EMS)

```
        AH = 4Bh


        on return:
        AH = 00  success
           = error code (see EMS STATUS)
        BX = number of currently active EMM handles (0-256)
             zero indicates EMS not in use


        - to determine the number of handles available subtract BX
from 255
        - registers not listed are preserved
```

INT 67,4C - Get Page Count for Handle (LIM EMS)

```
AH = 4Ch
DX = EMM handle


on return:
AH = 00  success
   = error code (see EMS STATUS)
BX = total allocated page count for handle (1-512)


- a handle can refer to up to 512 logical pages
- registers not listed are preserved
```

INT 67,4D - Get Page Count for All Handles (LIM EMS)

```
AH = 4Dh
ES:DI = pointer to handle array


on return:
AH = 00  success
   = error code (see EMS STATUS)
BX = number of active EMM handles (0..255)
ES:DI = pointer to an array of entries of the format:

   Offset  Size     Description
     00    word   EMS handle number
     02    word   number of pages

- buffer at ES:DI should be able to hold an array of at
least 4*BX
- an error will occur if ES:DI points to a buffer that will
  cause a segment swap
- registers not listed are preserved
```

INT 67,4E - Get/Set Page Map Context (LIM EMS 3.2+)

```
AH = 4Eh
AL = 00  get page mapping context into array
         ES:DI = pointer to destination array
     01  set page mapping context from array
         DS:SI = pointer to context source array
     02  get and set page mapping registers at once
         ES:DI = pointer to destination array
         DS:SI = pointer to context source array
     03  get size of page mapping array
     04-31  reserved


on return:
AH = 00  success
   = error code (see EMS STATUS)
AL = bytes in pagemapping array (when AL=3)
ES:DI = pointer to array with mapping info (AL=00/02)

- an error will occur if ES:DI points to a buffer that will
  cause a segment swap
- designed for use by multitasking operating systems only
```

INT 67,4F - Get/Set Partial Page Map (LIM EMS 4.0+)

```
        AH = 4Fh

        AL = 00  get partial page map
             DS:SI = pointer to structure containing list of
segments whose
                     mapping contexts are to be saved
             ES:DI = pointer to array to receive page map

        AL = 01  set partial page map
             DS:SI = pointer to structure containing saved partial
page map

        AL = 02  get size of partial page map
             BX = number of mappable segments in the partial map to
be saved


        on return:
        AH = 00  success
           = error code (see EMS STATUS)
        AL = size of partial page map (if AL=2)
```

INT 67,50 - Map/Unmap Multiple Handle Pages (LIM EMS 4.0+)

```
AH = 50h
AL = 00 Map/unmap pages
   = 01 map/unmap segments
DX = EMM handle
CX = number of entries in array
DS:SI = pointer to mapping array


on return:
AH = 00  success
   = error code (see EMS STATUS)
```

INT 67,51 - Reallocate Pages (LIM EMS 4.0+)

```
AH = 51h
DX = EMM handle
BX = number of pages to be allocated to handle


on return:
AH = 00  success
   = error code (see EMS STATUS)
BX = actual number of pages allocated to handle
```

INT 67,52 - Get/Set Handle Attributes (LIM EMS 4.0+)

```
AH = 52h
AL = 00  get handle attributes
     01  set handle attributes
     02  get attribute capability
BL = new attribute (if AL=1)
DX = EMM handle


on return:
AH = 00  success
   = error code (see EMS STATUS)
AL = attribute (input AL=0)
     00  handle is volatile
     01  handle is nonvolatile
AL = attribute capability (input AL=2)
     00  only volatile handles supported
     01  both volatile and nonvolatile supported
```

INT 67,53 Get/Set Handle Name (LIM EMS 4.0+)

```
AH = 53h
AL = 00 get handle name
        ES:DI = pointer to 8byte handle name array

     01 set handle name
        DS:SI = pointer to 8byte handle name
        DX = EMM handle


on return:
AH = status  (see EMS STATUS)
```

INT 67,54 - Get Handle Directory (LIM EMS 4.0+)

```
AH = 54h
AL = 00  get handle directory
         ES:DI = pointer to buffer for handle directory

     01  search for named handle
         DS:SI = pointer to 8byte name

     02  get total number of handles


on return:
AL = number of entries in handle directory (AL = 00h)
DX = value of named handle (if AH was 01)
BX = total number of handles (if AH was 02)
AH = status  (see EMS STATUS)
```

INT 67,55 - Alter Page Map and Jump (LIM EMS 4.0+)

```
AH = 55h
AL = 00  physical page numbers provided by caller
     01  segment addresses provided by caller
DX = EMM handle
DS:SI = pointer to structure with map and jump address


on return:
AH = status  (see EMS STATUS)
```

INT 67,56 - Alter Page Map and Call (LIM EMS 4.0 +)

```
        AH = 56h
        AL = 00  physical page numbers provided by caller
                 DX = EMM handle
                 DS:SI = pointer to structure with page map and
call address
             01  segment addresses provided by caller
                 DX = EMM handle
                 DS:SI = pointer to structure with page map and
call address
             02  get page map stack space required


        on return:
        BX = stack space required (AL = 02)
        AH = status  (see EMS STATUS)


        - if successful, the target address is called
        - use a RETF to return and restore mapping context
```

INT 67,57 - Move/Exchange Memory Region (LIM EMS 4.0+)

```
        AH = 57h
        AL = 00  move memory region
             01  exchange memory region
        DS:SI = pointer to structure describing source and
destination


        on return:
        AH = status  (see EMS STATUS)
```

INT 67,58 - Get Mappable Physical Address Array (LIM EMS 4.0+)

```
AH = 58h
AL = 00  get mappable physical address array
         ES:DI = pointer to buffer to be filled with array
     01  get number of entries in m.p.a. array


on return:
CX = number of entries in array
AH = status  (see EMS STATUS)
```

INT 67,59 - Get Expanded Memory Hardware Information (LIM EMS 4.0+)

```
AH = 59h
AL = 00  get hardware configuration array
         ES:DI = pointer to buffer to be filled with array
     01  get unallocated raw page count


on return:
BX = unallocated raw pages (AL = 01)
DX = total raw pages (AL = 01)
AH = status  (see EMS STATUS)


- subfunction 00 is for use by operating systems only; can
be
  enabled or disabled at any time by the operating system
```

INT 67,5A - Allocate Standard/Raw Pages (LIM EMS 4.0+)

```
AH = 5A
AL = 00  allocate standard pages
     01  allocate raw pages
BX = number of pages to allocate


on return:
DX = EMM handle
AH = status  (see EMS STATUS)
```

INT 67,5B - Alternate Map Register Set (LIM EMS 4.0+)

```
        AH = 5B
        AL = 00  get alternate map register set
             01  set alternate map register set
                 BL = new alternate map register set number
                 ES:DI = pointer to map register context save area
if BL=0
             02  get alternate map save array size
             03  allocate alternate map register set
             04  deallocate alternate map register set
                 BL = number of alternate map register set
             05  allocate DMA register set
             06  enable DMA on alternate map register set
                 BL = DMA register set number
                 DL = DMA channel number
             07  disable DMA on alternate map register set
                 BL = DMA register set number
             08  deallocate DMA register set
                 BL = DMA register set number

        on return:
        AH = status  (see EMS STATUS)
        BL = active alternate map register set number if nonzero
(AL=0)
           = number of alternate map register set; 0 if not
supported (AL=3)
           = DMA register set number; zero if not supported (AL =
05)
        DX = array size in bytes (AL = 02)
        ES:DI = pointer to a map register context save area if BL=0
(AL=0)


        - for use by operating systems only; can be enabled or
disabled
          at any time by the operating system
```

INT 67,5C - Prepare Expanded Memory for Warm Boot (LIM EMS 4.0+)

```
AH = 5C


on return:
AH = status  (see EMS STATUS)
```

INT 67,5D - Enable/Disable OS Functions (LIM EMS 4.0+)

```
        AH = 5D
        AL = 00  enable OS function Set
             01  disable OS function Set
             02  return access key
        BX,CX = access key returned by first invocation


        on return:
        AH = status  (see EMS STATUS)
        BX,CX = access key, returned only on first invocation of
function


        - function 2 resets memory manager, returns access key at
next
          invocation
```

INT 10,1 - Set Cursor Type

AH = 01
CH = cursor starting scan line (cursor top) (low order 5 bits)
CL = cursor ending scan line (cursor bottom) (low order 5 bits)

returns nothing

- cursor scan lines are zero based
- cursor size can also be set via the 6845 CRT controller
- cursor size can be determined using the CRTC, INT 10,3 or the BIOS Data Area bytes 40:60 (ending scan line) and 40:61 (starting scan line)
- the following is a list of the cursor scan lines associated with most common adapters; screen sizes over 40 lines may differ depending on adapters.

Video	Line Count	Starting Scan Line	Ending Scan Line	Character Point Size
CGA	25	06	07	08
MDA	25	0B	0C	0E
EGA	25	06	07	0E
EGA	43	04/06	07	08
VGA	25	0D	0E	10
VGA	40	08	09	0A
VGA	50	06	07	08

- use CX = 2000h to disable cursor

INT 10,F - Get Video State

AH = 0F

on return:
AH = number of screen columns
AL = mode currently set (see VIDEO MODES)
BH = current display page

- video modes greater than 13h on EGA, MCGA and VGA indicate INT 10,0 was called with the high bit of the mode (AL) set to 1, meaning the display does not need cleared
- function returns byte value at 40:49; On EGA, MCGA and VGA bit 7 of register AL is determined by bit 7 of BIOS Data Area byte 40:87. This bit is usually set by INT 10,0 with bit 7 of the requested mode (in AL) set to 1

INT 15,2 - Read Blocks from Cassette (PC,PCjr only)

```
AH = 02
CX = count of bytes to read
ES:BX = pointer to data buffer


on return:

AH = error code if CF=1
   = 1  if CRC error
   = 2  data transition lost, (bit signal scrambled)
   = 3  no data found on tape

DX = count of bytes actually read
ES:BX = pointer to byte following last byte read


- data is transferred in 256 byte blocks, though only the
  number of bytes requested are actually stored
```

INT 16,1 - Get Keyboard Status

```
AH = 01


on return:
ZF = 0 if a key pressed (even Ctrl-Break)
AX = 0 if no scan code is available
AH = scan code
AL = ASCII character or zero if special function key


- data code is not removed from buffer
- Ctrl-Break places a zero word in the keyboard buffer but does
  register a keypress.
```

INT 1A,8 - Set RTC Activated Power On Mode (convertible)

```
AH = 08
CH = hours in BCD
CL = minutes in BCD
DH = seconds in BCD
```

- see RTC

INT 21,1 - Keyboard Input with Echo

```
AH = 01


on return:
AL = character from standard input device


- waits for keyboard input from STDIN and echoes to STDOUT
- returns 0 for extended keystroke, then function must be
  called again to return scan code
- if Ctrl-Break is detected, INT 23 is executed
```

INT 21,2 - Display Output

```
AH = 02
DL = character to output


returns nothing


- outputs character to STDOUT
- backspace is treated as non-destructive cursor left
- if Ctrl-Break is detected, INT 23 is executed
```

INT 21,3 - Wait for Auxiliary Device Input

```
AH = 03


on return:
AL = character from the auxiliary device


- does not supply error returns
- waits for character and reads from STDAUX
- default DOS AUX parameters are 2400,N,8,1
```

INT 21,4 - Auxiliary Output

```
AH = 04
DL = character to output


returns nothing


- sends character in DL to STDAUX
- does not supply error returns
- waits until STDAUX is available
- default DOS AUX parameters are 2400,N,8,1
```

INT 21,44 - I/O Control for Devices (IOCTL)

Standard Call Format

```
AH = 44h
AL = function value
BX = file handle
BL = logical device number (0=default, 1=A:, 2=B:,
3=C:, ...)
CX = number of bytes to read or write
DS:DX = data or buffer


on return:
AX = error code if CF set
AX = # of bytes transferred if CF not set
```

For more information, see the following topics:

IOCTL,0	Get Device Information
IOCTL,1	Set Device Information
IOCTL,2	Read From Character Device
IOCTL,3	Write to Character Device
IOCTL,4	Read From Block Device
IOCTL,5	Write to Block Device
IOCTL,6	Get Input Status
IOCTL,7	Get Output Status
IOCTL,8	Device Removable Query
IOCTL,9	Device Local or Remote Query
IOCTL,A	Handle Local or Remote Query
IOCTL,B	Set Sharing Retry Count
IOCTL,C	Generic I/O for Handles
IOCTL,D	Generic I/O for Block Devices (3.2+)
IOCTL,E	Get Logical Drive (3.2+)
IOCTL,F	Set Logical Drive (3.2+)

- see: DEVICE COMMAND CODES
DEVICE REQUEST HEADER
DEVICE STATUS

DEVICE HEADER
DEVICE ATTRIBUTES

INT 21,44,1 / IOCTL,1 - Set Device Information

```
AH = 44h
AL = 01
BX = handle
DH = must be zero
DL = device data low order byte  (see below)


on return
AX = error code if CF set
DX = device information  (see below)


- applicable to character devices only
- allows setting of device data word for character devices
- usually used to change from binary to translated I/O
- handle in BX must be an opened file or device
```

Device Data Word

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  Device Data Word
 | | | | | | | | | | | | | | | `---- 1 = standard input
device
 | | | | | | | | | | | | | | `---- 1 = standard output
device
 | | | | | | | | | | | | | `---- 1 = NUL device
 | | | | | | | | | | | | `---- 1 = clock device
 | | | | | | | | | | | `---- reserved
 | | | | | | | | | | `---- 1 = binary mode, 0 = translated
 | | | | | | | | | `---- 0 = end of file on input
 | | | | | | | | `---- 1 = character device
 `------------------ reserved, must be zero
```

INT 21,44,2 / IOCTL,2 - Read From Character Device

```
AH = 44h
AL = 02
BX = handle
CX = number of bytes to read
DS:DX = pointer to data buffer


on return
AX = number of bytes read if CF clear
   = error code if CF set   (see DOS ERROR CODES)


- see bit 14 if IOCTL function 00h to determine if
  driver can support IOCTL control strings
```

INT 21,44,3 / IOCTL,3 - Write to Character Device

```
AH = 44h
AL = 03
BX = handle
CX = number of bytes to send
DS:DX = pointer to data buffer


on return
AX = number of bytes written if CF clear
   = error code if CF set   (see DOS ERROR CODES)


- see bit 14 if IOCTL function 00h to determine if
  driver can support IOCTL control strings
```

INT 21,44,4 / IOCTL,4 - Read from Block Device

```
AH = 44h
AL = 04
BL = drive number (0=default, 1=A:, 2=B:, 3=C:, ...)
CX = number of bytes to read
DS:DX = pointer to data buffer


on return
AX = number of bytes read if CF clear
   = error code if CF set  (see DOS ERROR CODES)


- block drivers are not required to support this function
- returns AX = 01 (invalid function code) if not supported
```

INT 21,44,5 / IOCTL,5 - Write to Block Device

```
AH = 44h
AL = 05
BL = drive number (0=default, 1=A:, 2=B:, 3=C:, ...)
CX = number of bytes to send
DS:DX = pointer to data buffer


on return
AX = number of bytes written if CF clear
   = error code if CF set  (see DOS ERROR CODES)


- block devices are not required to support this function
- returns AX = 01 (invalid function code) if not supported
```

INT 21,44,6 / IOCTL,6 - Get Input Status

```
AH = 44h
AL = 06
BX = handle


on return
CF = 0 if successful
   = 1 if error
AX = error code if CF set  (see DOS ERROR CODES)
AL = 00  if EOF (files), or not ready (char devices)
   = FF  if not EOF (files), or ready (char devices)


- used to determine if a file or device is ready for input
- can be used to determine EOF unless EOF caused by INT
21,42
```

INT 21,44,B / IOCTL,B - Set Sharing Retry Count

```
AH = 44h
AL = 0B
CX = pause between retries  (default 1)
DX = number of retries  (default 3)


on return
AX = error code if CF set  (see DOS ERROR CODES)


- retry parameters are associated with file locking
- differences in CPU and clock speeds affect length of
pauses
- requires SHARE be loaded or invalid function is returned
- implemented from DOS 3.0
```

INT 21,44,C / IOCTL,C - Generic I/O for Handles

```
        AH = 44h
        AL = 0C
        BX = handle
        CH = device type
           = 00  unknown device type  (DOS 3.3+)
           = 01  COMx  (DOS 3.3+)
           = 03  CON  (DOS 3.3+)
           = 05  LPTx  (DOS 3.3+),  printer  (DOS 3.2)
        CL = minor function code (when CH = 3 or CH = 5)
           = 45  set iteration count  (DOS 3.2 only)

           = 4A  select code page (DOS 3.3+); parameter format:
                   00  word  length of data
                   02  word  code page ID
                   04 nwords character set data array (see offset
00)

           = 4C  code page prepare start  (DOS 3.3+); parameter
format:
                   00  word  flags
                   02  word  length of remainder of parameter block
                   04  word  number of code pages following
                   06 nwords code page 1,...,N

           = 4D  code page prepare end  (DOS 3.3+); parameter
format:
                   00  word  length of data
                   02  word  code page ID

           = 5F  set display info  (DOS 4.x, when CH=3); parameter
format:
                   00  byte  level  (0 for DOS 4.0)
                   01  byte  reserved
                   02  word  length of following data
                   04  word  control flags
                             bit 0 set for blink, clear for
intensity
                             bits 1 to 15 reserved
                   06  byte  mode type (1=text, 2=graphics)
                   07  byte  reserved
                   08  word  colors;  0=monochrome, n=bits per pixel
```

```
                  0A  word  pixel columns
                  0C  word  pixel rows
                  0E  word  character columns
                  10  word  character rows

          = 65  get iteration count  (DOS 3.2 only)

          = 6A  query selected code page  (DOS 3.3+); parameter
format:
                  00  word  length of data
                  02  word  code page ID
                  04 nwords character set data array  (see offset
00)

          = 6B  query prepare list  (DOS 3.3+); Parameter format:
                  00  word  length of following data
                  02  word  number of hardware code pages
                  04 nwords hardware code page array
                  xx  word  number of prepared code pages
                  xx nwords prepared code page array

          = 7F  get display info  (DOS 4.x, CH = 3)
                  00  byte  level  (0 for DOS 4.0)
                  01  byte  reserved
                  02  word  length of following data
                  04  word  control flags
                            bit 0 set for blink, clear for
intensity
                            bits 1 to 15 reserved
                  06  byte  mode type (1=text, 2=graphics)
                  07  byte  reserved
                  08  word  colors;  0=monochrome, n=bits per pixel
                  0A  word  pixel columns
                  0C  word  pixel rows
                  0E  word  character columns
                  10  word  character rows

        DS:DX = pointer to iteration count word  (DOS 3.2)
              = pointer to parameter block  (DOS 3.3)


        on return
        AX = error code if CF set  (see DOS ERROR CODES)


        - iteration count word specifies the number of times to
```

retry
an operation before aborting
- DOS 3.3 changed this function to handle code page switching
- implemented from DOS 3.2
- DOS 4.x adds support for double byte characters
- see IBM DOS Technical Reference Manual for more details

INT 21,44,D / IOCTL,D - Generic I/O for Block Devices

```
AH = 44h
AL = 0D
BL = drive number (0=default, 1=A:, 2=B:, 3=C:, ...)
CH = device type
   = 08 for disk drive  (block device)
CL = minor function code
   = 40  set device parameters
   = 41  write track on logical device
   = 42  format/verify track on logical drive
   = 47  set access flag  (DOS 4.x)
   = 60  get device parameters
   = 61  read track on logical device
   = 62  verify track on logical drive
   = 67  get access flag  (DOS 4.x)
DS:DX = pointer to parameter block


on return
AX = error code if CF set  (see DOS ERROR CODES)


- provides device independent primitive control operations
- implemented from DOS 3.2
- see IBM DOS Technical Reference Manual for more details
```

Device Command Codes (Device Request Header)

Code	Function	
0	INIT	
1	MEDIA CHECK	(block devices,character = NOP)
2	BUILD BPB	(block devices,character = NOP)
3	IOCTL	
4	INPUT	(read)
5	NONDESTRUCTIVE INPUT NO WAIT	(character devices)
6	INPUT STATUS	(character devices)
7	INPUT FLUSH	(character devices)
8	OUTPUT	(write)
9	OUTPUT	(write with verify)
10	OUTPUT STATUS	(character devices)
11	OUTPUT FLUSH	(character devices)
12	IOCTL OUTPUT	
13	DEVICE OPEN	(DOS 3.x)
14	DEVICE CLOSE	(DOS 3.x)
15	REMOVABLE MEDIA	(DOS 3.x)

Device Request Header Format

Offset	Size	Description
00	byte	length in bytes of the request header
01	byte	unit code; the sub-unit the operation is for (block devices); meaningless for character devices
02	word	command code, (see DEVICE COMMANDS)
04	8bytes	reserved for DOS
0C	nbytes	request data (variable length)

- see INT 21,44 or DEVICE ATTRIBUTES or DEVICE CODES
- see also IOCTL,n where "n" is an IOCTL function

Device Driver Attribute Values (brief)

```
       |F|E|D|C|B|A-7|6|5|4|3|2|1|0|
        | | | | |  |  | | | | | | `-- 1 = character device is stdin
        | | | | |  |  | | | | | |     1 = block dev supports
generic IOCTL
        | | | | |  |  | | | | | `--- 1 = character device is stdout
        | | | | |  |  | | | | |      1 = block dev supports generic
IOCTL
        | | | | |  |  | | | | `---- 1 = current NUL device
        | | | | |  |  | | | `----- 1 = current clock device
        | | | | |  |  | | `-------- reserved by DOS
        | | | | |  |  | `--------- 1 = supports Get/Set logical device
        | | | | |  |  `------------ reserved (must be zero)
        | | | | `-------------- 1 = supports removable media
        | | | `--------------- reserved (must be zero)
        | | `---------------- 1 = non-IBM format (block device)
        | |                    1 = output until busy (char device)
        | `------------------ 1 = supports IOCTL strings
        `------------------- 1 = character device, 0 = block device

       - true bit values are described, a false indicates opposite
       - see  INT 21,44  or  IOCTL,n where "n" is an IOCTL function
```

Device Driver Attribute Bit Values (detailed)

```
         0  standard input device: used by character devices to
            tell DOS a character device driver is the standard
            input device.  For block devices, a 1 indicates generic
            IOCTL supported

         1  standard output device: used by character devices to
tell DOS a
            character device driver is the standard output device.
For
            block devices, a 1 indicates generic IOCTL supported

         2  NUL attribute:  used for character devices only.  Tells
            DOS the character device driver is a NUL device.  This
```

bit
is used by DOS to determine if the NUL device is being used.
The NUL device cannot be reassigned.

3 clock device: set to 1 to tell DOS this is the new CLOCK$ device.

0B open/close removable media: set to 1 tells DOS the device
driver can handle removable media. (DOS 3.x)

0D non-IBM format: for block devices, indicates the method the driver uses to determine media type. Set to 1 for drivers that use the BPB to determine media type, set to zero for drivers that use the media descriptor byte. For
character devices (usually printers), set to 1 if the driver supports output until busy, set to 0 otherwise.

0E IOCTL bit: used with both character and block devices. Indicates if the device driver can handle control strings
through the IOCTL. Zero if a device driver can't process
control strings. If an attempt to send/receive an IOCTL control strings, is made without this bit set, an error code
is returned. The IOCTL functions allow data to be sent to
and from the driver without doing normal reads or writes.
The device driver can use the data for information. It is
up to the device to interpret the string, but the information
must not be treated as a normal I/O request. Affects only IOCTL functions AL=2 and AL=5.

0F device type: used to indicate block or character device.

Device Status Word

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

```
|15|14-10|9|8|7-0| STATUS word
 |    |   | |  `---- Error return code (if bit 15=1)
 |    |   | `------ Done bit, function completed
 |    |   `------- Busy bit
 |    `----------- Reserved
 `-------------- Error flag bits 0-7 have error code
```

Error return codes

00	Write protect violation	01	Unknown unit
02	Device not ready	03	Unknown command
04	CRC error	05	Bad drive request structure length
06	Seek error	07	Unknown media
08	Sector not found	09	Printer out of paper
0A	Write fault	0B	Read fault
0C	General failure	0D	Reserved
0E	Reserved	0F	Invalid disk change

Device Driver Header

Offset	Size	Description
00	dword	pointer to next device header
04	word	attribute (see DEVICE ATTRIBUTE)
06	word	pointer to device strategy routine
08	word	pointer to device interrupt routine
0A	8bytes	name/unit field

- see INT 21,44 and IOCTL,n where "n" is an IOCTL function

INT 21,5 - Printer Output

```
AH = 05
DL = character to output


returns nothing


- sends character in DL to STDPRN
- waits until STDPRN device is ready before output
```

INT 21,6 - Direct Console I/O

```
        AH = 06
        DL = (0-FE) character to output
           = FF if console input request

        on return:
        AL = input character if console input request (DL=FF)
        ZF = 0  if console request character available (in AL)
           = 1  if no character is ready, and function request
             was console input


        - reads from or writes to the console device depending on
          the value of DL
        - cannot output character FF  (DL=FF indicates read
function)
        - for console read, no echo is produced
        - returns 0 for extended keystroke, then function must be
          called again to return scan code
        - ignores Ctrl-Break and Ctrl-PrtSc
```

INT 21,F - Open a File Using FCB

```
AH = 0F
DS:DX = pointer to unopened FCB


on return:
AL = 00 if file opened
   = FF if unable to open


- opens an existing file using a previously setup FCB
- the FCB fields drive identifier, filename and extension
  must be filled in before call
- sets default FCB fields; current block number is set to 0;
  record size is set to 80h; file size, date and time are
set
  to the values from the directory
- does not create file, see  INT 21,16
- DOS 2.x allows opening of subdirectories, DOS 3.x does not
```

INT 2A - DOS Network Critical Section and NETBIOS

For more information see the following topics:

INT 2A,0	Network installation query
INT 2A,1	Execute NETBIOS request with no error retry
INT 2A,2	Set network printer mode (not listed)
INT 2A,3	Check if direct I/O allowed
INT 2A,4	Execute NETBIOS request
INT 2A,5	Get network resource information
INT 2A,6	Network print stream control
INT 2A,80	Begin DOS critical section
INT 2A,81	End DOS critical section
INT 2A,82	End DOS critical sections 0 through 7
INT 2A,84	Keyboard busy loop
INT 2A,87	Critical section

- functions requests are supplied in AH

INT 2A,0 - Network Installation Query

```
AH = 00


on return:
AH = zero if not installed
   = nonzero if installed


- used to detemine if a Microsoft compatible network is
present
```

INT 2A,1 - Execute NETBIOS Request With No Error Retry

```
AH = 01
ES:BX = pointer to Network Control Block


on return:
AL = NETBIOS error code
AH = zero if success
   = nonzero if error


- similar to INT 5C
```

INT 2A,3 - Check if Direct I/O Allowed

```
        AH = 03
        DS:SI = pointer to ASCIIZ disk device name or pathname
(including ":")


        on return:
        CF = set if disk access is NOT allowed
           = clear if disk access is allowed


        - do not use INT 13, INT 25 or INT 26 if this function
returns
          with the CF set
        - see  INT 5F,2
```

INT 2A,4 - Execute NETBIOS Request

```
AH = 04
AL = 00  allow retry
   = 01  no retry allowed
ES:BX = pointer to Network Control Block (NCB)


on return:
AX = 00 if successful
AH = 01 if error
AL = error code if AH = 01
```

- invokes NETBIOS handler and allows retry of operation if specified

INT 2A,5 - Get Network Resource Information

```
        AX = 0500


        on return:
        AX = destroyed
        BX = number of available network names
        CX = number of available NCB commands
        DX = number of available sessions


        - returns information on the number of available network
resources
```

INT 2A,6 - Network Print Stream Control

```
        AH = 06
        AL = 01 set concatenation mode where all printer output is
placed
                in one job
           = 02 set truncation mode where each printer open and
printer
                close starts a new job
           = 03 flush printer output then start a new print job
(similar to
                pressing Ctrl-Alt-Kp-* )


        on return:
        AX = error code if CF set


        - allows control of network printers
```

INT 2A,80 - Begin DOS Critical Section

```
        AH = 80h
        AL = critical section identifier (00-0F)
           = 01  DOS kernel, SHARE.EXE and network data structure
integrity
           = 02  DOS kernel, device driver exclusivity
           = 05  IFSFUNC (Installable File System, IFS, DOS 4.0+)
           = 06  IFSFUNC (Installable File System, IFS, DOS 4.0+)
           = 08  ASSIGN.COM

        returns nothing


        - indicates that the region of code being entered should
not be
          interrupted and identifies the critical region
        - this function is not normally called by user applications
but is
          often hooked to monitor if DOS is in a critical section
        - this information is necessary to reenter DOS using the
swappable
          data area returned by INT 21,5D subfunction 0B
        - see  Bibliography reference to "Undocumented DOS"
```

INT 2A,81 - End DOS Critical Section

```
        AH = 81h
        AL = critical section identifier (00-0F)
           = 01  DOS kernel, SHARE.EXE and network data structure
integrity
           = 02  DOS kernel, device driver exclusivity
           = 05  IFSFUNC (Installable File System, IFS, DOS 4.0+)
           = 06  IFSFUNC (Installable File System, IFS, DOS 4.0+)
           = 08  ASSIGN.COM


        returns nothing


        - indicates that a critical code section is being exited
        - this function is usually hooked rather then called
directly from a
          user application
        - see  Bibliography reference to "Undocumented DOS"
```

INT 2A,82 - End DOS Critical Sections 0 thru 7

```
        AH = 82h


        returns nothing


        - the DOS function dispatcher calls this function during DOS
          function 00, functions above 0C (excluding function 59h)
and
          during process termination
        - this function is usually hooked rather then called
directly from
          a user application
        - see  Bibliography reference to "Undocumented DOS"
```

INT 2A,84 - Keyboard Busy Loop

```
        AH = 84h

        returns nothing

        - allows other processes to continue while waiting for
keyboard
          input, similar to INT 28
        - see  BIBLIO reference to "Undocumented DOS"
```

INT 2F - DOS Multiplex Interrupt

For more information see the following topics:

		|------ ERROR CODES ----
INT 2F,0	Get installed state	01 Invalid function
INT 2F,1	Submit file	02 File not found
INT 2F,2	Cancel file	03 Path not found
INT 2F,3	Cancel all files	04 Too many files
INT 2F,4	Pause / return status	05 Access denied
INT 2F,5	End of status	08 Queue full
INT 2F,8	DRIVER.SYS support	09 Busy
INT 2F,2E	Error Translation Tables	0C Name too long
		0F Invalid drive

- function request is specified in AL or AH
- see INT 2F,0 for installation checks
- INT 2F uses AL to specify most documented functions but occasionally
  uses AH also. This can be quite confusing so INT 2F functions using
  AH are not currently listed if the conflict but will be added at a
  later time.

INT 2F,2E - Setup Error Code Translation Tables (undoc. DOS 4.x+)

- discovered in DOS 4.x by Jim Kyle, this function provides more
  detailed information on the pending critical error.

INT 2F,3 - Multiplex Interrupt - Cancel all files

```
AL = 03
AH = 01  resident portion of PRINT
     02  resident portion of ASSIGN
     10  resident portion of SHARE
     B7  resident portion of APPEND
     C0-FF reserved for user applications


returns nothing
```

INT 2F,4 - Multiplex Interrupt - Pause / return status

```
        AL = 04
        AH = 01  resident portion of PRINT
             02  resident portion of ASSIGN
             10  resident portion of SHARE
             B7  resident portion of APPEND
             C0-FF reserved for user applications


        on return:
        DX = error count
        DS:SI = pointer to queue of files


        - each file in the queue is represented by 64 byte filename
        - the file currently being printed is the first entry in
the queue
        - the last entry in the queue is a null string
```

INT 2F,5 - Multiplex Interrupt - End of Status

```
AL = 05
AH = 01  resident portion of PRINT
     02  resident portion of ASSIGN
     10  resident portion of SHARE
     B7  resident portion of APPEND
     C0-FF reserved for user applications


on return:
AH = error code  (see INT 2F)
```

INT 2F,2 - Multiplex Interrupt - Cancel file

```
AL = 02
AH = 01  resident portion of PRINT
     02  resident portion of ASSIGN
     10  resident portion of SHARE
     B7  resident portion of APPEND
     C0-FF reserved for user applications
DS:DX = pointer to ASCIIZ filename to cancel


returns nothing


- accepts wildcards
```

INT 2F,6 - PRINT.COM Check for Error on Output Device (Undocumented)

```
AL = 06
AH = 01


on return:
AX = error code if CF set
   = 00 on success
DS:SI = device driver header if CF set


- checks the output device for an error condition
- undocumented, available since DOS 3.3+
```

INT 2F,8 - DRIVER.SYS Support (Undocumented DOS 3.2+)

```
        AH = 08
        AL = 00  installation check
           = 01  add new block device
           = 02  execute device driver request
           = 03  get drive data table list (DOS 4.0+)
        DS:DI = pointer to drive data table  (AL=01)
        ES:BX = pointer to device driver request header (AL=02)


        on return:
        AL = 00  not installed, ok to install (AL=00)
           = 01  not installed, do NOT install (AL=00)
           = FF  installed (AL=00)
        ES:BX = updated device request header (AL=02)

        - function 01 scans DOS's drive data tables and appends the
new
          data table to to the chain.  All tables referencing the
same
          physical drive are updated
```

INT 33,1 - Show Mouse Cursor

```
AX = 01

returns nothing

- increments the cursor flag;  the cursor is displayed if flag
  is zero;  default flag value is -1
```

INT 33,2 - Hide Mouse Cursor

```
AX = 02

returns nothing

- decrements cursor flag; hides cursor if flag is not zero
```

Uncategorized/Miscellaneous Topics

?	ANSI codes	ANSI control codes	ASCII	Bibliography
bootstrap	byte ordering	character codes	character set	cold boot
CPU	data ranges	dBASE files	DBF files	detecting
detection	determine	EBCDIC	extended ASCII	help
HelpPC format	install	numeric ranges	POST	ranges
reboot	setup	undocumented	warm boot	

HelpPC Quick Reference Utility

Please support HelpPC by sending $25 (US, $30 outside US) to:

David Jurgens
1550 Alton Darby Creek Road
Columbus, OH 43228
CompuServe 71270,2422

Keys used for navigating through HelpPC

Esc	Exit current screen or HelpPC
End	Moves text or menu to the last line/item
Home	Moves text or menu to the first line/item
PgUp	Moves text or menu to page top or previous page
PgDn	Moves text or menu to page bottom or next page
F1	Switches from menu to prompting mode and vice versa
Alt-P	Write current topic to printer
Alt-W	Write topic to file HelpPC.DAT in the current directory
Alt-X	Exit HelpPC and without clearing current screen
TAB	Go to next subtopic link
BackTab	Go to previous subtopic link
Enter	Jump to highlighted subtopic link

Using HelpPC

HelpPC has two modes of operation. The first is the Prompted Mode where you specify the topic on the command line and HelpPC prompts you for topics until you exit. The second is the Menu Mode which as its name implies is menu driven. To use the Menu

Mode just omit the topic from the command line. You may switch
between Menu mode and Prompted Mode with the F1 key.

Examples of Command Line Usage

For information about:	Use the following topics:
Installation:	HelpPC install
Creating help files:	HelpPC HelpPC format
Interrupts:	HelpPC int #
example:	HelpPC int 13
Interrupt subfunction:	HelpPC int #,func
example:	HelpPC int 21,4E
C function usage:	HelpPC function
examples:	HelpPC open
	HelpPC is...
	HelpPC str...
Other topics:	HelpPC interrupt table
examples:	HelpPC BIOS Data Area (or BDA)
	HelpPC ANSI codes
	HelpPC bibliography

All topics are case insensitive

Trademarks of Vendors Mentioned in HelpPC Databases

/ Compaq is a registered trademark of Compaq Computer Corporation
/ CompuServe is a registered trademark of CompuServe Incorporated
/ DESQview is a trademark of Quarterdeck Office Systems
/ Epson is a registered trademark of Seiko Epson Corporation
/ HP is a registered trademark of Hewlett-Packard Company
/ IBM, PC/AT, PC-DOS, PC/XT and PS/2 are trademarks of International

```
          Business Machines Corporation
        / Intel is a registered trademark of Intel Corporation
        / LIM and EMS are trademarks of Lotus, Intel and Microsoft
Corporations
        / Maxtor is a trademark of Maxtor Corporation
        / Microsoft is a registered trademark of Microsoft
Corporation
        / NEC is a registered trademark of NEC Electronics
Incorported
        / Tandy is a registered trademark of Tandy Corporation
        / Turbo C is a registered trademark of Borland
International Inc.
        / Other brand and product names are trademarks or registered
          trademarks of their respective holders.

        - see   INSTALL   HELPPC FORMAT
```

INSTALL/SETUP - HELPPC Installation

Installation requires two steps. First, simply decompress the HelpPC archive file. Second, issue the following command:

SET HLP=d:dirname (no blanks, except between "SET HLP")

Where 'd:' is the drive and 'dirname' is the directory where the HelpPC.NDX and the .TXT files will reside (should be fully qualified, with or without the trailing backslash '\', see example below). HelpPC.COM can exist anywhere it can be accessed through the DOS command search path (PATH). The "SET HLP=" command should be placed in the AUTOEXEC.BAT file to allow easy access to the database each time your PC is rebooted. It is recommended that HelpPC.COM be renamed to something quick and comfortable to use, like H.COM or HELP.COM.

As an example, if the files HelpPC.NDX and the .TXT files reside in a directory named C:\HELPPC then use:

SET HLP=C:\HELPPC or SET HLP=C:\HELPPC

HelpPC Text File Format

HelpPC in its standard form can handle 1800 indexed topics and 16 independent help files. If you find you need a larger capacity, please contact me at one of the addresses below. Each file must have its own title which will show up in the main menu. Each topic following the file title (see below) will show up in the subtopic menu. Also note that HelpPC will adjust the menu format based on the screen height and the number of items in the main menu.

HelpPC text files are simple ASCII files that contain control codes in column one. Each file must contain a title in the first line. The remainder of the file consists of keyed lines and help text. Each line must end with a CR/LF pair (standard DOS format) and shouldn't be longer than 70 characters. Tabs position the text using 8 character tab positions. The following is a list of keys and special characters:

'@' in column 1 indicates a file title which will appear in the main topic menu. This must be the very first line in the file and has a maximum length of 40 characters (excluding the '@').

':' in column 1 indicates a subtopic key. Multiple keys separated by colons ':' can be entered on the same line. Single spaces are allowed in a key, multiple spaces are compressed to single.

'%' in column 1 indicates to highlight the entire line

'^' in column 1 indicates to center and highlight the line

' ' (space) normal text

'~' Tilde, used to mark text as a subtopic link. Use two tilde characters to represent an actual tilde in the data. A word or phrase enclosed between tilde's will become a subtopic link for the current topic.

TAB start text in column 9

any other character in column 1 is invalid an will not display

Use the BUILD command to index/reindex the default help text files. To add your own files to the index use the command:

BUILD [fname [file2 ...]]

Limits of the HelpPC program

Max items in main topic menu:	16
Max items in subtopic menu:	512
Max topics in index:	1800
Max size of topic text:	16384 bytes
Max lines of text per topic:	512
Max topic key length:	20
Max file title length:	40
Max subtopic links:	120

No limit on text file size

Example help file:

```
@This title appears in main menu
:key1:key2:key3
^This line will be displayed centered and highlighted

%This line will appear highlighted

These lines are normal text.  All three topic keys will point to
this same database entry.  The following phrase ~SUBTOPIC LINK~
is a link to another topic.  The tilde characters will cause it
to be highlighted and selectable.
```

ANSI.SYS and NANSI.SYS Functions - ANSI x3.64

```
        ESC[y,xH        Cursor position y,x
        ESC[nA          Cursor Up n lines
        ESC[nB          Cursor Down n lines
        ESC[nC          Cursor Forward n characters
        ESC[nD          Cursor Backward n characters
        ESC[y;xf        Cursor position y,x (less frequently used)
        ESC[y;xR        Cursor position report y,x
        ESC[6n          Device status report (cursor pos)(n is
constant 'n')
        ESC[s           Save cursor position
        ESC[u           Restore cursor position
        ESC[2J          Erase display
        ESC[K           Erase to end of line
        ESC[nL          Inserts n blank lines at cursor line.
(NANSI)
        ESC[nM          Deletes n lines including cursor line.
(NANSI)
        ESC[n@          Inserts n blank chars at cursor.
(NANSI)
        ESC[nP          Deletes n chars including cursor char.
(NANSI)
        ESC[n;ny        Output char translate
(NANSI)
                When first char is encountered in output request, it
                is replaced with the second char.  When no
parameters
                are given, all chars are reset.
        ESC["str"p      Keyboard Key Reassignment. The first char
of str gives
                the key to redefine; the rest of the string is the
                key's new value.  To specify unprintable chars, give
                the ASCII value of the char outside of quotes, as a
                normal parm.  IBM function keys are two byte
strings.

                Ex:  ESC[0;";dir a:";13;p

                redefines F1 to have the value "dir a:" followed by
CR.
```

If no parameters given, all keys are reset to their default values. Single or double quotes are valid.

ESC[n;n;...nm Set Graphics Rendition is used to set attributes as well as foreground and background colors. If multiple parameters are used, they are executed in sequence, and the effects are cumulative. 'n' is one of the following attributes or colors:

0	All attributes off	5	Blink
1	Bold	7	Reverse Video
2	Dim	8	Invisible
4	Underline		

Foreground colors		**Background colors**	
30	Black	40	Black
31	Red	41	Red
32	Green	42	Green
33	Yellow	43	Yellow
34	Blue	44	Blue
35	Magenta	45	Magenta
36	Cyan	46	Cyan
37	White	47	White

ESC[=nh Set mode (see screen modes for n)
ESC[=nl Reset Mode (see screen modes for n)

Screen modes are similar to those found in the IBM BIOS:

0	text 40x25 Black & White
1	text 40x25 Color
2	text 80x25 Black & White
3	text 80x25 Color

4	320x200 4 bits/pixel
5	320x200 1 bit/pixel
6	640x200 1 bit/pixel
7	cursor wrap
13	320x200 4 bits/pixel (EGA)
14	640x200 4 bits/pixel (EGA)
16	640x350 4 bits/pixel (EGA)

- for more information see ANSI document x3.64 or your DOS manual

ASCII - EBCDIC - Character Codes and Character Sets

Dec	Octal	Hex	Extended ASCII	EBCDIC	ASCII	Control Codes
0	000	00		NUL	NUL	
1	001	01	blk Face	SOH	SOH	^A
2	002	02		STX	STX	^B
3	003	03		ETX	ETX	^C
4	004	04		PF	EOT	^D
5	005	05		HT	ENQ	^E
6	006	06		LC	ACK	^F
7	007	07		DEL	BEL	^G
8	010	08	...		BS	^H
9	011	09	...		HT	^I
10	012	0A	...	SMM	LF	^J
11	013	0B		VT	VT	^K
12	014	0C		FF	FF	^L
13	015	0D	...	CR	CR	^M
14	016	0E		SO	SO	^N
15	017	0F		SI	SI	^O
16	020	10		DLE	DLE	^P
17	021	11		DC1	DC1	^Q
18	022	12		DC2	DC2	^R
19	023	13		TM	DC3	^S
20	024	14		RES	DC4	^T
21	025	15		NL	NAK	^U
22	026	16		BS	SYN	^V
23	027	17		IL	ETB	^W
24	030	18		CAN	CAN	^X
25	031	19		EM	EM	^Y
26	032	1A	...	CC	SUB	^Z
27	033	1B		CU1	ESC	^[
28	034	1C		IFS	FS	^\
29	035	1D		IGS	GS	^]
30	036	1E		IRS	RS	^^
31	037	1F		IUS	US	^_
32	040	20		DS	SP	
33	041	21	!	SOS	!	
34	042	22	"	FS	"	
35	043	23	#		#	
36	044	24	$	BYP	$	

37	045	25	%	LF	%
38	046	26	&	ETB	&
39	047	27	'	ESC	'
40	050	28	(		(
41	051	29	)		)
42	052	2A	*	SM	*
43	053	2B	+	CU2	+
44	054	2C	,		,
45	055	2D	-	ENQ	-
46	056	2E	.	ACK	.
47	057	2F	/	BEL	/
48	060	30	0		0
49	061	31	1		1
50	062	32	2	SYN	2
51	063	33	3		3
52	064	34	4	PN	4
53	065	35	5	RS	5
54	066	36	6	UC	6
55	067	37	7	EOT	7
56	070	38	8		8
57	071	39	9		9
58	072	3A	:		:
59	073	3B	;	CU3	;
60	074	3C	<	DC4	<
61	075	3D	=	NAK	=
62	076	3E	>		>
63	077	3F	?	SUB	?
64	100	40	@	SP	@
65	101	41	A		A
66	102	42	B		B
67	103	43	C		C
68	104	44	D		D
69	105	45	E		E
70	106	46	F		F
71	107	47	G		G
72	110	48	H		H
73	111	49	I		I
74	112	4A	J	›	J
75	113	4B	K	.	K
76	114	4C	L	<	L
77	115	4D	M	{	M
78	116	4E	N	+	N
79	117	4F	O	|	O
80	120	50	P	&	P
81	121	51	Q		Q
82	122	52	R		R

83	123	53	S		S
84	124	54	T		T
85	125	55	U		U
86	126	56	V		V
87	127	57	W		W
88	130	58	X		X
89	131	59	Y		Y
90	132	5A	Z	!	Z
91	133	5B	[	$	[
92	134	5C	\	*	\
93	135	5D	]	)	]
94	136	5E	^	;	^
95	137	5F	_	ª	_
96	140	60	`		`
97	141	61	a	/	a
98	142	62	b		b
99	143	63	c		c
100	144	64	d		d
101	145	65	e		e
102	146	66	f		f
103	147	67	g		g
104	150	68	h		h
105	151	69	i		i
106	152	6A	j		j
107	153	6B	k	,	k
108	154	6C	l	%	l
109	155	6D	m	_	m
110	156	6E	n	>	n
111	157	6F	o	?	o
112	160	70	p		p
113	161	71	q		q
114	162	72	r		r
115	163	73	s		s
116	164	74	t		t
117	165	75	u		u
118	166	76	v		v
119	167	77	w		w
120	170	78	x		x
121	171	79	y		y
122	172	7A	z	:	z
123	173	7B	{	#	{
124	174	7C	|	@	|
125	175	7D	}	'	}
126	176	7E	~	=	~
127	177	7F	•	"	•
128	200	80	€		

129	201	81	•	a
130	202	82	‚	b
131	203	83	ƒ	c
132	204	84	„	d
133	205	85	…	e
134	206	86	†	f
135	207	87	‡	g
136	210	88	ˆ	h
137	211	89	‰	i
138	212	8A	Š	
139	213	8B	‹	
140	214	8C	Œ	
141	215	8D	•	
142	216	8E	Ž	
143	217	8F	•	
144	220	90	•	
145	221	91	‘	j
146	222	92	’	k
147	223	93	“	l
148	224	94	”	m
149	225	95	•	n
150	226	96	–	o
151	227	97	—	p
152	230	98	˜	q
153	231	99	™	r
154	232	9A	š	
155	233	9B	›	
156	234	9C	œ	
157	235	9D	•	
158	236	9E	ž	
159	237	9F	Ÿ	
160	240	A0		
161	241	A1	¡	
162	242	A2	¢	s
163	243	A3	£	t
164	244	A4	¤	u
165	245	A5	¥	v
166	246	A6	¦	w
167	247	A7	§	x
168	250	A8	¨	y
169	251	A9	©	z
170	252	AA	ª	
171	253	AB	«	
172	254	AC	¬	
173	255	AD	-	
174	256	AE	®	

175	257	AF	¯	
176	260	B0	°	
177	261	B1	±	
178	262	B2	²	
179	263	B3			
180	264	B4			
181	265	B5	µ	
182	266	B6	¶	
183	267	B7	·	
184	270	B8	¸	
185	271	B9	¹	
186	272	BA	º	
187	273	BB	»	
188	274	BC	¼	
189	275	BD	½	
190	276	BE	¾	
191	277	BF	.	
192	300	C0	`	
193	301	C1	-	A
194	302	C2	-	B
195	303	C3			C
196	304	C4	-	D
197	305	C5	+	E
198	306	C6	Æ	F
199	307	C7	Ç	G
200	310	C8	È	H
201	311	C9	É	I
202	312	CA	Ê	
203	313	CB	Ë	
204	314	CC	Ì	
205	315	CD	Í	
206	316	CE	Î	
207	317	CF	Ï	
208	320	D0	Ð	
209	321	D1	Ñ	J
210	322	D2	Ò	K
211	323	D3	Ó	L
212	324	D4	Ô	M
213	325	D5	Õ	N
214	326	D6	Ö	O
215	327	D7	×	P
216	330	D8	Ø	Q
217	331	D9	'	R
218	332	DA			
219	333	DB	Û	
220	334	DC	Ü	

221	335	DD	Ý	
222	336	DE	Þ	
223	337	DF	ß	
224	340	E0	à	
225	341	E1	á	
226	342	E2	â	S
227	343	E3	ã	T
228	344	E4	ä	U
229	345	E5	å	V
230	346	E6	æ	W
231	347	E7	ç	X
232	350	E8	è	Y
233	351	E9	é	Z
234	352	EA	ê	
235	353	EB	ë	
236	354	EC	ì	
237	355	ED	í	
238	356	EE	î	
239	357	EF	ï	
240	360	F0	ð	0
241	361	F1	ñ	1
242	362	F2	ò	2
243	363	F3	ó	3
244	364	F4	ô	4
245	365	F5	õ	5
246	366	F6	ö	6
247	367	F7	÷	7
248	370	F8	ø	8
249	371	F9	/	9
250	372	FA	ú	
251	373	FB	û	
252	374	FC	ü	
253	375	FD	ý	
254	376	FE	þ	
255	377	FF		

Byte Ordering of Different Computer Architectures

Architecture	16 Bit Integer	32 Bit Format	Floating Point
MC68000	MSB	MSB	MSB
Intel	LSB	LSB	LSB
PDP-11	LSB	MSW...LSW	MSW...LSW
VAX	LSB	LSB	MSW...LSW
IBM 360/370	MSB	MSB	MSB

MSB means Most Significant Byte first or a byte order of 3210

LSB means Least Significant Byte first or a byte order of 0123

MSW...LSW means a byte order of 3201 or 67452301

See BIBLIO reference "Computer Language Magazine", April, 1987, P.J. Plauger for more information

Numeric Ranges for Data Elements

Size	Range	Significant digits
1 byte integer	-128 to 127	
1 byte unsigned	0 to 255	
2 byte integer	-32,768 to 32,767	
2 byte unsigned	0 to 65535	
4 byte integer	-2,147,483,648 to 2,147,483,647	
4 byte unsigned	0 to 4,294,967,295	
8 byte integer	-9x10e18 to 9x10e18	
4 byte real	3.4x10e-38 to 3.37x10e38	7
8 byte real	1.7x10e-308 to 1.67x10e308	15
10 byte real	3.4x10e-4932 to 1.1x10e4932	19
10 byte packed dec	-99..99 to 99..99	

dBASE - File Header Structure (dBASE II)

Offset	Size	Description
00	byte	dBASE version number 02h=dBASE II
01	word	number of data records in file
03	byte	month of last update
04	byte	day of last update
05	byte	year of last update
06	word	size of each data record
08	512bytes	field descriptors (see below)
520	byte	0Dh if all 32 field descriptors used; otherwise 00h

- dBASE II file header has a fixed size of 521 bytes

DBASE - File header structure (DBASE III)

Offset	Size	Description
00	byte	dBASE vers num 03h=dBASE III w/o .DBT 83h=dBASE III w .DBT
01	byte	year of last update
02	byte	month of last update
03	byte	day of last update
04	dword	long int number of data records in file
08	word	header structure length
10	word	data record length
12	20bytes	version 1.0 reserved data space
32-n	32bytes ea.	field descriptors (see below)
n+1	byte	0dH field terminator.

- unlike dBASE II, dBASE III has a variable length header

dBASE - Field Descriptors

dBASE II Field Descriptors (header contains 32 FDs)

Offset	Size	Description
00	11bytes	null terminated field name string, 0Dh as first byte indicates end of FDs
11	byte	data type, Char/Num/Logical (C,N,L)
12	byte	field length
13	word	field data address, (set in memory)
15	byte	number of decimal places

dBASE III Field Descriptors (FD count varies):

Offset	Size	Description
00	11bytes	null terminated field name string
11	byte	data type, Char/Num/Logical/Date/Memo
12	dword	long int field data address, (set in memory)
16	byte	field length
17	byte	number of decimal places
18	14bytes	version 1.00 reserved data area

Tables and Formats used by DOS & BIOS

.EXE	backup headers	Batch Control Block	BCB	BDA
BIOS Data Area	BIOS memory	BIOS Parameter Block	boot record	boot sector
Box Drawing Chars	BPB	break codes	code pages	color table
colors	country codes	country info	DBT	default handles
device attributes	device codes	device command codes	device commands	device header
device request header	device status	directory format	Disk Base Table	disk formats
disk parameter table	disk partition table	disk status data	Disk Transfer Area	diskette status data
DOS error codes	DOS versions	DPB	DPT	drive parameter table
drive status data	DTA	EBDA	EXE file header	extended BIOS
extended errors	Extended FCB	extended file ctlblk	FAT	FCB
File Allocation Table	file attributes	File Control Block	file handles	file table
float formats	floating point	floppy formats	GDT	Global Descriptor Table
handles	ID bytes	KB flags	keyboard flags	machine identificatn
make codes	MCB	media descriptor byte	Memory Control Block	memory map
partition table	Program Segment Prefix	PSP	real numbers	scan codes
SFT	system descriptor	System File Table	version	video information
video pages	video tables	XFCB		

.EXE - DOS EXE File Structure

Offset	Size	Description
00	word	"MZ" - Link file .EXE signature (Mark Zbikowski?)
02	word	length of image mod 512
04	word	size of file in 512 byte pages
06	word	number of relocation items following header
08	word	size of header in 16 byte paragraphs, used to locate the beginning of the load module
0A	word	min # of paragraphs needed to run program
0C	word	max # of paragraphs the program would like
0E	word	offset in load module of stack segment (in paras)
10	word	initial SP value to be loaded
12	word	negative checksum of pgm used while by EXEC loads pgm
14	word	program entry point, (initial IP value)
16	word	offset in load module of the code segment (in paras)
18	word	offset in .EXE file of first relocation item
1A	word	overlay number (0 for root program)

- relocation table and the program load module follow the header
- relocation entries are 32 bit values representing the offset into the load module needing patched
- once the relocatable item is found, the CS register is added to the value found at the calculated offset

Registers at load time of the EXE file are as follows:

AX:	contains number of characters in command tail, or 0
BX:CX	32 bit value indicating the load module memory size
DX	zero
SS:SP	set to stack segment if defined else, SS = CS and SP=FFFFh or top of memory.
DS	set to segment address of EXE header
ES	set to segment address of EXE header
CS:IP	far address of program entry point, (label on "END" statement of program)

DOS BACKUP Control Information

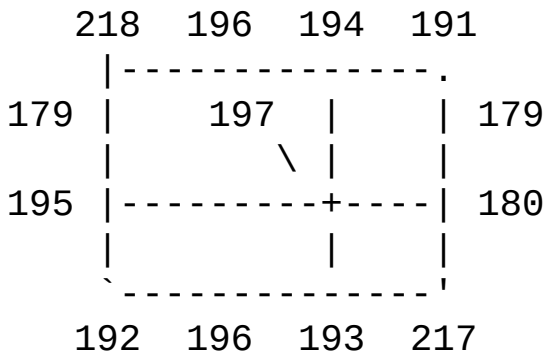
BACKUPID.@@@ Format

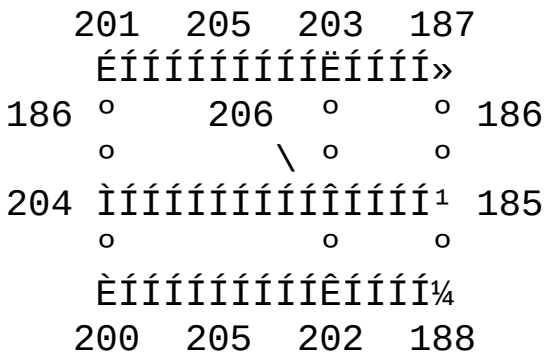
Offset	Size	Description
00	byte	Disk sequence flag: 00 indicates disk is not the last backup diskette FF indicates disk is the last backup diskette
01	word	Floppy disk sequence number (Intel format)
03	word	Backup year, four digits (Intel format)
05	byte	Day of month (1-31)
06	byte	Month of year (1-12)
07	dword	System time if /T was specified (see FILE ATTRIBUTES)
0B	117bytes	Unused

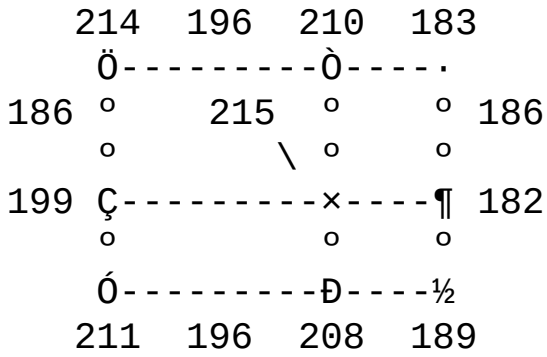
Backup File Header

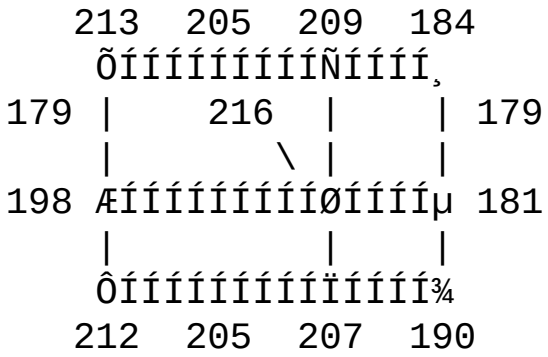
Offset	Size	Description
00	byte	Disk sequence flag: 00 indicates disk is not the last backup diskette FF indicates disk is the last backup diskette
01	byte	Floppy disk sequence number
02	3bytes	Unused
05	64bytes	Full pathname (without drive designator)
45	14bytes	Unused
53	byte	Length of file path name at offset 05 plus 1
54	44Bytes	Unused

Box Drawing Characters









- see ASCII

Code Page and Country Codes

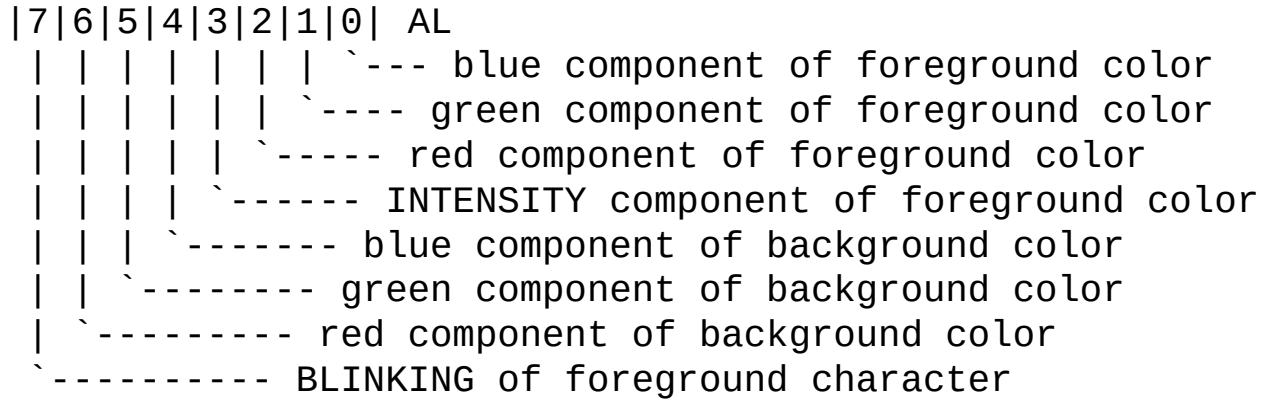
Country	Country Code	Keyboard Code	Valid Code Pages
Arabic	785		437
Australia	061	US	437,850
Belgium	032	BE	437,850
Canada (English)	001	US	437,850
Canada (French)	002	CF	863,850
Denmark	045	DK	865,850
Finland	358	SU	437,850
France	033	FR	437,850
Germany	049	GR	437,850
Hebrew	972		437
Italy	039	IT	437,850
Latin America	003	LA	437,850
Netherlands	031	NL	437,850
Norway	047	NO	865,850
Portugal	351	PO	860,850
Spain	034	SP	437,850
Sweden	046	SV	437,850
Switzerland (French)	041	SF	437,850
Switzerland (German)	041	SF	437,850
United Kingdom	044	UK	437,850
United States	001	US	437,850

- code pages are lookup tables containing the definition of one or more character sets
- contain country specific information
- implemented starting with DOS 3.3

Color Definitions

Definitions found in TURBO C's "conio.h"

0 - BLACK	4 - RED	8 - DARKGRAY	C - LIGHTRED
1 - BLUE	5 - MAGENTA	9 - LIGHTBLUE	D - LIGHTMAGENTA
2 - GREEN	6 - BROWN	A - LIGHTGREEN	E - YELLOW
3 - CYAN	7 - LIGHTGRAY	B - LIGHTCYAN	F - WHITE



- see ANSI for ANSI color definitions

Common Floppy Disk Formats

Disk Type	Supporting DOS Vers	Sides	Trks	Sectors per TRK	Sectors per FAT	Sectors per DIR	Sectors per Cluster	Entries per DIR	Total Sectors
5¬ 160k	(1.0)	1	(40)	8	1	4	1	64	320
5¬ 320k	(1.1)	2	(40)	8	1	7	2	112	640
5¬ 180k	(2.0)	1	(40)	9	2	4	1	64	360
5¬ 360k	(2.0)	2	(40)	9	2	7	2	112	720
5¬ 1.2M	(3.0)	2	(80)	15	7	14	1	224	2400
3« 720k	(3.2)	2	(80)	9	3	7	2	112	1440
3« 1.44M	(3.3)	2	(80)	18	9	14	1	224	2880

FCB - Standard DOS File Control Block

Offset	Size	Description
-7	byte	if FF this is an extended FCB Ø
-6	5bytes	reserved Ø
-1	byte	file attribute if extended FCB Ø
00	byte	drive number (0 for default drive, 1=A:, 2=B:, ...)
01	8bytes	filename, left justified with trailing blanks
09	3bytes	filename extension, left justified w/blanks
0C	word	current block number relative to beginning of the file, starting with zero
0E	word	logical record size in bytes
10	dword	file size in bytes
14	word	date the file was created or last updated

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0| 15,14 (Intel reverse order)
 | | | | | | | | | | `--------- day 1-31
 | | | | | | | `---------------- month 1-12
 `------------------------------ year + 1980
```

16	word	time of last write

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0| 17,16 (Intel reverse order)
 | | | | | | | | | | `---------- secs in 2 second increments
 | | | | `---------------------- minutes (0-59)
 `------------------------------ hours (0-23)
```

18	8bytes	see below for version specific information Ø	
1A	dword	address of device header if character device Ø	
20	byte	current relative record number within current BLOCK	
21	dword	relative record number relative to the beginning of the file, starting with zero; high bit	

omitted if
record length is 64 bytes

DOS 2.x Values for reserved fields at offsets 18h-1Ah Ø

Offset	Size	Description
18	byte	|7|6|5|4|3|2|1|0| | | `---------- unknown | `---------- 1 = open `---------- 1 = logical device
19	word	starting cluster number Ø

DOS 3.x Values for reserved fields at offsets 18h-19h Ø

Offset	Size	Description
18	byte	System File Table (SFT) entry for file Ø
19	byte	attributes Ø |7|6|5|4|3|2|1|0| attributes | | `---------- unknown `--- share status 00 = SHARE not loaded block device 01 = SHARE not loaded characted device 10 = SHARE loaded, remote file 11 = SHARE loaded local file

DOS 3.x with SHARE, local file reserved offsets 1Ah-1Eh Ø

Offset	Size	Description
1A	word	starting cluster number
1C	word	offset within SHARE of sharing record
1E	byte	file attribute

DOS 3.x with SHARE, remote file reserved offsets 1Ah-1Eh Ø

Offset	Size	Description
1A	word	sector number containing directory entry
1C	word	last cluster accessed relative to beginning of file
1E	byte	absolute cluster number of last cluster accessed

DOS 3.x without SHARE reserved offsets 1Ah-1Fh Ø

Offset	Size	Description
1A	byte	((device attribute word low byte) & 0Ch) || (open mode)
1B	word	starting cluster number
1D	word	sector number containing directory entry
1F	byte	number of directory entry within sector

The following are FCB related DOS functions:

INT 21,F	Open file using FCB
INT 21,10	Close file using FCB
INT 21,11	Search for first entry using FCB
INT 21,12	Search for next entry using FCB
INT 21,13	Delete file using FCB
INT 21,14	Sequential read using FCB
INT 21,15	Sequential write using FCB

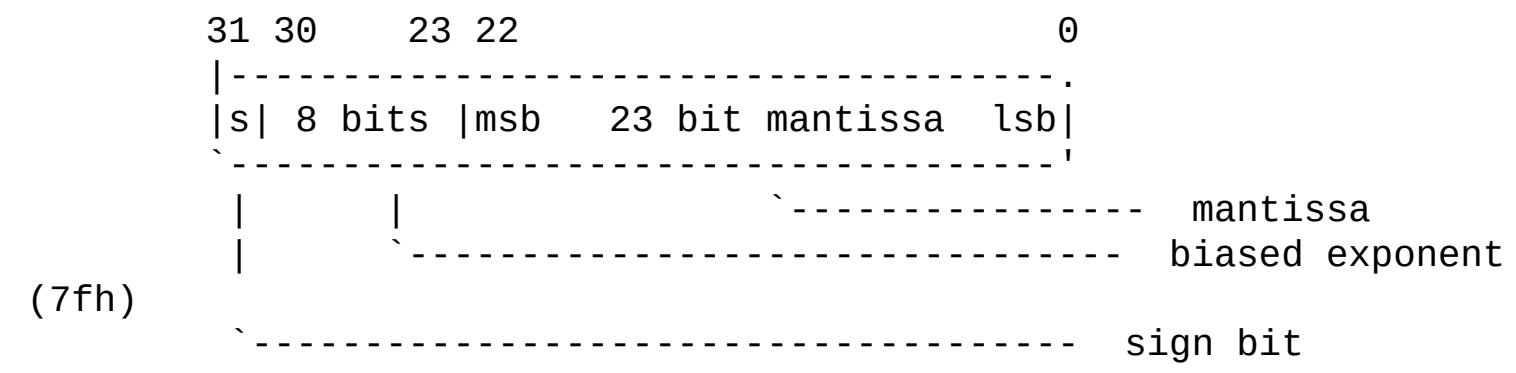
INT 21,16 Create a file using FCB
INT 21,17 Rename file using FCB
INT 21,21 Random read using FCB
INT 21,22 Random write using FCB
INT 21,23 Get file size using FCB
INT 21,24 Set relative record field for FCB
INT 21,27 Random block read using FCB
INT 21,28 Random block write using FCB
INT 21,29 Parse filename for FCB

Ø see Bibliography reference to "Undocumented DOS"

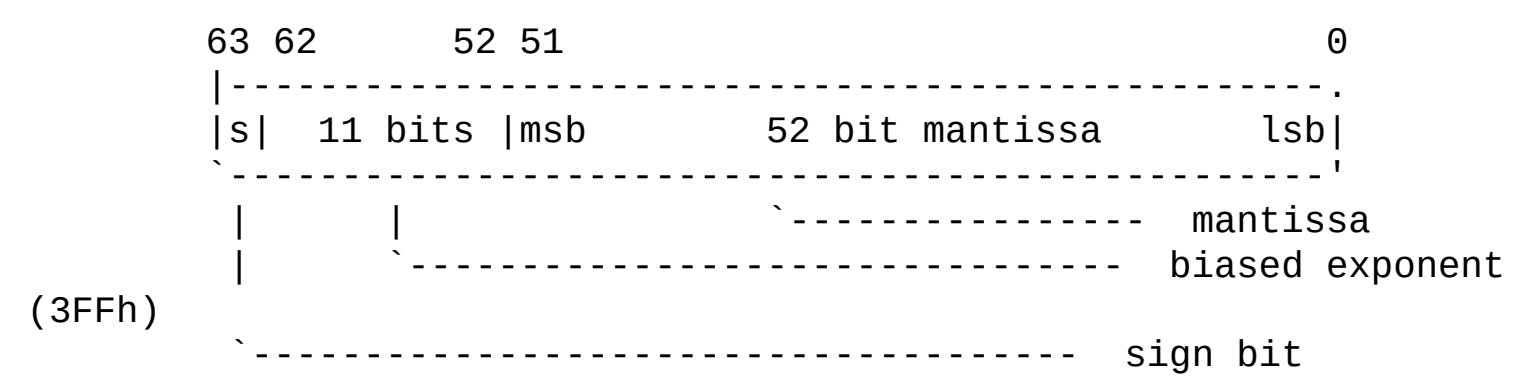
- see XFCB INT 21,52

Floating Point Formats

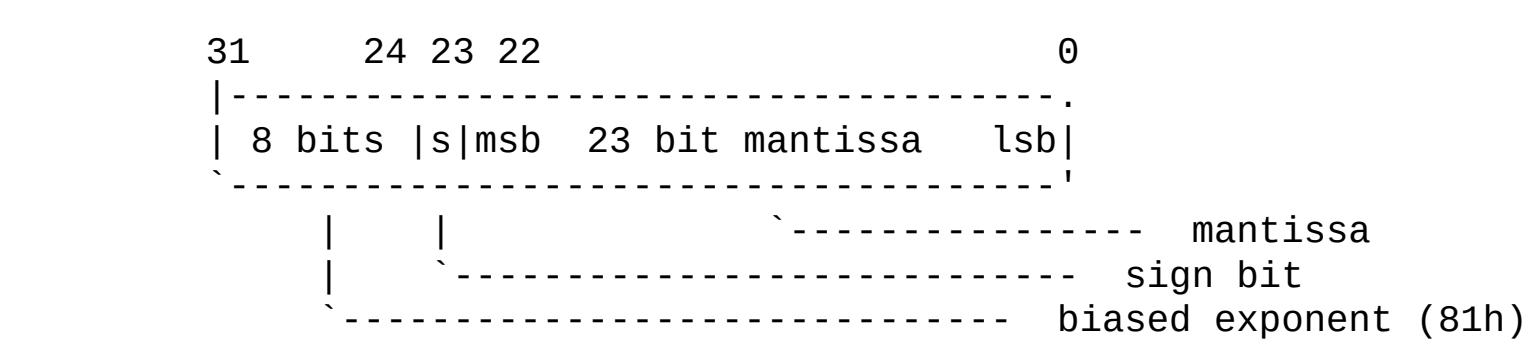
IEEE 4 byte real



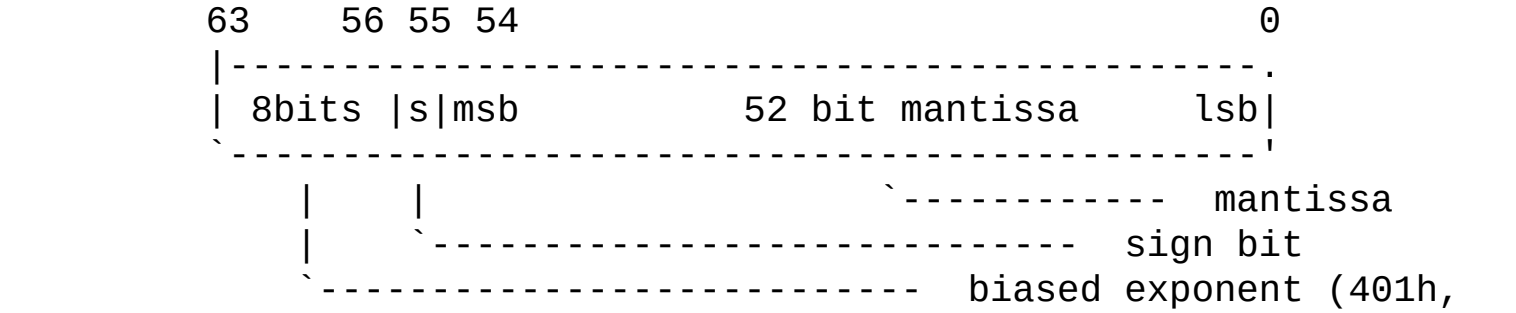
IEEE 8 byte real



Microsoft 4 byte real

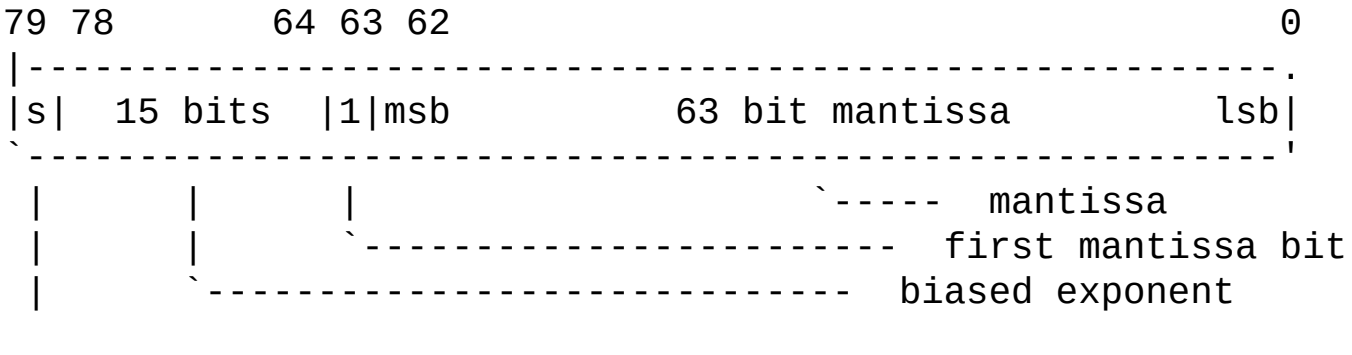


Microsoft 8 byte real (see note below)

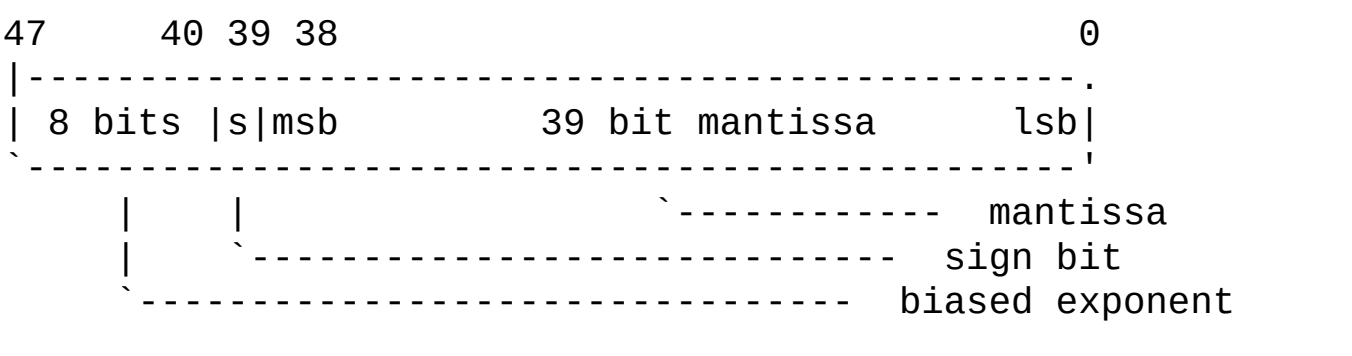


see below)

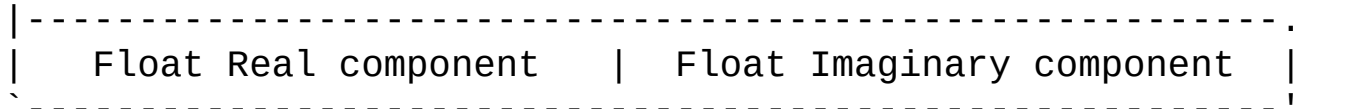
IEEE 10 byte real (temporary real)



Turbo Pascal 6 byte real



Microsoft Fortran Complex number



(each component is either 8 or 16 byte IEEE real)

- sign bit representation: 0 is positive and 1 is negative
- in all float formats except the IEEE 10 byte real, the mantissa is stored without most significant bit; since the state of this bit is known to be set, it is not included and the exponent is adjusted accordingly
- all formats use binary float representation
- memory representation uses 80x86 reverse byte/word order.
- Microsoft languages use the IEEE real formats; BASIC is the only normal user of the Microsoft float format
- Microsoft 8 byte real format has not been verified; several Microsoft publications show an 8 bit exponent instead of

11 bits
and state the BIAS is 401h; the discrepancy is that 8 bits can't
hold the value 401h (requires 11 bits)

True exponent is the exponent value minus the following bias:

81h for Microsoft 4 byte real
401h for Microsoft 8 byte real
7Fh for IEEE 4 byte real
3FFh for IEEE 8 byte real
80h for Turbo Pascal 6 byte real

Size	Range	Significant digits
4 byte real	8.43x10E-37 to 3.37x10E38	6-7
8 byte real	4.19x10E-307 to 1.67x10E308	15-16
10 byte real	3.4x10E-4932 to 1.2x10E4932	19

- see dmsbintoieee() dieeetomsbin() NUMERIC RANGES

Regular ASCII Chart (character codes 0 - 127)

000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	○	(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026	→	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Extended ASCII Chart (character codes 128 - 255)

128	Ç	143	Å	158	×	172	¼	186	‖	200	╚	214	í	228	õ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	Î	229	Õ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	Ï	230	µ	244	¶
131	â	146	Æ	161	í	175	»	189	¢	203	╦	217	┘	231	þ	245	§
132	ä	147	ô	162	ó	176	▒	190	¥	204	╠	218	┌	232	Þ	246	÷
133	à	148	ö	163	ú	177	▓	191	┐	205	═	219	█	233	Ú	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Û	248	°
135	ç	150	û	165	Ñ	179	▌	193	┴	207	¤	221	¦	235	Ù	249	¨
136	ê	151	ù	166	ª	180	┤	194	┬	208	ð	222	Ì	236	ý	250	·
137	ë	152	ÿ	167	º	181	Á	195	├	209	Ð	223	▀	237	Ý	251	¹
138	è	153	Ö	168	¿	182	Â	196	─	210	Ê	224	ó	238	¯	252	³
139	ï	154	Ü	169	®	183	À	197	┼	211	Ë	225	ß	239	´	253	²
140	î	155	ø	170	¬	184	©	198	ã	212	È	226	Ô	240	-	254	■
141	ì	156	£	171	½	185	╣	199	Ã	213	ı	227	Ò	241	±	255	
142	Ä	157	Ø														