Programming PCs circa 2000: A Cheatsheet

18 Jun 2000

Table of Contents

1. System Variables
    - 1.1. BIOS variables located in low memory
    - 1.2. CMOS Variables

2. Devices and their handling
    - 2.1. Mouse
    - 2.2. MS-DOS
    - 2.3. LPT
    - 2.4. Drive
        - 2.4.1. Drive Specification
        - 2.4.2. Interrupt 13h
    - 2.5. Communication
        - 2.5.1. COMM
        - 2.5.2. Modem Commands
    - 2.6. Keyboard
    - 2.6.1. Typematic Delay
    - 2.6.2. Keyboard Port Handling

3. Text & Graphics
    - 3.1. Text Mode
        - 3.1.1. VGA 254x62
        - 3.1.2. Fonts & Colors in Text
    - 3.2. Graphics
        - 3.2.1. Interrupt 10h - BASICS
        - 3.2.2. DAC / Ports for VGA & Xmodes
        - 3.2.3. VESA 1.2 SVGA
        - 3.2.4. VESA 2.0 SVGA - Linear Framebuffer
    - 3.3. Colors
    - 3.4. 3D GRAPH
    - 3.5. 2D GRAPH

4. System Core
    - 4.1. Mem Access
        - 4.1.1. Basics
        - 4.1.2. Extended Memory Access
    - 4.2. BIOS Access
    - 4.3. PORTS Access
    - 4.4. Timer
        - 4.4.1. Reprogramming the timer
        - 4.4.2. Microsectimer
    - 4.5. Interrupts
        - 4.5.1. Overall
        - 4.5.2. Hooking the interrupts

5. Miscellaneous
    5.1. ASCII Codes
        5.1.1. Borders
        5.1.2. DOS Codes
        5.1.3. Port scancodes
    5.2. Formats
        5.2.1. COM
        5.2.2. EXE
        5.2.3. SYS
        5.2.4. BMP
        5.2.5. PCX
        5.2.6. PIF
        5.2.7. WAV
    5.3. Assembler
        5.3.1. 8086 REGISTERS
        5.3.2. 8086 MNEMONICS
    5.4. Other

Notation:

[p]x (port access, read or write)
[x:x] (memory access, read or write)
[ax] (register, read or write)
[int] (interrupts)
>(return value)
[b], [w], [dw] (sizes: byte, word, double word, can be multiple)
+x (offsets)
[%]x (bit fields)

1. System Variables

1.1. BIOS variables located in low memory

[0h:417h] [b] Shift Status:
	[%]0=Right Shift
	[%]1=Left Shift
	[%]2=Ctrl
	[%]3=Alt
	[%]4=ScrollLock On
	[%]5=NumLock On
	[%]6=CapsLock On
	[%]7=Insert On
[0h:418h] [b] Extended Shift Status (101/102 keys):
	[%]0=Left Ctrl
	[%]1=Left Alt
	[%]2=SysReq Hit
	[%]3=Pause On
	[%]4=ScrollLock Hit
	[%]5=NumLock Hit
	[%]6=CapsLock Hit
	[%]7=Insert Hit
[0h:41Ah] [w] Keyboard buffer head Addr
[0h:41Ch] [w] Keyboard buffer tail Addr
[0h:41Eh] 32[b] Keyboard buffer
[0h:46Ch] [l] Timer Ticks, in 55ms increments
[0h:470h] [b] Timer Overload
[0h:471h] [b] [%]7=Break Hit
[0h:497h] [b] [%]0-2=LEDs
[F000h:FFF0h] 5[b] FAR JMP to begin POST (Power-On SELF test)
[F000h:FFF5h] 8[b] ROM BIOS production date

1.2. CMOS Variables

[p]70h=n, the byte ordinal
[p]71h=value read/write (remember the checksum)

Byte ordinals:
0h (second)
2h (minute)
4h (hour)
6h (day of week, with 1=sunday)
7h (day of month)
8h (month)
9h (last two digits of the year)
2eh-2fh (checksum: fields 10h-20h summed up, 16bit, High/Low order)
32h (century)

All values in BCD ([%]7-4 first digit, [%]3-0 second)

2. Devices and their handling

2.1. Mouse

[int]=33h
[ax]=0h >[ax] Mouse Installed? (<>0=Yes, 0=No) >[bx] Button #
[ax]=1h Show Mouse
[ax]=2h Hide Mouse
[ax]=3h >[bx] Buttons state, >[cx,dx] Position
[ax]=4h, [cx,dx] Set Position
[ax]=7h, [cx,dx] Limit mouse horizontal movement
[ax]=8h, [cx,dx] Limit mouse vertical movement
[ax]=ah Set mouse look (text) [bx]=0; [cx]=AND mask; [dx]=XOR mask. [cx],[dx]= [%]15: flash, [%] 14-12: bg color, [%]11:bright, [%]10-8 color, [%]7-0 code or [bx]=1; [cx]=start line; [dx]=end line
[ax]=bh Read delta >[cx] = dx >[dx] = dy (integer)
[ax]=1ah Set sensitivity [bx]=sy; [cx]=sx; [dx]=speed

2.2. MS-DOS

[int]=21h
[ah]=0h Terminate program, free memory. [cs]=segment of program head
[ah]=3h >[al] code of RS 232 char

[ah]=4h [dl]=code to put to RS 232
[ah]=5h print [dl] char
[ah]=ah Buffered input [ds:dx]=buffer addr, >0th byte contains length >2nd byte contains # chars
[ah]=bh >[al]=ffh if keyboard buffer is not empty
[ah]=eh Set new drive [dl]=Drive 0=A..
[ah]=25h Set new Int, [al]=Int #, [ds:dx] - Addr
[ah]=2ah Get date >[al]=Day of week (0=Sunday), >[dl]=day, >[dh]=month, >[cx]=year
[ah]=2bh Set date ([dl], [dh], [cx]), >[al]=ffh bad date
[ah]=2ch Get time >[dl]=s/100, >[dh]=s, >[cl]=min, >[ch]=hour
[ah]=2dh Set time ([dl], [dh], [cl], [ch]) >[al]=ffh bad
[ah]=31h TSR Leave, [al]=end code, >[dx]=paragraphs of reserved memory
[ah]=35h Get Int, [al]=Int#, [es:bx]=buffer
[ah]=39h MkDIR [ds:dx] - ASCIIZ
[ah]=3ah RmDIR [ds:dx] - ASCIIZ
[ah]=3bh CD [ds:dx]
[ah]=3ch New File [ds:dx] >[ax]-handle
[ah]=3dh Open File [ds:dx], >[ax]-handle
[ah]=3eh Close File [bx]-handle
[ah]=3fh Read from file [bx]=handle, [cx]=#byte, [ds:dx]=buffer
[ah]=40h Write to file [bx]=handle, [cx]=#byte, [ds:dx]=buffer
[ah]=41h Del [ds:dx] - ASCIIZ
[ah]=42h Seek in File [al]=0(abs),1(cur),2(end), [bx]=Handle, [cx:dx] - bytes#
[ah]=43h Attr [al]=0(read)/1(write), [ds:dx] - ASCIIZ, [cx] - Attr ([%]0:readonly, [%]1:hidden, [%] 2:system, [%]3:label, [%]4:dir, [%]5:archive) >[cl] - attr
[ah]=48h Alloc [bx]=paragraphs# >[ax]=segment#
[ah]=49h Free [es]=segment
[ah]=4ah Realloc [bx]=new pars, [es]=segment
[ah]=4bh Load COM/EXE [al]=0(load+run),3(load), [ds:dx] - ASCIIZ
[ah]=4ch Terminate, Free, [al]=Code
[ah]=56h MoveFile [ds:dx]=from, [es:di]=to
[ah]=57h Modify Time/Date of File [al]=0(read)/1(write) [bx]=handle, [cx]=new time, [dx]=new date, > [cx],[dx]-time,date to read

2.3. Printer Port (LPT)

[int]=17h
Pins: 1=StobeImpulse;2-9=Data;10=ACK;11=Busy;12=NoPaper;13=Ready; 14=LF after CR; 16=Zero; 17=Ground; 18=+5V DC;19-25 Mass
[ah]=0h; Send [al]=Char; [dx]=Port # (0=LPT1) >[ah] status ([%]0: no response; [%]1-2:not used; [%]3:error; [%]4:ready; [%]5:nopaper; [%]6:response; [%]7:free)
[ah]=1h; Init [dx]=Port # >[ah] (see 0h)
[ah]=2h; Read status [dx]=Port # >[ah] (see 0h)

2.4. Drive

2.4.1. Drive Specification

cyl 0+ ; head 0-1 ; sector 1-9
First sector: BootSector (loaded to [0h:7C00h]):
+0h Jump Instruction (3[b]) - FAR JUMP..
+3h System Info (8[b])
+bh BPB: bytes per sector (2[b]), sectors per cluster (1[b]), sectors for loader (2[b]), FAT copies # (1[b]), # of positions in root (2[b]), sectors all # (2[b]), ID (1[b]) = f0h if HD Disk, sectors per cluster (2[b]), sectors per cylinder - HDD=17, Drive=9 (2[b]), headno (2[b]), hidden sectors (2[b])

2.4.2. Interrupt 13h

[int]=13h
[ah]=0h; Init disc [dl]=Diskno (0=A.. 80h=first hard..) >[ah] status (0h=OK; 1h=non-recognized cmd; 2h=bad address;3h=write protect;4h=bad disk #; 5h=init err;8h=DMA error;9h=DMA overload;ah=bad sector#;bh=bad cyl#; 10h=CRC err;11h=ECC used;20h=controller err;40h=can't find cyl; 80h=no response; aah=drive not ready; bbh=other; cch=write error; ffh=bad operation code)
[ah]=1h; Get status of last oper [dl]=Disc# >[ah] Status >[cf] - Is Err?
[ah]=2h; Read Sectors [al]=sectors#; [ch]=cyl#; [cl]=sector#(if cyl#>255, [%]6-7 of [cl] is extension of [ch]); [dl]=Disc#; [dh]=Head#; [es:bx]=buffer; >[ah]=status; >[cf] - Is Err?
[ah]=3h; Write Sectors (as 2h) >[ah]=status; >[cf] - Is Err?
[ah]=4h; Verify [al]=Sectors#; [cx]=Cyl/Sec (see 2h); [dl]=Drive; [dh]=Head; >[ah],>[cf] (as 2h)
[ah]=8h; Get hard info [dl]=Disc No; >[cx] (max cyl - see 2h) >[dl] #of hard drives >[dh] max head > [ah] status >[cf] Is Err?

2.5. Communication

2.5.1. COMM

[int]=14h
Pins: 1=Mass;2=TX Out;3=RX In;4=DSR;7=Mass Signal;8=DCD;20=DTR;22=Bell

[ah]=0h;[al]=Params ([%]0-1: Word length = 10b=7bits, 11b=8bits; [%]2: Stop bits 0b=1bit; [%]3-4:Parity control = 00,10(no), 01(parity), 11(non-parity); [%]5-7 bauds = 000=110..111=9600); [dx]=port # (0=COM1) >[ah]=Status ([%]0: Ready;[%]1: Bad char#;[%]2: Parity Err; [%]3: Border Err; [%]4: Break Detected; [%]5: Empty Buffer; [%]6:Empty Move;[%]7:TimeLimit Exc;
[ah]=1h; Send Char: [al]=Char Code; [dx]=Port # >[ah] Status (see 0h)

[ah]=2h; Read Char: [dx]=Port # >[al]=Char >[ah] Status (see 0h)
[ah]=3h; GetStatus [dx]=Port # >[ax]=Status ([ah] as in 0h,[al] = [%]0:Ready to send chg; [%]
1:Ready to rcv chg; [%]2:Rcv Req chg; [%]3:Sig chg; [%]4:Send Ready; [%]5:Receive Ready; [%]
6:Rcv Req; [%]7:Signal

2.5.2. Modem Commands

AT.. (max 40 chars, upper or lower, not mixed), one line
AT..enter command
A/ last command
+++ (after OK - data>>commands)
AT0 enter (commands>>data)
A answer incoming
Bx connection x=v.22/bell212a/v.23/v.
23/300/1200/2400/4800/9600/14400/16800/19200/21600/28800/31200/33600 (0-15)
D.. 0-9, A-D, #, * call
P pulse
R beginning of answer-mode command
T tone
W wait for operator
, pause
@ 5seconds of silence
! disconnect
; command tribe after dialing
S=x call one of 4 memorized numbers (0-3)
^ request signal (xfer)
Ex Echo of commands (0/1)
Fx Local echo (0/1)
Hx Leave/take line (0/1)
Ix 0 product code, 1 CRC, 2 memtest, 3 version, 4 version, 5 country code
Lx speaker power (0-3)
Mx 0 speaker off, 1 on until connect, 2 on, 3 off while calling, on until conn
Nx auto-transmission (0/1)
Ox 0 data tribe on, 1 data tribe on + modem fit, 2 +fallforward, 3 +fallback
P pulse
Qx messages to computer ~ (0/1)
Sx? get register (0-97)
Sx=y set register (0-97) to (0-255)
T tone
Vx 0 short, 1 long answers
Xx 0 Hayes mesgs, 1 +connect, 2+nodial, 3+busy, 4 all
Yx reaction on long pause (0/1)
Zx reset to profile x (0/1)
&F load init config
&Gx 0 operator echo filtering, 1 send 550Hz, 2 send 1800Hz
&L0 back to dialing
&Zx=y remember number y in x-th (0-3)
&Wx save config to x (0/1)
&Yx auto load config from x (0/1)

S=0 bells before answering	0-255	#	0
1 #of bells	0-255	#	0
2 code of Esc	0-255	ASCII	43
3 code of LF	0-127	ASCII	13
4 code of CR	0-127	ASCII	10
5 code of BS	0-32,127	ASCII	8
6 time to wait for operator	2-255	s	2
7 time to wait for line call	1-255	s	45
8 pause time	0-255	s	2
9 response time	1-255	s/10	6

10 time lost<>disconnect	1-255	s/10	14
11 tone length	50-255	ms	95
12 code for safety	20-255	s/50	50

codes: 0 OK; 1 CONNECT; 2 RING; 3 NO CARRIER; 4 ERROR; 5; CONNECT (1200); 6 NO DIALTONE; 7 BUSY; 8 NO ANSWER; 10+ CONNECT (..)

2.6. Keyboard

2.6.1. Typematic Delay

[int]=16h
[ax]=0305h; [bh]=Delay1 (0=250ms.. 3=1000ms); [bl]=Delay2 (00h=30/s..1fh=2/s)
[p]60h (var) 10ms port [p]60h (value). For Delay=f3h/0-4:Delay2,5-6:Delay1,7=0

2.6.2. Keyboard Port Handling

[p]60h edh/0-2 LEDs (not locks, only LEDs, for locks, See Bios Variables)
ffh Init, feh Reset last trsm, f6h Set Default, f5h Lock Keyboard, f4h Unlock, eeh Echo. Sends eeh

Every time a Key is pressed: Read [p]60h, Read Control Register ([p]60h), OR it with 82h, write result to 61h, AND it with 7Fh, [p]20h=20h

3. Text & Graphics

3.1. Text Mode

3.1.1. VGA 254x62

scroll: x=0..174*9, y=0..37*16;

DJGPP (gcc clone)

```
unsigned int f;
f=(x/9)+(y/16)*_farpeekw(_dos_ds,0x00404);
outp(0x3d4,0xc);
outp(0x3d4+1,(f >> 8));
outp(0x3d4,0xd);
outp(0x3d4+1,(f & 255));
for(;;) if (!(inp(0x3da) & 8)) break;
for(;;) if (inp(0x3da) & 8) break;
f=inp(0x3da);
outp(0x3c0,0x13);
outp(0x3c0,(unsigned short)((unsigned short)(x % 9)-1));
outp(0x3c0,32);
outp(0x3d4,8);
outp(0x3d4+1,(y % 16));
// SET MODE

f=inp(0x3da);
outp(0x3c0,16);
```

```
outp(0x3c0,(255 & inp(0x3c1)));
outp(0x3c0,32);
outp(0x3d4,0x13);
outp(0x3d4+1,(254/2));
_farpokew(_dos_ds,0x00404,254);
```

3.1.2. Fonts & Colors in Text

init DOS memory - 48h of 21h, 49h of 21h
init colors: for(a=0;a<15;++a) {[ax]=1000h, [bh]=a; [bl]=a; [int]=10h;}

3.2. Graphics

3.2.1. Interrupt 10h - BASICS

[int]=10h
11h 640x480x2 [video starts at a000h:0h]
13h 320x200x256 [video starts at a000h:0h]
03h 80x25t [characters start at b800h:0h]
SetColor [p]3c6h=ffh, [p]3c8h=color, [p]3c9=seq RGB
GetColor [p]3c6h=ffh, [p]3C7h=color, >[p]3c9 seq RGB
[ah]=0h, [al]=tribeno
[ah]=1h, [ch]=first curs line, [cl]=last curs line ([ch]=[cl]=20h hidecurs, 0ch-0dh def)
[ah]=2h, Set curs pos [bh]=page#, [dh]=Y, [dl]=X (from 0!)
[ah]=3h, Get curs pos [bh]=page #, >[dh]=Y, >[dl]=X, >[ch]=first curs line, >[cl]=last
[ah]=5h, Select active page [al]=page#
[ah]=6h, ScrollUp [al]=line#, [bh]=attr for inserted line, [ch]=Y1, [cl]=X1, [dh]=Y2, [dl]=X2
[ah]=7h, ScrollDown (as 6h)
[ah]=8h, Read char [bh]=page#, >[al]=char, >[ah]=attr
[ah]=9h, Put char [bh]=page#, [al]=code, [bl]=attr. For graphics [%]7=1 (XOR), [cx]=rep#
[ah]=ah, Put char no attr [bh]=page#, [al]=code, [bl]: [%]7=1 (XOR),[cx]=rep#
[ah]=ch, Put pixel [al]=col (80h=XOR, no 13h), [bh]=page#, [cx]=X, [dx]=Y
[ah]=dh, Read pixel [bh]=page#, [cx]=X, [dx]=Y, >[al]=col
[ah]=fh, Get mode# >[ah]=width (chars), >[al]=tribe#, >[bh]=page#
[ax]=1001h, Set Overscan [bh]=[%]0-5 new register no
[ax]=1003h, Flash/Bright [bl]=0 [%]7 bright, [bl]=1 [%]7 flash
[ax]=101Bh, RGB->Gray [bx]=first col, [cx]=col#
[ax]=1100h, User->Gener [bh]=bytesperchar, [bl]=block# (0-7), [cx]=chars#, [dx]=firstchar, [es:bp] buffer
[ax]=1103h, Select font [bl]=[%]5,3,2:block# for those with [%]3 of attr set, [%]4,1,0: block# for those with [%]3 of attr not set
[ax]=1130h, Get standard font [bh]=0h (8x8 in RAM), 2h (8x14), 3-4h (8x8 in ROM), 5h (9x14 ROM), 6h (8x16 ROM), 7h (9x16 ROM) >[cx]=bytes per char, >[dl]=line#-1, [es:bp]=char addr

3.2.2. DAC / Ports for VGA & Xmodes

[p]3dah >[%]0=1(drawing)/0(VRT or HRT), [%]3=1(VRT)/0(drawing)
[p]3ceh=03h, [p]3cfh [%]0-2: 000b-no transpose, 001b-111b-transpose 1-7 bits, 3-4b: 00b DirectDraw, 01b AND, 10b OR, 11b XOR

256x256(QMode)=(3c2h,e300h,3d4h,5f00h,3d4h,3f01h,3d4h,4002h,3d4h,8203h,3d4h,4a04h,3d4h,9a05h,3d4h,2306h,3d4h,b207h,3d4h,0008h,3d4h,6109h,3d4h,0a10h,3d4h,ac11h,3d4h,ff12h,3d4h,

```
   2013h,3d4h,0014h,3d4h,0715h,3d4h,1a16h,3d4h,e317h,3c4h,0101h,3c4h,0604h,3ceh,4005h,3ceh,
   0506h,3c0h,4110h,3c0h,0013h);
320x200(4pages)=(3c2h,6300h,3d4h,5f00h,3d4h,4f01h,3d4h,5002h,3d4h,8203h,3d4h,5404h,3d4h,
   8005h,3d4h,bf06h,3d4h,1f07h,3d4h,0008h,3d4h,4109h,3d4h,9c10h,3d4h,8e11h,3d4h,8f12h,3d4h,
   2813h,3d4h,0014h,3d4h,9615h,3d4h,b916h,3d4h,e317h,3c4h,0101h,3c4h,0604h,3ceh,4005h,3ceh,
   0506h,3c0h,4110h,3c0h,0013h);
320x240(squarepix)=(3c2h,e300h,3d4h,5f00h,3d4h,4f01h,3d4h,5002h,3d4h,8203h,3d4h,5404h,3d4h,
   8005h,3d4h,0d06h,3d4h,3e07h,3d4h,0008h,3d4h,4109h,3d4h,ea10h,3d4h,ac11h,3d4h,df12h,3d4h,
   2813h,3d4h,0014h,3d4h,e715h,3d4h,0616h,3d4h,e317h,3c4h,0101h,3c4h,0604h,3ceh,4005h,3ceh,
   0506h,3c0h,4110h,3c0h,0013h);
360x360=(3c2h,6700h,3d4h,6b00h,3d4h,5901h,3d4h,5a02h,3d4h,8e03h,3d4h,5e04h,3d4h,8a05h,
   3d4h,bf06h,3d4h,1f07h,3d4h,0008h,3d4h,4009h,3d4h,8810h,3d4h,8511h,3d4h,6712h,3d4h,2d13h,
   3d4h,0014h,3d4h,6d15h,3d4h,ba16h,3d4h,e317h,3c4h,0101h,3c4h,0604h,3ceh,4005h,3ceh,0506h,
   3c0h,4110h,3c0h,0013h);
360x480=(3c2h,e700h,3d4h,6b00h,3d4h,5901h,3d4h,5a02h,3d4h,8e03h,3d4h,5e04h,3d4h,8a05h,3d4h,
   0d06h,3d4h,3e07h,3d4h,0008h,3d4h,4009h,3d4h,ea10h,3d4h,ac11h,3d4h,df12h,3d4h,2d13h,3d4h,
   0014h,3d4h,e715h,3d4h,0616h,3d4h,e317h,3c4h,0101h,3c4h,0604h,3ceh,4005h,3ceh,0506h,3c0h,
   4110h,3c0h,0013h);


// SET MODE
int ScrOfs;
int i;
unsigned VGAPort;
short VGARegister, VGAValue, test;

[p]3d4h=11h; test=[p]3d5h & 7Fh; [p]3d4h=11h; [p]3d5h=test;
for(i=1;i<=25;++i)
        { VGAPort=Mode[2*i-2]; VGARegister=Lo(Mode[2*i-1]);
        VGAValue=Hi(Mode[2*i-1]);
        if(VGAPort==3c0h) {test=[p]3dah; [p]VGAPort=(VGARegister | 20h);
                [p]VGAPort=VGAValue}
        if(VGAPort==3c2h,3c3h) {[p]VGAPort=VGAValue}
      else {[p]VGAPort=VGARegister; [p]VGAPort+1=VGAValue} }

CLEAR: [p]3c4h=2h; [p]3c5h=fh; [es:di]=[a000h,0h]; [cx]=32767; [ax]=0;
        Until([cx]>0): {[es:di]=0[w]; [di]+=2;}

PLOT(X,Y,W(/4!!),C): [p]3c4h=2h; [p]3c5h=1<<Lo(X & 3);
        [a000h:(X>>2)+W*Y+ScrOfs]=C[b]

SCROLL: [p]3d4h=ch; [p]3d5h=Hi(OFFS(/4!!)); [p]3d4h=dh;  [p]3d5h=Lo(OFFS(/4!!));
```

3.2.3. VESA 1.2 SVGA

```
Tribes: 100h (640x400x8), 101h (640x480x8), 103h (800x600x8), 105h (1024x768x8), 107h
   (1280x1024x8), 108h (80x60t), 109h (132x25t), 10ah (132x43t), 10bh (132x50t), 10ch (132x60t),
   10dh+x (320x200x15/16/32), 110h+x (640x480x15/16/32), 113h+x (800x600x15/16/32), 116h+x
   (1024x768x15/16/32), 119h+x (1280x1024x15/16/32)
[int]=10h
[ax]=4f00h Return SVGA Info [es:di] - 256b buffer (*) > [ah]=0h OK, 1h Err >[al]=4fh is BIOS inst.
[ax]=4f01h Return mode Info [cx]- tribe, [es:di]- 256b buffer (**) >[ax]
[ax]=4f02h Set SVGA mode [bx]=[%]0-13 mode #, [%]14:LFB, [%]15:Do not cls >[ax]
[ax]=4f03h Return SVGA mode >[bx]=[%]0-14 mode #, >[ax]
[ax]=4f05h Set Bank [bh]=0(set),1(get), [bl]=window(0/1), [dx]=bankno
      >[dx]=bankno, >[ax]
[ax]=4f06h Set width [bl]=0(set),1(get) [cx]=width >[ax], >[bx]=bytesperline, >[cx]=width (pixels), >
   [dx]=max line#
```

[ax]=4f07h Set display start [bh]=0; [bl]=0(set),1(get) [cx]=X offset, [dx]=Y offset, >[cx]=X offset (for read) >[dx]=Y offset (for read) >[ax]

(*)
+0h [l] "VESA"
+4h [w] version no
+6h 2[w] pointer to copyright word in ROM
+ah 4[b] other info (0 usual)
+eh 2[w] pointer to tribe no, finished by ffffh
+12h 238[b] reserved
(**)
+0h [w] [%]0:tribe available, [%]1:other info avail, [%]2: can use (low10h) [%]3: 0(col)/1(mono), [%]4: 0(text)/1(graph)
+2h [b] A window attr - [%]0: exist, [%]1:can read, [%]2:can write
+3h [b] B window attr - as above
+4h [w] granularity=min space between banks (kb)
+6h [w] win size (kb)
+8h [w] seg of A window
+ah [w] seg of B window
+ch 2[w] far ptr to function switching banks
+10h [w] bytes per one logic line
+12h [w] width (pixels)
+14h [w] height (pixels)
+16h [b] matrix width (pixels)
+17h [b] martix height (pixels)
+18h [b] memory fields #
+19h [b] bits per pixel
+1ah [b] banks no
+1bh [b] =0h text, =1h (CGA), =2h(16col, fields) =4h(packed)
+1ch [b] bank size (kb)
+1dh [b] pages no

3.2.4. VESA 2.0 SVGA - Linear Framebuffer

DJGPP

```
__dpmi_regs regs;
__dpmi_meminfo info,buffer;

unsigned width,height,bpp; // BYTES per pixel
unsigned graphsel,buffersel;
unsigned long addr,size;
unsigned char *data;
/*
GRAPHICS MEMORY ACCESSING:

1. DirectDraw (Byte, Word, DWord)

 _farpokeb(graphsel,offset,(unsigned char)byte  );
 _farpokew(graphsel,offset,(unsigned)     bytes2);
 _farpokel(graphsel,offset,(unsigned long)bytes4);

 (unsigned char)byte  =_farpeekb(graphsel,offset);
 (unsigned char)bytes2=_farpeekw(graphsel,offset);
 (unsigned char)bytes4=_farpeekl(graphsel,offset);

2. Buffered DirectDraw (Byte, Word, DWord and CopyAtOnce)

 DirectDraw as [1.], but buffersel instead of graphsel, then:
 movedata(buffersel,offset1,graphsel,offset2,size1) to copy buffer->screen
```

```
 movedata(graphsel,offset1,buffersel,offset2,size2) to copy screen->buffer

 offset1=offset2=0; size1=size2=size to copy entire screen

3. Block CopyAtOnce

 Alloc data with 'malloc', then fill it as you wish (memset, memcpy..), then:
 movedata(_my_ds(),(int)data,graphsel,0,size) to copy data->screen
 movedata(graphsel,0,_my_ds(),(int)data,size) to copy data->screen

*/

// SET GRAPH - LFB

 int seg,ret;
 long x;

 seg=__dpmi_allocate_dos_memory(16,&ret);

 regs.x.ax=0x4F01;
 regs.x.cx=mode;
 regs.x.es=seg;
 regs.x.di=0;
 __dpmi_int(0x10,&regs);

 if(regs.x.ax!=0x004F) return(1);
 width=_farpeekw(_dos_ds,seg*16+0x12);
 height=_farpeekw(_dos_ds,seg*16+0x14);
 bpp=((_farpeekb(_dos_ds,seg*16+0x19))>>3);
 addr=_farpeekl(_dos_ds,seg*16+0x28);
 size=((long)width*(long)height*(long)bpp);

 __dpmi_free_dos_memory(ret);

 graphsel=__dpmi_allocate_ldt_descriptors(1);
 info.address=addr; info.size=size;
 __dpmi_physical_address_mapping(&info);
 __dpmi_lock_linear_region(&info);
 __dpmi_set_segment_base_address(graphsel,info.address);
 __dpmi_set_segment_limit(graphsel,info.size-1);

 buffersel=__dpmi_allocate_ldt_descriptors(1);
 buffer.address=0; buffer.size=size;
 __dpmi_allocate_memory(&buffer);
 __dpmi_lock_linear_region(&buffer);
 __dpmi_set_segment_base_address(buffersel,buffer.address);
 __dpmi_set_segment_limit(buffersel,buffer.size-1);

 regs.x.ax=0x4F02; regs.x.bx=mode;
 __dpmi_int(0x10,&regs);

 return(0);

FREE ALL MEMORIES

 regs.x.ax=0x0003;
 __dpmi_int(0x10,&regs);

 __dpmi_free_ldt_descriptor(graphsel);
 __dpmi_free_ldt_descriptor(buffersel);
 __dpmi_free_memory(buffer.handle);
```

3.3. Colors

RGB system R(0-255)Red G(0-255)Green B(0-255)Blue
HSV system: Hue=0(Red), 2PI/3 (Green), 4PI/3 (Blue), angle in radians Saturation(0-1), Value(0-1)
Generate Rainbow: y=H*exp(-((x-x0)/w)^2), H-max of color (255), W=75, x=color#, x0=color#, for which function is maximum.

```
for(i=1;i<256;++i)
{
 p[i].red=255*exp(-1.0*pow(((double)i-64)/75,2.0));
 p[i].green=255*exp(-1.0*pow(((double)i-128)/75,2.0));
 p[i].blue=255*exp(-1.0*pow(((double)i-192)/75,2.0));
}
```

RGB->gray Y=0.30*R+0.59*G+0.11*B;
HSV->RGB (H=0..2PI, S=0..1, V=0..1)
S=0 > RGB=(V,V,V);
H=2PI > H=0
H=3H/PI; p=255*V*(1-S); q=255*V*(1-(S*{H})); t=255*V*(1-(S*(1-{H})));
v=255*V;
[H]=0 > RGB=(V,t,p); [H]=1 > RGB=(q,V,p); [H]=2 > RGB=(p,V,t);
[H]=3 > RGB=(p,q,V); [H]=4 > RGB=(t,p,V); [H]=5 > RGB=(V,p,q);

3.4. 3D Graphics

Rotate OX: Yn=YCos(X)-ZSin(X), Zn=YSin(X)+ZCos(X)
OY: Xn=XCos(Y)-ZSin(Y), Zn=XSin(Y)+ZCos(Y)
OZ: Xn=XCos(Z)-YSin(Z), Yn=XSin(Z)+YCos(Z)
ScX(or Y)=256*X(or Y)/(Z+Dist)+D_X(or D_Y);

Gouraud: angle = normal to surface and vector pointed to light source

Normal: N=VxU, where V & U are egdal vectors

3.5. 2D Graphics

2D Rotate: Xn=XCos(A)-YSin(A), Yn=YCos(A)+XSin(A)

Breshenham Line: dx=Abs(x2-x1), dy=y2-y1, Delta1=2dy-dx, Delta2=2dy, Delta3=2(dy-dx),Delta4=2(dy+dx)
For each point:
If(Delta1>=0 && dy<=0) Delta1+=Delta2;
If(Delta1>=0 && dy>0) {++y, Delta1+=Delta3}
If(Delta1<0 && dy>=0) Delta1+=Delta2;
If(Delta1<0 && dy<0) {--y, Delta1+=Delta4}

4. System Core

4.1. Mem Access

4.1.1. Basics

DJGPP

```
#include <sys/farptr.h>
#include <go32.h>
_farpokeb(_dos_ds,seg*16+ofs,b);
=farpeekb(_dos_ds,seg*16+ofs);
```

or

```
#include <crt0.h>
#include <sys/farptr.h>

void *makeptr(unsigned seg, unsigned ofs)
{
   if ( !(_crt0_startup_flags & _CRT0_FLAG_NEARPTR) )
   if (!__djgpp_nearptr_enable ())
   return (void *)0;
   return (void *) (seg*16 + ofs + __djgpp_conventional_base);
}
memset(makeptr(seg,ofs),qu,b);
```

PASCAL

```
Mem[seg:ofs]=z;
z=Mem[seg:ofs]
```

OTHER GCC

```
unsigned char far *x=(unsigned char far *)0xA0000000;
x[0]=y;  z=x[0];
```

or

```
#include <dos.h>
pokeb(seg,off,b);
b=peekb(seg,off);
```

4.1.2. Extended Memory Access

DJGPP

```
// INIT
__dpmi_meminfo info;
int selector;
selector=__dpmi_allocate_ldt_descriptors(1);
info.address=physicaladdress;
info.size=physicalsize;
__dpmi_physical_address_mapping(&info);
__dpmi_lock_linear_region(&info);
__dpmi_set_segment_base_address(selector,info.address);
__dpmi_set_segment_limit(selector,info.size-1);
```

```
// FREE: __dpmi_free_ldt_descriptor(selector);
// PEEK: _farpeekb(selector,addr-physicaladdress);
// POKE: _farpokeb(selector,addr-physicaladdress,byt);
```

4.2. BIOS Access

DJGPP

```
#include <dpmi.h>
__dpmi_regs regs;
regs.x.ax=y; __dpmi_int(0x33,&regs); z=regs.h.al;
```

PASCAL

```
uses Dos;
var Regs:Registers;
Regs.AX:=x;
Intr(y);
z:=Regs.CL;
```

OTHER GCC

```
#include <dos.h>
_AX=x;
geninterrupt(y);
z=_CL;
```

4.3. PORTS Access

DJGPP

```
#include <pc.h>
```
(other gcc: conio.h)

```
outp(x,y); y=inp(x);
```

PASCAL

```
Port[x]:=y;
z:=Port[x];
```

4.4. Timer

4.4.1. Reprogramming the timer

value=1193180/T; T=frequeny of timer ticks
[p]43h=3Ch; [p]40h=Lo(value); [p]40h=Hi(value);

4.4.2. Microsectimer

DJGPP

```
 (global) unsigned long last,init_count;
 (global) unsigned long *c_ptr;
```

```
// (at initialisation):

 c_ptr = (unsigned long *)makeptr(0x40, 0x6c);
 disable();
 outportb(0x43, 0x34); outportb(0x40, 0); outportb(0x40, 0);
 init_count = *c_ptr;
 enable();

(routine):

 unsigned char m,l;
 unsigned int tt;
 unsigned long ct, ut;

 disable(); outportb(0x43, 0);
 l=(unsigned char)inportb(0x40); m=(unsigned char)inportb(0x40);
 ct=*c_ptr; enable();
 if (ct<init_count) ct+=last; else last=ct;
 ct-=init_count; tt=(unsigned)(-1)-((m<<8)|l); ut=ct*54925;

 return(ut+((long)tt*8381+ut%10000)/10000);
```

4.5. Interrupts

4.5.1. Overall

[p]20h=20h Interrupt has been handled.
[int]=0h DIV0
[int]=1h StepWork (if Trap Flag is set)
[int]=8h Timer
[int]=9h Keyboard
[int]=eh Disc
[int]=fh LPT
[int]=10h Screen
[int]=13h Discs
[int]=14h COMM
[int]=16h Keyboard ([ah]=0h Get char >[ah]=number, >[al]=ASCII; [ah]=1h Is any in buffer? >[zf]=0/1, >[ah]=number, >[al]=ASCII)
[int]=17h Printer
[int]=1ah Timer >[cx,dx] and >[al] if overloaded

4.5.2. Hooking the interrupts

DJGPP

```
<dpmi>,<go32>,<sys/nearptr>
// Hooking the interrupt
_go32_dpmi_seginfo oldvec,newvec;
_go32_dpmi_registers vecregs;
_go32_dpmi_get_real_mode_interrupt_vector(int_no, &oldvec);
newvec.pm_offset=(unsigned long)interrupt_handler;
_go32_dpmi_allocate_real_mode_callback_iret(&newvec,&vecregs);
_go32_dpmi_set_real_mode_interrupt_vector(int_no, &newvec);
// Dehooking the interrupt
_go32_dpmi_set_real_mode_interrupt_vector(int_no, &oldvec);
```

```
_go32_dpmi_free_real_mode_callback(&newvec);
// OR
_go32_dpmi_seginfo oldvec,newvec;
_go32_dpmi_get_protected_mode_interrupt_vector(int_no,&oldvec);
newvec.pm_offset=(int)interrupt_handler; newvec.pm_selector=_go32_my_cs();
_go32_dpmi_chain_protected_mode_interrupt_vector(int_no,&newvec);
// Dehooking
_go32_dpmi_set_protected_mode_interrupt_vector(int_no,&oldvec);
```

5. Miscelaneous

5.1. ASCII Codes

5.1.1. Borders

(single)					(double)				
218	196	194	196	191	201	205	203	205	187
179		179		179	186		186		186
195	196	197	196	180	204	205	206	205	185
179		179		179	186		186		186
192	196	193	196	217	200	205	202	205	188

shade 176, 177, 178, 219
half-lines 220(down), 221(left), 223(right), 223(up)

5.1.2. DOS Codes

F1-F10, F11, F12	59-68, 133, 134	up	72	down	80	left	75	right	77
+shift	84-93, 135, 136		56		50		52		54
+ctrl	94-103, 137, 138		160		164		115		116
+alt	104-113, 139, 140								

For Alt+A-Z: see Port scancodes

shift+tab 15	Alt+1-0 120-129	Alt+- 130
Alt+= 131	^left 115	^right 116
^End 117	^PrtSc 114	^Home 119
^PgDw 11	^PgUp 132	

home 71 (s)55	end 79 (s)49 (c)117	pgup 73 (s)57 (c)132
pgdw 81 (s)51 (c)118	ins 82 (s)48 (c)165	del 83 (s)46 (c)166

32 spc	37 %	42 *	47 /	61 =	91 [	96 `
33 !	38 &	43 +	[]	62 >	92 \	[]
34 "	39 '	44 ,	58 :	63 ?	93]	123 {
35 #	40 (	45 -	59 ;	64 @	94 ^	124 |
36 $	41)	46 .	60 <	[]	95 _	125 }

48-57 0-9 65-90 A-Z 97-122 a-z 126 ~

0 ^@ NUL: end of string
1 ^A SOH: start of heading (column move back) [happy white face]
2 ^B STX: start of text [happy black face]
3 ^C ETX: end of text (pgdw) [black heart]
4 ^D EOT: end of transmission (goright) [black diamond]
5 ^E ENQ: enquiry (goup) [black club]
6 ^F ACK: acknowledgement (column move forward) (confirm) [black spade]
7 ^G BEL: bell (del) [black dot]
8 ^H BS: backspace [dot cut of black square]
9 ^I HT: horizontal tabulation [circle]
10 ^J LF: line feed [circle cut of block square]
11 ^K VT: vertical tabulation [male]
12 ^L FF: form feed (new page) [female]
13 ^M CR: carriage return [note]
14 ^N SO: shift out (line split) [two notes]
15 ^O SI: shift in [circle with four outgrowths]
16 ^P DLE: data link escape (disconnect) [triangle right]
17 ^Q DC1: device control1 (X-ON) [triangle left]
18 ^R DC2: 2 (pgup) [arrows up-down]
19 ^S DC3: 3 (X-OFF) (goleft) [two exclamations]
20 ^T DC4: 4 (delline) [PI]
21 ^U NAK: negative aknowlegdement [paragraph]
22 ^V SYN: synchronous idle (ins/over) [underline]
23 ^W ETB: end of block [arrows up-down+underline]
24 ^X CAN: cancel (godown) [arrow up]
25 ^Y EM: end of media (del whole line) [arrow down]
26 ^Z SUB: substitute [arrow right]
27 ^[ESC: escape [arrow left]
28 ^\ FS: file separator [paragraph marker]
29 ^] GS: group separator [arrows left-right]
30 ^ RS: record separator [triangle up]
31 ^_ US: unit separator [triangle down]

5.1.3. Port scancodes

00h -	33h , <	4fh end gray1
01h Esc	34h . >	50h down gray2
02h-0Bh 1-0	35h ? / gray/	51h pgdw gray3
0Ch - _	36h Rshift	52h Ins gray0
0Dh + =	37h gray* Prt	53h Del gray.
0Eh Backspace	38h Alt	54h SysReq (Alt+Prt)
0Fh Tab	39h Space	57h F11
10h-19h Q-P	3ah CapsLock	58h F12
1ah [{	3bh-44h F1-F10	5bh LWin
1bh] }	45h NumLock	5ch RWin
1ch Enter gEnter	46h ScrollLock	5dh MenuWin
1dh Ctrl	47h Home gray7	5fh CallWin
1eh-26h A-L	48h up gray8	80h InsLocked
27h ; :	49h pgup gray9	81h CapsLocked
28h " '	4ah gray-	82h NumLocked
29h ~ `	4bh left gray4	83h ScrollLocked
2ah Lshift	4ch gray5	84h lCtrl
2bh \ |	4dh right gray6	85h lAlt
2ch-32h Z-M	4eh gray+	
Release +80h		

Esc 01	F1 3b	F2 3c	F3 3d	F4 3e		F5 3f	F6 40	F7 41	F8 42		F9 43	F10 44		F11 57	F12 58		Ps 54	Sl 46	Pa ?				
~ 29	1 02	2 03	3 04	4 05	5 06	6 07	7 08	8 09	9 0a	0 0b	- 0c	+ 0d	\ 2b		Bs 0e		In 52	Ho 47	Pu 49		Nl 45	/ 35	* 37	- 4a
Tab 0f		Q 10	W 11	E 12	R 13	T 14	Y 15	U 16	I 17	O 18	P 19	[1a		] 1b			De 53	En 4f	Pd 51		7 47	8 48	9 49	
Caps 3a		A 1e	S 1f	D 20	F 21	G 22	H 23	J 24	K 25	L 26	; 27	” 28	Enter 1c								4 4b	5 4c	6 4d	+ 4e
Shift 2a			Z 2c	X 2d	C 2e	V 2f	B 30	N 31	M 32	, 33	. 34	? 35	Shift 36					Up 48			1 4f	2 50	3 51	Ent
Ctrl 84	Win 5b	Alt 85	Space 39							Alt 38	Win 5c	Menu 5f	Ctrl 1d				Le 4b	Dw 50	Rt 4d		0 52		. 53	1c

5.2. Formats

5.2.1. Executables

COM: raw data, starts from 100h, one segment size

EXE:
+0h 2[b] "MZ"
+2h 2[b] Length of last page (skip)
+4h 2[b] Program+Header length (in pages=512b)
+6h 2[b] # of elements in move table
+8h 2[b] header length in paragraphs=16b
+ah 2[b] minimum memory over program (pars)
+ch 2[b] max memory over program (pars)
+eh 2[b] SS offset
+10h 2[b] SP value
+12h 2[b] CRC - negative sum of all bytes in file
+14h 2[b] IP value
+16h 2[b] CS offset
+18h 2[b] beginning of move table (01ch often)
+1ah 2[b] overlay number (0 often)
+1ch .. Other
[=18h] (4*[=6h]) movetable
zeroes to end of paragraph

SYS:
DEVICE=.. (CONFIG.SYS)
+0h 4[b] next addr (-1 if last or standalone)
+4h 2[b] attributes: [%]0: IsSTI (input console); [%]1: IsSTO (output console) [%]2: is a black hole (NUL); [%]3: is a clock (CLOCK); [%]6: 3.20+DOS [%]11: I/O operations; [%]15: char device
+6h 2[b] Requests address
+8h 2[b] Interrupts address
+ah 8[b] name

5.2.4. BMP

Bottom-up, Every line ends on dword boundary (if isn't - zeroes).

```
0000h 2[b]   ID='BM'
0002h [l]    filesize
0006h 4[b]   reserved
000Ah [l]    data offset
000Eh [l]    =40
0012h [l]    width
0016h [l]    height
001Ah [w]    planes#
001Ch [w]    Bits per pixel
001Eh [l]    0
0022h [l]    picture sizes (byt)
0026h [l]    x-res
002Ah [l]    y-res
002Eh [l]    used colors#
0032h [l]    important colors#
0036h        (1 << "BPP") colors def (BGRZ)
[000Ah]            Image data
```

5.2.5. PCX

```
Line - even number (zero if isn't)
0000h          1 byte   10
0001h          1 byte   Version 0/2/3(nopal)/4/5
0002h          1 byte   1 = RLE
0003h          1 byte   Bits per pixel
0004h          1 word    left margin of image
0006h          1 word    upper margin of image
0008h          1 word    right margin of image
000Ah          1 word    lower margin of image
000Ch          1 word    Horizontal DPI resolution
000Eh          1 word    Vertical DPI resolution
0010h         48 byte    Color palette setting (for 16c imgs - 16xRGB)
0040h          1 byte   reserved
0041h          1 byte   color planes="NCP"
0042h          1 word    bytes per line (even, =[rightma-leftma])="NBS"
0044h          1 word    1(color/bw), 2(gray)
0046h          1 word    Horizontal screen size
0048h          1 word    Vertical screen size
004Ah         54 byte    reserved, set to 0
header has 128 bytes, afterwards - RLE (192 is separator), then - palette:
0000h          1 byte   VGA palette ID (=0Ch)
0001h        768 byte    (RGB)
```

5.2.6. PIF

```
0000h          1 byte   reserved
0001h          1 byte   Checksum
0002h         30 char   Title for the window
0020h          1 word   Maximum memory reserved for program
0022h          1 word   Minimum memory reserved for program
```

0024h	63 char	Path and filename of the program
0063h	1 byte	0 - Do not close window on exit, other - Do
0064h	1 byte	Default drive (0=A: ??)
0065h	64 char	Default startup directory
00A5h	64 char	Parameters for program
00E5h	1 byte	Initial screen mode, 0 equals mode 3 ?
00E6h	1 byte	Text pages to reserve for program
00E7h	1 byte	First interrupt used by program
00E8h	1 byte	Last interrupt used by program
00E9h	1 byte	Rows on screen
00EAh	1 byte	Columns on screen
00EBh	1 byte	X position of window
00ECh	1 byte	Y position of window
00EDh	1 word	System memory ?? whatever
00EFh	64 char	?? Shared program path
012Fh	64 char	?? Shared program data file
016Fh	1 word	Program flags

5.2.7. WAV

0h (4b) "RIFF"
4h (4b) file length-8
8h (4b) "WAVE"
Ch (4b) "fmt "
10h (4b) 0x00000010
14h (2b) 0x0001 (PCM format)
16h (2b) #channels= 1(mono) 2(stereo)
18h (4b) sample rate, samples per second (44100)
1Ch (4b) sample rate * block align, bytes/second
20h (2b) block align, channels * bits/sample / 8
22h (2b) bits/sample (8 or 16)
24h (4b) "data"
28h (4b) length of data block
Must end on even byte (LOW/HIGH). 8bit 0..255, 16bit -32768..32767

5.3. Assembler

5.3.1. 8086 REGISTERS

AX,BX,CX,DX - 16bit, old=AH..DX, young=AL..DL
CS,DS,SS,ES - 16bit segm: CS=code, DS=data, SS=stack, ES=extra
DI,SI - memory indexing registers (D)estination (S)ource
SP - stack pointer
BP - memory indexing
Flags: S (char), Z (zero), O (overflow), I (ints), D (direction), P (parity)

5.3.2. 8086 MNEMONICS

MOV a,b - Move b to a, can be mov ah,byte ptr ds:[100], can't be mov cs,ds
PUSH a - put on stack (reg/16bit), ++SP; POP a - get from stack 16bit, --SP
ADD a,b - Add b to a, same lengths; SUB a,b - Substractions
INC a - ++a; DEC a - --a

MUL a - if 8bit: ax="a"*al, if 16bit dx:ax="a"*ax; DIV a - as MUL
ROL a,b - rotate a left b bits; ROR a,b - rotate a right b bits
NOT a - every bit change
NEG a - logic negation of last bit
AND a,b - put to a the result of a AND b; TEST a,b - no result, flags modified OR a,b - OR; XOR a,b - XOR
JMP a - jump. a can be label, (name:);
JE/JZ a - if Z=1;
JNE/JNZ a - if Z=0;
JS a - if S=1; JNS a - if S=0;
JC a - if C=1; JNC a - if C=0;
JO a - if O=1; JNO a - if O=0;
LOOP a - if cx<>0- If jumped, --cx;
CALL a - can return (RET)- call cs:[bx];
RET – return
INT a - call a interrupt
NOP - nothing
IN AL/AX,DX/a - get AL/AX from DX/a
OUT DX/a,AL/AX - send AL/AX from DX/a
CLC - C=0; STC - C=1; CLD - D=1; STD - D=1; CLI - I=0; STI - I=1
MOVSB - copy 1b from DS:[SI] to ES:[DI] , inc or dec SI, DI (depends on D)
CMP a,b - as SUB, no result, only flags

5.4. Other

DJGPP

```
/*Random*/   if(firstrandom) {firstrandom=0; srandom((gettimer()>>1)); }
 return ( (int) ((random()/(float)RAND_MAX)*(float)seedmax ) );

/*Ignore CTRL+C*/   __djgpp_set_ctrl_c(0);
```