```
!
!	setup.S		Copyright (C) 1991, 1992 Linus Torvalds
!
! setup.s is responsible for getting the system data from the BIOS,
! and putting them into the appropriate places in system memory.
! both setup.s and system has been loaded by the bootblock.
!
! This code asks the bios for memory/disk/other parameters, and
! puts them in a "safe" place: 0x90000-0x901FF, ie where the
! boot-block used to be. It is then up to the protected mode
! system to read them from there before the area is overwritten
! for buffer-blocks.
!
! Move PS/2 aux init code to psaux.c
! (troyer@saifr00.cfsat.Honeywell.COM) 03Oct92
!
! some changes and additional features by Christoph Niemann, March 1993
! (niemann@rubdv15.ETDV.Ruhr-Uni-Bochum.De)
!

! NOTE! These had better be the same as in bootsect.s!
#include <linux/config.h>
#include <linux/segment.h>

#ifndef SVGA_MODE
#define SVGA_MODE ASK_VGA
#endif

INITSEG  = DEF_INITSEG! we move boot here - out of the way
SYSSEG   = DEF_SYSSEG ! system loaded at 0x10000 (65536).
SETUPSEG = DEF_SETUPSEG	! this is the current segment

.globl begtext, begdata, begbss, endtext, enddata, endbss
.text
begtext:
.data
begdata:
.bss
begbss:
.text

entry start
start:

! ok, the read went well so we get current cursor position and save it for
! posterity.

	mov	ax,#INITSEG	! this is done in bootsect already, but...
	mov	ds,ax

! Get memory size (extended mem, kB)

	mov	ah,#0x88
	int	0x15
	mov	[2],ax

! set the keyboard repeat rate to the max

	mov	ax,#0x0305
	xor	bx,bx		! clear bx
	int	0x16

! check for EGA/VGA and some config parameters

	mov	ah,#0x12
	mov	bl,#0x10
	int	0x10
	mov	[8],ax
	mov	[10],bx
	mov	[12],cx
	mov	ax,#0x5019
	cmp	bl,#0x10
	je	novga
	mov	ax,#0x1a00	! Added check for EGA/VGA discrimination
	int	0x10
	mov	bx,ax
	mov	ax,#0x5019
	cmp	bl,#0x1a	! 1a means VGA, anything else EGA or lower
	jne	novga
	call	chsvga
```

```
novga:  mov     [14],ax
        mov     ah,#0x03        ! read cursor pos
        xor     bh,bh           ! clear bh
        int     0x10            ! save it in known place, con_init fetches
        mov     [0],dx          ! it from 0x90000.

! Get video-card data:

        mov     ah,#0x0f
        int     0x10
        mov     [4],bx          ! bh = display page
        mov     [6],ax          ! al = video mode, ah = window width

! Get hd0 data

        xor     ax,ax           ! clear ax
        mov     ds,ax
        lds     si,[4*0x41]
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0080
        mov     cx,#0x10
        cld
        rep
        movsb

! Get hd1 data

        xor     ax,ax           ! clear ax
        mov     ds,ax
        lds     si,[4*0x46]
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0090
        mov     cx,#0x10
        cld
        rep
        movsb

! Check that there IS a hd1 :-)

        mov     ax,#0x01500
        mov     dl,#0x81
        int     0x13
        jc      no_disk1
        cmp     ah,#3
        je      is_disk1
no_disk1:
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0090
        mov     cx,#0x10
        xor     ax,ax           ! clear ax
        cld
        rep
        stosb
is_disk1:

! check for PS/2 pointing device

        mov     ax,#INITSEG
        mov     ds,ax
        mov     [0x1ff],#0      ! default is no pointing device
        int     0x11            ! int 0x11: equipment determination
        test    al,#0x04        ! check if pointing device installed
        jz      no_psmouse
        mov     [0x1ff],#0xaa   ! device present
no_psmouse:
! now we want to move to protected mode ...

        cli                     ! no interrupts allowed !
        mov     al,#0x80        ! disable NMI for the bootup sequence
        out     #0x70,al

! first we move the system to its rightful place

        mov     ax,#0x100       ! start of destination segment
        mov     bx,#0x1000      ! start of source segment
        cld                     ! 'direction'=0, movs moves forward
```

```
do_move:
        mov     es,ax           ! destination segment
        add     ax,#0x100
        cmp     ax,#0x9000
        jz      end_move
        mov     ds,bx           ! source segment
        add     bx,#0x100
        sub     di,di
        sub     si,si
        mov     cx,#0x800
        rep
        movsw
        jmp     do_move

! then we load the segment descriptors

end_move:
        mov     ax,#SETUPSEG    ! right, forgot this at first. didn't work :-)
        mov     ds,ax
        lidt    idt_48          ! load idt with 0,0
        lgdt    gdt_48          ! load gdt with whatever appropriate

! that was painless, now we enable A20

        call    empty_8042
        mov     al,#0xD1                ! command write
        out     #0x64,al
        call    empty_8042
        mov     al,#0xDF                ! A20 on
        out     #0x60,al
        call    empty_8042

! make sure any possible coprocessor is properly reset..

        xor     ax,ax
        out     #0xf0,al
        call    delay
        out     #0xf1,al
        call    delay

! well, that went ok, I hope. Now we have to reprogram the interrupts :-(
! we put them right after the intel-reserved hardware interrupts, at
! int 0x20-0x2F. There they won't mess up anything. Sadly IBM really
! messed this up with the original PC, and they haven't been able to
! rectify it afterwards. Thus the bios puts interrupts at 0x08-0x0f,
! which is used for the internal hardware interrupts as well. We just
! have to reprogram the 8259's, and it isn't fun.

        mov     al,#0x11                ! initialization sequence
        out     #0x20,al                ! send it to 8259A-1
        call    delay
        out     #0xA0,al                ! and to 8259A-2
        call    delay
        mov     al,#0x20                ! start of hardware int's (0x20)
        out     #0x21,al
        call    delay
        mov     al,#0x28                ! start of hardware int's 2 (0x28)
        out     #0xA1,al
        call    delay
        mov     al,#0x04                ! 8259-1 is master
        out     #0x21,al
        call    delay
        mov     al,#0x02                ! 8259-2 is slave
        out     #0xA1,al
        call    delay
        mov     al,#0x01                ! 8086 mode for both
        out     #0x21,al
        call    delay
        out     #0xA1,al
        call    delay
        mov     al,#0xFF                ! mask off all interrupts for now
        out     #0xA1,al
        call    delay
        mov     al,#0xFB                ! mask all irq's but irq2 which
        out     #0x21,al                ! is cascaded

! well, that certainly wasn't fun :-(. Hopefully it works, and we don't
! need no steenking BIOS anyway (except for the initial loading :-).
! The BIOS-routine wants lots of unnecessary data, and it's less
! "interesting" anyway. This is how REAL programmers do it.
```

```
!
! Well, now's the time to actually move into protected mode. To make
! things as simple as possible, we do no register set-up or anything,
! we let the gnu-compiled 32-bit programs do that. We just jump to
! absolute address 0x00000, in 32-bit protected mode.
!
! Note that the short jump isn't strictly needed, althought there are
! reasons why it might be a good idea. It won't hurt in any case.
!
       mov     ax,#0x0001     ! protected mode (PE) bit
       lmsw    ax             ! This is it!
       jmp     flush_instr
flush_instr:
       jmpi    0x1000,KERNEL_CS      ! jmp offset 1000 of segment 0x10 (cs)

! This routine checks that the keyboard command queue is empty
! (after emptying the output buffers)
!
! No timeout is used - if this hangs there is something wrong with
! the machine, and we probably couldn't proceed anyway.
empty_8042:
       call    delay
       in      al,#0x64       ! 8042 status port
       test    al,#1          ! output buffer?
       jz      no_output
       call    delay
       in      al,#0x60       ! read it
       jmp     empty_8042
no_output:
       test    al,#2          ! is input buffer full?
       jnz     empty_8042     ! yes - loop
       ret
!
! Read a key and return the (US-)ascii code in al, scan code in ah
!
getkey:
       xor     ah,ah
       int     0x16
       ret

!
! Read a key with a timeout of 30 seconds. The cmos clock is used to get
! the time.
!
getkt:
       call    gettime
       add     al,#30         ! wait 30 seconds
       cmp     al,#60
       jl      lminute
       sub     al,#60
lminute:
       mov     cl,al
again: mov     ah,#0x01
       int     0x16
       jnz     getkey         ! key pressed, so get it
       call    gettime
       cmp     al,cl
       jne     again
       mov     al,#0x20       ! timeout, return default char `space'
       ret

!
! Flush the keyboard buffer
!
flush: mov     ah,#0x01
       int     0x16
       jz      empty
       xor     ah,ah
       int     0x16
       jmp     flush
empty: ret
```

```
!
! Read the cmos clock. Return the seconds in al
!
gettime:
        push    cx
        mov     ah,#0x02
        int     0x1a
        mov     al,dh                   ! dh contains the seconds
        and     al,#0x0f
        mov     ah,dh
        mov     cl,#0x04
        shr     ah,cl
        aad
        pop     cx
        ret

!
! Delay is needed after doing i/o
!
delay:
        .word   0x00eb                  ! jmp $+2
        ret

! Routine trying to recognize type of SVGA-board present (if any)
! and if it recognize one gives the choices of resolution it offers.
! If one is found the resolution chosen is given by al,ah (rows,cols).

chsvga: cld
        push    ds
        push    cs
        mov     ax,[0x01fa]
        pop     ds
        mov     modesave,ax
        mov     ax,#0xc000
        mov     es,ax
        mov     ax,modesave
        cmp     ax,#NORMAL_VGA
        je      defvga
        cmp     ax,#EXTENDED_VGA
        je      vga50
        cmp     ax,#ASK_VGA
        jne     svga
        lea     si,msg1
        call    prtstr
        call    flush
nokey:  call    getkt
        cmp     al,#0x0d                ! enter ?
        je      svga                    ! yes - svga selection
        cmp     al,#0x20                ! space ?
        je      defvga                  ! no - repeat
        call    beep
        jmp     nokey
defvga: mov     ax,#0x5019
        pop     ds
        ret
/* extended vga mode: 80x50 */
vga50:
        mov     ax,#0x1112
        xor     bl,bl
        int     0x10            ! use 8x8 font set (50 lines on VGA)
        mov     ax,#0x1200
        mov     bl,#0x20
        int     0x10            ! use alternate print screen
        mov     ax,#0x1201
        mov     bl,#0x34
        int     0x10            ! turn off cursor emulation
        mov     ah,#0x01
        mov     cx,#0x0607
        int     0x10            ! turn on cursor (scan lines 6 to 7)
        pop     ds
        mov     ax,#0x5032      ! return 80x50
        ret
```

```
/* extended vga mode: 80x28 */
vga28:
        pop     ax              ! clean the stack
        mov     ax,#0x1111
        xor     bl,bl
        int     0x10            ! use 9x14 fontset (28 lines on VGA)
        mov     ah, #0x01
        mov     cx,#0x0b0c
        int     0x10            ! turn on cursor (scan lines 11 to 12)
        pop     ds
        mov     ax,#0x501c      ! return 80x28
        ret
/* svga modes */
svga:   cld
        lea     si,id9GXE       ! Check for the #9GXE (jyanowit@orixa.mtholyoke.edu,thanks
dlm40629@uxa.cso.uiuc.edu)
        mov     di,#0x49        ! id string is at c000:049
        mov     cx,#0x11        ! length of "Graphics Power By"
        repe
        cmpsb
        jne     of1280
is9GXE: lea     si,dsc9GXE      ! table of descriptions of video modes for BIOS
        lea     di,mo9GXE       ! table of sizes of video modes for my BIOS
        br      selmod          ! go ask for video mode
of1280: cld
        lea     si,idf1280      ! Check for Orchid F1280 (dingbat@diku.dk)
        mov     di,#0x10a       ! id string is at c000:010a
        mov     cx,#0x21        ! length
        repe
        cmpsb
        jne     nf1280
isVRAM: lea     si,dscf1280
        lea     di,mof1280
        br      selmod
nf1280: lea     si,idVRAM
        mov     di,#0x10a
        mov     cx,#0x0c
        repe
        cmpsb
        je      isVRAM
        cld
        lea     si,idati                ! Check ATI 'clues'
        mov     di,#0x31
        mov     cx,#0x09
        repe
        cmpsb
        jne     noati
        lea     si,dscati
        lea     di,moati
        br      selmod
noati:  mov     ax,#0x200f              ! Check Ahead 'clues'
        mov     dx,#0x3ce
        out     dx,ax
        inc     dx
        in      al,dx
        cmp     al,#0x20
        je      isahed
        cmp     al,#0x21
        jne     noahed
isahed: lea     si,dscahead
        lea     di,moahead
        br      selmod
noahed: mov     dx,#0x3c3               ! Check Chips & Tech. 'clues'
        in      al,dx
        or      al,#0x10
        out     dx,al
        mov     dx,#0x104
        in      al,dx
        mov     bl,al
        mov     dx,#0x3c3
        in      al,dx
        and     al,#0xef
        out     dx,al
        cmp     bl,[idcandt]
        jne     nocant
        lea     si,dsccandt
        lea     di,mocandt
        br      selmod
```

```
nocant: mov     dx,#0x3d4               ! Check Cirrus 'clues'
        mov     al,#0x0c
        out     dx,al
        inc     dx
        in      al,dx
        mov     bl,al
        xor     al,al
        out     dx,al
        dec     dx
        mov     al,#0x1f
        out     dx,al
        inc     dx
        in      al,dx
        mov     bh,al
        xor     ah,ah
        shl     al,#4
        mov     cx,ax
        mov     al,bh
        shr     al,#4
        add     cx,ax
        shl     cx,#8
        add     cx,#6
        mov     ax,cx
        mov     dx,#0x3c4
        out     dx,ax
        inc     dx
        in      al,dx
        and     al,al
        jnz     nocirr
        mov     al,bh
        out     dx,al
        in      al,dx
        cmp     al,#0x01
        jne     nocirr
        call    rst3d4
        lea     si,dsccirrus
        lea     di,mocirrus
        br      selmod
rst3d4: mov     dx,#0x3d4
        mov     al,bl
        xor     ah,ah
        shl     ax,#8
        add     ax,#0x0c
        out     dx,ax
        ret
nocirr: call    rst3d4                  ! Check Everex 'clues'
        mov     ax,#0x7000
        xor     bx,bx
        int     0x10
        cmp     al,#0x70
        jne     noevrx
        shr     dx,#4
        cmp     dx,#0x678
        je      istrid
        cmp     dx,#0x236
        je      istrid
        lea     si,dsceverex
        lea     di,moeverex
        br      selmod
istrid: lea     cx,ev2tri
        jmp     cx
noevrx: lea     si,idgenoa              ! Check Genoa 'clues'
        xor     ax,ax
        seg es
        mov     al,[0x37]
        mov     di,ax
        mov     cx,#0x04
        dec     si
        dec     di
l1:     inc     si
        inc     di
        mov     al,(si)
        test    al,al
        jz      l2
        seg es
        cmp     al,(di)
```

```
l2:     loope   l1
        cmp     cx,#0x00
        jne     nogen
        lea     si,dscgenoa
        lea     di,mogenoa
        br      selmod
nogen:  cld
        lea     si,idoakvga
        mov     di,#0x08
        mov     cx,#0x08
        repe
        cmpsb
        jne     nooak
        lea     si,dscoakvga
        lea     di,mooakvga
        br      selmod
nooak:  cld
        lea     si,idparadise           ! Check Paradise 'clues'
        mov     di,#0x7d
        mov     cx,#0x04
        repe
        cmpsb
        jne     nopara
        lea     si,dscparadise
        lea     di,moparadise
        br      selmod
nopara: mov     dx,#0x3c4               ! Check Trident 'clues'
        mov     al,#0x0e
        out     dx,al
        inc     dx
        in      al,dx
        xchg    ah,al
        xor     al,al
        out     dx,al
        in      al,dx
        xchg    al,ah
        mov     bl,al           ! Strange thing ... in the book this wasn't
        and     bl,#0x02        ! necessary but it worked on my card which
        jz      setb2           ! is a trident. Without it the screen goes
        and     al,#0xfd        ! blurred ...
        jmp     clrb2           !
setb2:  or      al,#0x02        !
clrb2:  out     dx,al
        and     ah,#0x0f
        cmp     ah,#0x02
        jne     notrid
ev2tri: lea     si,dsctrident
        lea     di,motrident
        jmp     selmod
notrid: mov     dx,#0x3cd               ! Check Tseng 'clues'
        in      al,dx                   ! Could things be this simple ! :-)
        mov     bl,al
        mov     al,#0x55
        out     dx,al
        in      al,dx
        mov     ah,al
        mov     al,bl
        out     dx,al
        cmp     ah,#0x55
        jne     notsen
        lea     si,dsctseng
        lea     di,motseng
        jmp     selmod
notsen: mov     dx,#0x3cc               ! Check Video7 'clues'
        in      al,dx
        mov     dx,#0x3b4
        and     al,#0x01
        jz      even7
        mov     dx,#0x3d4
```

```
even7:  mov     al,#0x0c
        out     dx,al
        inc     dx
        in      al,dx
        mov     bl,al
        mov     al,#0x55
        out     dx,al
        in      al,dx
        dec     dx
        mov     al,#0x1f
        out     dx,al
        inc     dx
        in      al,dx
        mov     bh,al
        dec     dx
        mov     al,#0x0c
        out     dx,al
        inc     dx
        mov     al,bl
        out     dx,al
        mov     al,#0x55
        xor     al,#0xea
        cmp     al,bh
        jne     novid7
        lea     si,dscvideo7
        lea     di,movideo7
        jmp     selmod
novid7: lea     si,dsunknown
        lea     di,mounknown
selmod: xor     cx,cx
        mov     cl,(di)
        mov     ax,modesave
        cmp     ax,#ASK_VGA
        je      askmod
        cmp     ax,#NORMAL_VGA
        je      askmod
        cmp     al,cl
        jl      gotmode
        push    si
        lea     si,msg4
        call    prtstr
        pop     si
askmod: push    si
        lea     si,msg2
        call    prtstr
        pop     si
        push    si
        push    cx
tbl:    pop     bx
        push    bx
        mov     al,bl
        sub     al,cl
        call    modepr
        lodsw
        xchg    al,ah
        call    dprnt
        xchg    ah,al
        push    ax
        mov     al,#0x78
        call    prnt1
        pop     ax
        call    dprnt
        push    si
        lea     si,crlf         ! print CR+LF
        call    prtstr
        pop     si
        loop    tbl
        pop     cx
        lea     si,msg3
        call    prtstr
        pop     si
        add     cl,#0x30
        jmp     nonum
```

```
nonumb: call    beep
nonum:  call    getkey
        cmp     al,#0x30        ! ascii `0'
        jb      nonumb
        cmp     al,#0x3a        ! ascii `9'
        jbe     number
        cmp     al,#0x61        ! ascii `a'
        jb      nonumb
        cmp     al,#0x7a        ! ascii `z'
        ja      nonumb
        sub     al,#0x27
        cmp     al,cl
        jae     nonumb
        sub     al,#0x30
        jmp     gotmode
number:  cmp    al,cl
        jae     nonumb
        sub     al,#0x30
gotmode:        xor     ah,ah
        or      al,al
        beq     vga50
        push    ax
        dec     ax
        beq     vga28
        add     di,ax
        mov     al,(di)
        int     0x10
        pop     ax
        shl     ax,#1
        add     si,ax
        lodsw
        pop     ds
        ret

! Routine to print asciiz-string at DS:SI

prtstr: lodsb
        and     al,al
        jz      fin
        call    prnt1
        jmp     prtstr
fin:    ret

! Routine to print a decimal value on screen, the value to be
! printed is put in al (i.e 0-255).

dprnt:  push    ax
        push    cx
        xor     ah,ah           ! Clear ah
        mov     cl,#0x0a
        idiv    cl
        cmp     al,#0x09
        jbe     lt100
        call    dprnt
        jmp     skip10
lt100:  add     al,#0x30
        call    prnt1
skip10: mov     al,ah
        add     al,#0x30
        call    prnt1
        pop     cx
        pop     ax
        ret
```

```
!
! Routine to print the mode number key on screen. Mode numbers
! 0-9 print the ascii values `0' to '9', 10-35 are represented by
! the letters `a' to `z'. This routine prints some spaces around the
! mode no.
!

modepr: push    ax
        cmp     al,#0x0a
        jb      digit           ! Here is no check for number > 35
        add     al,#0x27
digit:  add     al,#0x30
        mov     modenr, al
        push    si
        lea     si, modestring
        call    prtstr
        pop     si
        pop     ax
        ret

! Part of above routine, this one just prints ascii al

prnt1:  push    ax
        push    cx
        xor     bh,bh
        mov     cx,#0x01
        mov     ah,#0x0e
        int     0x10
        pop     cx
        pop     ax
        ret

beep:   mov     al,#0x07
        jmp     prnt1

gdt:
        .word   0,0,0,0         ! dummy

        .word   0,0,0,0         ! unused

        .word   0x07FF          ! 8Mb - limit=2047 (2048*4096=8Mb)
        .word   0x0000          ! base address=0
        .word   0x9A00          ! code read/exec
        .word   0x00C0          ! granularity=4096, 386

        .word   0x07FF          ! 8Mb - limit=2047 (2048*4096=8Mb)
        .word   0x0000          ! base address=0
        .word   0x9200          ! data read/write
        .word   0x00C0          ! granularity=4096, 386

idt_48:
        .word   0                       ! idt limit=0
        .word   0,0                     ! idt base=0L

gdt_48:
        .word   0x800           ! gdt limit=2048, 256 GDT entries
        .word   512+gdt,0x9     ! gdt base = 0X9xxxx

msg1:           .ascii  "Press <RETURN> to see SVGA-modes available, <SPACE> to continue
or wait 30 secs."
                db      0x0d, 0x0a, 0x0a, 0x00
msg2:           .ascii  "Mode:  COLSxROWS:"
                db      0x0d, 0x0a, 0x0a, 0x00
msg3:           db      0x0d, 0x0a
                .ascii  "Choose mode by pressing the corresponding number or letter."
crlf:           db      0x0d, 0x0a, 0x00
msg4:           .ascii  "You passed an undefined mode number to setup. Please choose a new
mode."
                db      0x0d, 0x0a, 0x0a, 0x07, 0x00
modestring:     .ascii  "   "
modenr:         db      0x00    ! mode number
                .ascii  ":    "
                db      0x00
```

```
idati:          .ascii  "761295520"
idcandt:        .byte   0xa5
idgenoa:        .byte   0x77, 0x00, 0x99, 0x66
idparadise:     .ascii  "VGA="
idoakvga:       .ascii   "OAK VGA "
idf1280:        .ascii  "Orchid Technology Fahrenheit 1280"
id9GXE:         .ascii   "Graphics Power By"
idVRAM:         .ascii  "Stealth VRAM"

! Manufacturer:         Numofmodes+2:       Mode:
! Number of modes is the number of chip-specific svga modes plus the extended
! modes available on any vga (currently 2)

moati:          .byte   0x04,   0x23, 0x33
moahead:        .byte   0x07,   0x22, 0x23, 0x24, 0x2f, 0x34
mocandt:        .byte   0x04,   0x60, 0x61
mocirrus:       .byte   0x06,   0x1f, 0x20, 0x22, 0x31
moeverex:       .byte   0x0c,   0x03, 0x04, 0x07, 0x08, 0x0a, 0x0b, 0x16, 0x18, 0x21, 0x40
mogenoa:        .byte   0x0c,   0x58, 0x5a, 0x60, 0x61, 0x62, 0x63, 0x64, 0x72, 0x74, 0x78
moparadise:     .byte   0x04,   0x55, 0x54
motrident:      .byte   0x09,   0x50, 0x51, 0x52, 0x57, 0x58, 0x59, 0x5a
motseng:        .byte   0x07,   0x26, 0x2a, 0x23, 0x24, 0x22
movideo7:       .byte   0x08,   0x40, 0x43, 0x44, 0x41, 0x42, 0x45
mooakvga:       .byte    0x08,   0x00, 0x07, 0x4e, 0x4f, 0x50, 0x51
mo9GXE:         .byte   0x04,   0x54, 0x55
mof1280:        .byte   0x04,   0x54, 0x55
mounknown:      .byte   0x02

!                       msb = Cols lsb = Rows:
! The first two modes are standard vga modes available on any vga.
! mode 0 is 80x50 and mode 1 is 80x28

dscati:         .word   0x5032, 0x501c, 0x8419, 0x842c
dscahead:       .word   0x5032, 0x501c, 0x842c, 0x8419, 0x841c, 0xa032, 0x5042
dsccandt:       .word   0x5032, 0x501c, 0x8419, 0x8432
dsccirrus:      .word   0x5032, 0x501c, 0x8419, 0x842c, 0x841e, 0x6425
dsceverex:      .word   0x5032, 0x501c, 0x5022, 0x503c, 0x642b, 0x644b, 0x8419, 0x842c,
0x501e, 0x641b, 0xa040, 0x841e
dscgenoa:       .word   0x5032, 0x501c, 0x5020, 0x642a, 0x8419, 0x841d, 0x8420, 0x842c,
0x843c, 0x503c, 0x5042, 0x644b
dscparadise:    .word   0x5032, 0x501c, 0x8419, 0x842b
dsctrident:     .word   0x5032, 0x501c, 0x501e, 0x502b, 0x503c, 0x8419, 0x841e, 0x842b,
0x843c
dsctseng:       .word   0x5032, 0x501c, 0x503c, 0x6428, 0x8419, 0x841c, 0x842c
dscvideo7:      .word   0x5032, 0x501c, 0x502b, 0x503c, 0x643c, 0x8419, 0x842c, 0x841c
dscoakvga:      .word    0x5032, 0x501c, 0x2819, 0x5019, 0x503c, 0x843c, 0x8419, 0x842b
dscf1280:       .word   0x5032, 0x501c, 0x842b, 0x8419
dsc9GXE:        .word   0x5032, 0x501c, 0x842b, 0x8419
dsunknown:      .word   0x5032, 0x501c
modesave:       .word   SVGA_MODE


.text
endtext:
.data
enddata:
.bss
endbss:
```