

```
; - - - - -  
-  
;  
; Description: Heroic attempt to disassemble the Windows 95 Boot Sector.  
; Date: 16. Aug. 1998  
; Author: Mr. X  
; Email: unknown  
; Comment: This boot code is messy.  
; Status: PART I, II and III are now pretty much done.  
; Important: "SHLD EDX,EAX,16" This is a Microsoft Patent.  
; Also take a look at the "list near EOF"  
;  
; ---> CUT HERE IF YOU LIKE TO LISTEN TO ME <---  
;  
; This file will assemble into a fully functional (I hope) Win95B boot code.  
;  
; tasm win95 /m  
; tlink win95,win95.bin /t  
;  
; Ask someone for the proper dropper code...  
; - - - - -  
-  
;  
; AFTER DISASSEMBLY I have this impression:  
;  
; This is what Bill Gates said... when Win95 was going to be released:  
;  
; "Gates: OK, then we're ready to press the CD's and start shipping out  
; this new load of shit, but of course nobody will notice... harr harr.  
; Employee: Hey, Mr. Gates what about the Boot Sector?? We haven't  
; written the FAT32 support routines there yet...  
; Gates: Ah, that's right... anybody?? We have just 45 minutes...  
; Employee #2: Well, I think I can hack some shit together...  
; Gates: Fine, go for it... remember you have only 44 minutes...  
; Employee #2: I think I can do it.  
; Gates: Fine, then I'll just go home... We've made it!!"  
;  
; FUNNY?  
;  
; There is some really strange code in this boot record....  
;  
; I bet Bill Gates hired some crazy mother fucker to write this shit.  
; It seems like he had really tried to make the code fit within one sector.  
; But when it didn't hid just decided to use three instead...  
;
```

```
; This is a typical microsoft solution, they mix stupid 8086 code... with
; cheap solutions and then suddenly they use 386 code...
;
; And then there is the new FAT32 data structures where they have moved
; the volume label, FileSystem ID and serial number down to make room for
; some extended FAT32 variables... it sucks. Why not rearrange the whole
; structure... An OS would not try to interpret the shit anyway, because
; the Partitioni Table SYSID has changed with FAT32.
;
; As I said... crazy mother fucker...
;
; Well, well... here's some of the stuff... with a mix of mine and sourcer's
; comments...
;
; Another thing about TASM, which I use, of course I didn't buy it... I'm
; have a shareware version ;) on a 30 year trial period.
;
; Back to what I was about to say again... When I use the brXXXX variables
; in indexing with BP, TASM generates 16-bit offset in operands even when
; they are less than 128... the Win95 code uses byte offsets (I'm not sure
; if I'm expressing myself clear here). When I changed the code from:
;
; mov AX,[bp+brHPC] to mov AX,[bp+128], TASM did use the byte offset form...
; This made my code a little less readable... but the comments should give
; you an idea of what data is being accessed.
;
; Basically this boot sector code is 32 bit extension for a 16 bit patch to
; an 8 bit boot sector originally coded for a 4 bit microprocessor, written
; by a 2 bit company, that can't stand 1 bit of competition.
;
; ---> CUT HERE IF YOU DOES NOT LIKE TO LISTEN TO ME <---
```

.386C

```
CODE                SEGMENT USE16
```

```
                    ASSUME  CS:CODE, DS:CODE, SS:NOHING
```

```
; BOOT RECORD - PART I - MAIN BOOT SECTOR CODE
```

```
; Just so I've said it - ASM opcodes are only readable when capitalized,
; but I forgot to set the option in sourcer... so I wrote a small TP program
; that capitalized everything but what's after the semicolon...
```

```
Win95b              PROC    FAR
```

```
                    JMP     SkipData          ; 0000h
```

```
brINT13Flag      DB      90H          ; 0002h - 0EH for INT13 AH=42 READ
brOEM            DB      'MSWIN4.1'   ; 0003h - OEM ID - Windows 95B
brBPS            DW      512          ; 000Bh - Bytes per sector
brSPC            DB      8            ; 000Dh - Sector per cluster
brResCount       DW      32          ; 000Eh - Reserved sectors
brFATs           DB      2            ; 0010h - FAT copies
brRootEntries    DW      0            ; 0011h - Root directory entries
brSectorCount    DW      0            ; 0013h - Sectors in volume, < 32MB
brMedia          DB      0F8H        ; 0015h - Media descriptor
brSPF            DW      0            ; 0016h - Sectors per FAT
brSPH            DW      63          ; 0018h - Sectors per head/track
brHPC            DW      128         ; 001Ah - Heads per cylinder
brHidden         DD      63          ; 001Ch - Hidden sectors
brSectors        DD      6305985     ; 0020h - Total number of sectors
brSPF32          DD      6153        ; 0024h - Sector per FAT (FAT32)
brFlags          DW      0            ; 0028h - Flags (FAT32)
brVersion        DW      0            ; 002Ah - FS Version (FAT32)
brRootCluster    DD      2            ; 002Ch - Root start cluster (FAT32)
brFSInfoSector   DW      1            ; 0030h - FS Info Sector (FAT32)
brBackupBoot     DW      6            ; 0032h - Backup Boot Record
brReserved       DB      6 DUP (0)    ; 0038h - Reserved
brShitter        DB      6 DUP (0)    ; 003Bh - Unused filler??
brDrive          DB      80H         ; 0040h - BIOS drive number
brHeadTemp       DB      00H         ; 0041h - Head/temp number????
brSignature      DB      29H         ; 0042h - Extended Boot Record sig.
brSerialNum      DD      404418EAH   ; 0043h - Volume serial number
brLabel          DB      'HARDDISK   ' ; 0047h - Volume label
brFSID           DB      'FAT32     ' ; 0052h - File System ID
```

SkipData:

```
      CLI
      XOR      CX,CX
      MOV      SS,CX          ; SS=CX=0
```

```
; Set up stack 8 bytes below us, do you know why???
; Yes, it's because somewhere in this code, the shitter who
; wrote this... save the start of data area DWORD at 7C00H - 4 and
; the value -1 at 7C00H - 8... cool?
```

```
      MOV      SP,7C00H - 8
      MOV      ES,CX          ; ES=CX=0
      MOV      BP,78H
```

```
; Point DS:SI to INT 1E - DISKETTE PARAMS structure...
```

```
LDS      SI,DWORD PTR SS:[BP]
```

```
PUSH    DS
```

```
PUSH    SI
```

```
PUSH    SS
```

```
PUSH    BP
```

```
MOV     DI,522H
```

```
MOV     SS:[BP],DI          ; setup our INT 1E
```

```
MOV     SS:[BP+2],CX
```

```
; copy 11 bytes from old diskette parameter table into  
; es:522h, that is 0000:0522 or 0050:0022 - into the PrtScr/DOS
```

area.

```
; I assume that 0001-0021 is used for something else 0050:0000 I
```

know

```
; is the PrtScr flag byte.
```

```
MOV     CL,11
```

```
CLD
```

```
REP     MOVSB
```

```
MOV     DS,CX              ; DS=CX=0
```

```
MOV     BP,7C00H          ; point BP to start of us
```

```
MOV     BYTE PTR [DI-2],0FH ; modify head settle time
```

```
MOV     AX,SS:[BP+18H]
```

```
MOV     [DI-7],AL         ; modify sectors per track
```

```
; compare drive number with 0...
```

```
; if greater than or equal... go to MBRReadError
```

```
; I guess that lower than zero... must be -1 .. -128 (80H..FFH)
```

```
; Which would mean Harddisk boot...
```

```
CMP     SS:[BP+40H],CL    ; Boot from diskette?
```

```
JGE     MBRReadError
```

```
MOV     AX,CX             ; AX=CX=0
```

```
CWD     ; DX=AX[15]-> (Zero)
```

```
MOV     BX,0700H         ; Use 0000:0700 for sector
```

```
; read buffer
```

```
CALL    ReadSector      ; load Master Boot Record
JC      MBRReadError    ; error?

SUB     BX,58            ; BX = 08C6h (700h - 3Ah)
                        ; point to "start" field

MOV     EAX,DS:[7C1CH]   ; load hidden sectors
CheckMBR:
CMP     EAX,[BX]        ; Is this our entry??

MOV     DL,[BX-4]       ; Put System ID in DL

JNZ     NotOurs         ; Jump if not our entry

; If system ID or "partition type", is 0Ch or 0Eh, ReadSector
; will use INT13/42H...

OR      DL,2            ; set bit 1, to allow for
                        ; types 0Ch or 0Eh to be
                        ; thought of as both 0Eh.

MOV     SS:[BP+2],DL    ; set brINT13Flag
NotOurs:
ADD     BL,16           ; skip to next entry...
JNB     CheckMBR        ; More entries?
MBRReadError:
MOV     DI,2

; FAT32 - Is sector per FAT zero?

CMP     WORD PTR SS:[BP+16H],0
JNE     ShowErrMsg1

; Put number of hidden sectors in DX:AX

MOV     AX,WORD PTR SS:[BP+1CH]
MOV     DX,WORD PTR SS:[BP+1EH]

MOV     CX,3           ; Boot Record is 3 sectors...

; Start loading reminder of Boot Record for FAT32
LoadIt:
DEC     CX
INC     AX              ; next Boot Record sector

JNZ     Skipper        ; AX wrap-around?
```

```
INC      DX      ; Yes, inc DX too
Skipper:
MOV      BX,7E00H ; into 0000:7E00
CALL     ReadSectorX ; Read Sector
JNC      ReadOK   ; no error?
MOV      AL,0F8H  ; what's this????
DEC      DI
JZ       NoMore   ; Jump if no more sectors
MOV      AX,SS:[BP+32H] ; get backup boot sector
XOR      DX,DX
MOV      CX,3
CMP      CX,AX    ; compare backup BS num
JA       ShowErrMsg1 ; with 3 (or vice versa)
; if 3 is higher than
; backup sector number,
; Bill's ooutta here...
MOV      SI,SS:[BP+0EH] ; SI = # of reserved sectors
CMP      CX,SI
JAE      ShowErrMsg1 ; same thing here... if 3 is
; higher then the number of
; reserved sectors... Bill's
; gone
SUB      SI,CX    ; get number reserved
sectors
; excluding the three boot
; sectors...
; add number of hidden sectors to DX:AX
ADD      AX,WORD PTR SS:[BP+1CH]
ADC      DX,WORD PTR SS:[BP+1EH]
JMP      LoadIt
NoMore:
JNC      ShowErrMsg1 ; Jump if carry=0
JMP      ShowErrMsg2
ReadOK:
CMP      WORD PTR SS:[BP+2AH],0
JA       ShowErrMsg1 ; Jump if not version 0.0?
```

```

                JMP      GOFAT32
ShowErrMsg1:
                MOV      SI,OFFSET ErrMsg1 + 7C00H
PrintMessage:
                LODSB                       ; get msg Skip length
                CBW
                ADD      SI,AX               ; Skip control data
NextChar:
                LODSB                       ; get chacacter
                TEST     AL,AL
                JZ       LastChar           ; End of string?
                CMP     AL,-1
                JE       SkipChar           ; End of first part?
                MOV     AH,0EH               ; TTY write character
                MOV     BX,7
                INT     10H
                JMP     NextChar            ; repeat write...
SkipChar:
                MOV     SI,OFFSET ErrMsg4 + 7C00H ; point to tail
message
                JMP     PrintMessage
ShowErrMsg2:
                MOV     SI,OFFSET ErrMsg2 + 7C00H
                JMP     PrintMessage
LastChar:
                CBW                          ; Ah, clever... save one byte, take
                ; advantage of the fact that LODSB
                INT     16H                  ; returns the null-terminator.
                POP     SI                   ; restore the stack... why???
                POP     DS                   ; the stack is killed at startup...
                POP     DWORD PTR [SI]
                INT     19H                  ; BIOS bootstrap loader...
Win95b         ENDP

```

```

;=====
;
;                      READ SECTOR
;=====

```

```
ReadSector      PROC      NEAR

                INC      CX          ; increase SECTOR COUNT

ReadSectorX:

rsReadMore:

                PUSH    SI
                PUSH    DWORD PTR 0
                PUSH    DX
                PUSH    AX
                PUSH    ES
                PUSH    BX
                PUSH    1
                PUSH    10H

                MOV     SI,SP        ; save stack pointer
                                        ; for later use by LEA

                PUSHA                ; Save "all" registers

                CMP     BYTE PTR SS:[BP+2],0EH    ; Use INT13 extensions?
                JNE     rsOldINT13

                MOV     AH,42H        ; Do ext INT13 READ
                JMP     RSDiskIO

rsOldINT13:

                XCHG   CX,AX          ; swap CX and AX
                XCHG   DX,AX          ; swap DX and AX
                XOR    DX,DX          ; clear DX

                DIV    WORD PTR SS:[BP+18H]    ; div LBA_HI by sectors/

track

                XCHG   CX,AX          ; save result in CX and put
                                        ; the LBA_LO in AX

                DIV    WORD PTR SS:[BP+18H]    ; divide remainder and LBA_LO
                                        ; by sectors/track too

                INC    DX              ; make sector 1-based

                XCHG   CX,DX          ; save it in CX and get the
                                        ; result of the 1st division
                                        ; in DX

                DIV    WORD PTR SS:[BP+1AH]    ; divide this new result by
```

```
                                ; heads per cylinder

MOV     DH,DL                    ; save Head of CHS in DH
                                ; head was in the remainder
                                ; after the division above

MOV     CH,AL                    ; save LO cylinder in CH
                                ; cylinder was in the result
                                ; after the division above

ROR     AH,2                     ; rotate AH to make bits 8-9
                                ; of cylinder appear as bits
                                ; 6-7 in AH and...

OR      CL,AH                    ; or it with the sector num

MOV     AX,201H                  ; setup for READ - 1 sector
rsDiskIO:

MOV     DL,SS:[BP+40H]           ; load drive number
INT     13H                      ; call INT13

POPA                                ; Restore "all" registers

; the entry code pushed 12h bytes on the stack...
; the last word pushed was 0001h, restore SP to point to it...

LEA     SP,[SI+10H]              ; Load effective addr

; Now, SI should contain 0001h

POP     SI

; was there an error from INT13?

JC      RSDone

INC     AX                        ; increment LBA sector num
JNZ     rsSkip                    ; wrap-around?

INC     DX                        ; yes raise high word too
rsSkip:

ADD     BX,SS:[BP+0BH]           ; increment by sector size

DEC     CX                        ; decrement SECTOR COUNT
JNZ     rsReadMore               ; Jump if more to read
rsDone:

RET
```

ReadSector            ENDP

;

=====  
;                    DATA AREA FOR MESSAGES - IN "NORSK" NORWEGIAN  
;=====  
=====

ErrMsg1            DB        03H        ; Skip counter for message1  
ErrMsg2            DB        18H        ; Skip counter for message2  
ErrMsg3            DB        01H        ; Skip counter for message3  
ErrMsg4            DB        27H        ; Skip counter for message4

                  DB        13,10,'Ugyldig systemdisk ',-1  
                  DB        13,10,'Disk I/U-feil ',-1  
                  DB        13,10,'Sett inn en annen disk, og trykk en  
tast',13,10,0

;

=====  
=====

                  DB        0,0                    ; Padding?

                  ; ROOT file names to search for...?

IO\_SYS             DB        'IO        SYS'  
MSDOS\_SYS          DB        'MSDOS     SYS'

                  DB        7EH,1,0                    ; What is this?

WINBOOT\_SYS       DB        'WINBOOT SYS'                    ; When is this used?

                  DB        0,0                    ; Padding?  
                  DW        0AA55H                    ; 1st Boot Signature

;

; BOOT RECORD - PART II - FSINFO sector

;

                  DB        'RRaA'                    ; FAT32 Extension Signature

                  DB        480 DUP (0)

; FSINFO information...

                  DB        'rrAa'                    ; FAT32 FSINFO Signature

```
brFreeClusters DD      56990          ; I have 233431040 bytes free!
brNextFree     DD      466175         ; My next free cluster!
               DD      3 DUP (0)      ; Reserved, acroding to FAT32API.HLP

               DW      ?              ; word padding

               DW      0AA55H         ; 2nd Boot Signature
;
; BOOT RECORD - PART III - FAT32 specific code, I think? only Bill knows?
;
GOFAT32:
        CLI

        ; calculate total size of FAT area

        MOVZX    EAX, BYTE PTR SS:[BP+10H] ; number of FATs
        MOV      ECX, SS:[BP+24H]         ; sectors per FAT
        MUL      ECX                     ; mul'em

        ; add hidden sectors

        ADD      EAX, SS:[BP+1CH]

        ; add reserved sectors

        MOVZX    EDX, WORD PTR SS:[BP+0EH]
        ADD      EAX, EDX

        XOR      CX, CX                  ; clear CX for some reason...
                                                ; By looking down the code, I can't
                                                ; seem to find out why CX is cleared
                                                ; It's set to 1 down there...
                                                ; before it's ever used...

        ; EAX will now point to the start of the data area (cluster 2)
        ; save start of data area below us at 0000:7BFC, or there around...

        MOV      SS:[BP-4], EAX

        ; Save another value to... This one is checked by GetFAT32Sector

        MOV      DWORD PTR SS:[BP-8], 0FFFFFFFFH

        ; Oh... at Microsoft they take no chances... disable INTs again!
        ; This is what I call proper software writing! Hail M$
```

CLI

; load Root Start Cluster in EAX

MOV EAX,SS:[BP+2CH]

; Is it cluster 2?

CMP EAX,2

JB ShowErrMsg1 ; error if less than 2

; Is it an EOF marker or something above?

CMP EAX,0FFFFFF8H

JAE ShowErrMsg1 ; error if it is

; Put upper 16-bits of cluster number into DX??

SHLD EDX,EAX,16

STI ; Puh. Safe again.

GetRootCluster:

PUSH DX

PUSH AX

CLI ; Eh?

; clear upper 16-bits of cluster number, and of course move up the  
; lower bits...

SHL EAX,16

; shift lower 16-bits of cluster number back down, and at the same  
; time shift in the high 16-bits in top of EAX?

SHRD EAX,EDX,16

; make cluster number 0-based... "the way it's supposed to be"

SUB EAX,2

; put Sectors Per Cluster in EBX

MOVZX EBX,BYTE PTR SS:[BP+0DH]

; save it in SI too! Yippi

```
MOV     SI,BX
```

```
; calculate relative sector of first part of root... right?
```

```
MUL     EBX
```

```
; add the "start of data area" value we saved below us!
```

```
ADD     EAX,SS:[BP-4]
```

```
; Maybe now, some shitter is trying to make DX:AX what EAX is??
```

```
; Shift upper 16-bits of EAX into DX... and AX is lower bits...
```

```
SHLD    EDX,EAX,10H
```

```
STI                                ; Enable interrupts
```

```
GetRootSector:
```

```
; Use 0070:0000 as a directory buffer...
```

```
MOV     BX,0700H
```

```
MOV     DI,BX
```

```
; read 1 sector
```

```
MOV     CX,1
```

```
CALL    ReadSectorX                ; this shit should be pretty  
JC      ShowErrMsg2                ; obvious...
```

```
CheckEntry:
```

```
CMP     [DI],CH                     ; is the first entry of the  
JE      EndOfRoot                   ; root empty???
```

```
MOV     CL,11                       ; the stupid CP/M filenames  
                                              ; are 11 bytes...
```

```
PUSH    SI
```

```
MOV     SI,OFFSET IO_SYS + 7C00H
```

```
REPE    CMPSB                       ; Is it IO.SYS?
```

```
POP     SI
```

```
JZ      FoundOS                     ; Yeah...
```

```
ADD     DI,CX                       ; add what's left after CMPSB
```

```
        ADD     DI,15H           ; and then 21 more...

; Yeah, yeah, anyway... point to the next dir entry...
; and check if it is above the last entry... INT13 increments
; BX with 512 on the sector read, so it points past the sector.

        CMP     DI,BX
        JB      CheckEntry      ; Jump if below

; are there any more sectors in this cluster???

        DEC     SI
        JNZ     GetRootSector    ; yeap, read more

        POP     AX               ; restore cluster number
        POP     DX

; Get FAT value... "GetFAT32Value" will compare the value with
; -8, and the JB below continues if below... that is, non-EOF/BAD
; the "previous cluster" value is taken from DX:AX (as restored
; above with POP).

        CALL    GetFAT32Value
        JB      GetRootCluster

; if not end of root... go to GetRootCluster..

EndOfRoot:                                ; EOF/BAD cluster...
        ADD     SP,4              ; clean up stack...
        JMP     ShowErrMsg1       ; and print error message

FoundOS:
        ADD     SP,4              ; clean up...

; Now... DI should point just above the IO.SYS name...

; SI would be set to DirEntry[14H] - starting cluster (HI)
; DI would be set to DirEntry[1AH] - starting cluster (LO)

        MOV     SI,[DI+09H]
        MOV     DI,[DI+0FH]

; copy FAT32 starting cluster upper 16-bits to AX

        MOV     AX,SI

        CLI                       ; Disable interrupts
```

```
; shift cluster high into upper half of EAX and store lower half  
; from DI into AX
```

```
    SHL     EAX,10H  
    MOV     AX,DI
```

```
; cluster out of range??
```

```
    CMP     EAX,2           ; clusters start with 2  
    JB     InvalidCluster
```

```
    CMP     EAX,0FFFFFF8H  ; cluster 0FFFFFF8 is EOF  
    JAE    InvalidCluster
```

```
    DEC     EAX           ; make it 0-based...  
    DEC     EAX
```

```
; Multiply cluster number with "sectors per cluster"
```

```
    MOVZX   ECX,BYTE PTR SS:[BP+0DH]  
    MUL     ECX
```

```
; Add the "start of data area" value that was saved back there...
```

```
    ADD     EAX,SS:[BP-4]
```

```
; And for the N'th time, make DX:AX same as EAX - sector number.
```

```
    SHLD    EDX,EAX,10H
```

```
    STI                    ; aha...
```

```
    MOV     BX,0700H       ; IO.SYS loads here!
```

```
    PUSH    BX  
    MOV     CX,4           ; load 4 IO.SYS sectors  
    CALL    ReadSectorX   ; 2K is minimum FAT32
```

cluster

```
    POP     BX           ; size anyway...
```

```
    JC     ShowErrMsg2   ; error...???
```

```
; COMMENT:  
;
```

```
; Now, there is enough code here... to read the entire IO.SYS  
; file into memory. This code has code to go through the FAT,  
; there is code to read cluster... bla bla. And still only 2K  
; of IO.SYS is read. If the entire file was read... IO.SYS would  
; not have to do this... well well.
```

```
; Is there a Mark Zibikowski in the room?
```

```
    CMP     WORD PTR [BX], 'ZM'           ; EXE signature...  
    JNE     InvalidCluster
```

```
; Is there a Barabara Jones in the room?
```

```
    CMP     WORD PTR DS:[0200H][BX], 'JB' ; IO.SYS signature?  
    JE      ExecutIOSYS
```

```
; The above shit appear in the IO.SYS file at offsets 0 and 200h  
; The MZ is the usual EXE signature while the "BJ" is unknown to  
; me. Maybe they chose it because it translates to harmless code:
```

```
;  
;   INC DX - DEC DX, pretty dull if you ask me ;)  
;
```

```
InvalidCluster:
```

```
    MOV     SI, OFFSET ErrMsg3 + 7C00H  
    JMP     PrintMessage
```

```
ExecutIOSYS:
```

```
    DB     0EAH           ; Jump to IO.SYS at 0070:0200  
    DW     0200H, 0070H
```

```
;  
;===== GET FAT32 VALUE =====  
;
```

```
GetFAT32Value PROC NEAR
```

```
    ADD     AX, AX           ; Multiply DX:AX by 4,  
    ADC     DX, DX  
    ADD     AX, AX           ; convert DX:AX from FAT32  
    ADC     DX, DX           ; index value to offset
```

```
; DX:AX is passed on as the FAT offset to lookup...
```

```
    CALL    GetFAT32Sector   ; read FAT sector
```

```
; the correct sector is returned... with DI as index...??  
; At least that's what the MOV below assumes...
```

```
        CLI

        MOV     EAX,ES:[BX+DI]           ; EAX = cluster value
; mask of top 4 bits of because Microsoft say it's reserved.
        AND     EAX,0FFFFFFFH

; Make DX:AX the cluster number too...

        SHLD   EDX,EAX,16                ; EAX[HI] into EDX[LO]

; Check for EOF/BAD

        CMP    EAX,0FFFFFFF8H           ; Is it the EOF marker?

        STI    ; return with ZF=1 if the
                ; last cluster was read??

        RET
```

GetFAT32Value ENDP

```
;=====
;                               GET FAT32 SECTOR
;=====
```

; On entry DX:AX is the FAT offset in bytes...

GetFAT32Sector PROC NEAR

```
; When this is called 0070:0200 seems to be the buffer in ES:BX
; but, the code below uses the DI value set down under here...
```

```
        MOV    DI,7E00H
```

```
        CLI    ; Disable interrupts
```

```
; make EAX the sector number again... move DX into top of EAX...
```

```
        SHL    EAX,16
        SHRD   EAX,EDX,16
```

```
; move bytes per sector into ECX
```

```
        MOVZX  ECX,WORD PTR SS:[BP+0BH]
```

```
; divide EDX:EAX by BPS... EAX = sector, EDX = offset in sector...

        XOR     EDX,EDX
        DIV     ECX

; Check FAT sector number agains... saved value on stack...
; This one is initially -1 (also known as 0FFFFFFFFH)

        CMP     EAX,SS:[BP-8]
        JE      LOC_30

; If sector is <> from -1, save this sector at 0000:7BF8

        MOV     SS:[BP-8],EAX

; add hidden sectors...

        ADD     EAX,SS:[BP+1CH]

; add reserved sectors too...

        MOVZX   ECX,WORD PTR SS:[BP+0EH]
        ADD     EAX,ECX

; get FAT32 flags into EBX

        MOVZX   EBX,WORD PTR SS:[BP+28H]

; keep "Active FAT" bits 0-3

        AND     BX,0FH

; If zero, we're at the correct FAT

        JZ      CorrectFAT

; compare active FAT with number of FATs...

        CMP     BL,SS:[BP+10H]
        JAE     ShowErrMsg1      ; oops... invalid active FAT

        PUSH   DX                ; save DX for a while...

; save FAT sector in ECX

        MOV     ECX,EAX
```

```
; Put sectors per fat in EAX
```

```
MOV EAX,SS:[BP+24H]
```

```
; Multiply active FAT number with sectors per FAT
```

```
MUL EBX
```

```
; Add to first FAT sector number we already had...
```

```
ADD EAX,ECX
```

```
; NOW, EAX contains the correct FAT sector number.
```

```
POP DX
```

CorrectFAT:

```
PUSH DX
```

```
; And for the N'th time, make DX:AX same as EAX - sector number.
```

```
SHLD EDX,EAX,16
```

```
STI ; Enable interrupts
```

```
MOV BX,DI ; read FAT sector into  
; 0000:7E00
```

```
; They sucker who wrote this could have saved 1 byte by  
; saying XOR CX,CX instead of MOV CX,1 and called ReadSector  
; instead of ReadSectorX, because there is an INC CX at  
; ReadSector... haha...
```

```
MOV CX,1 ; 1 sector
```

```
CALL ReadSectorX
```

```
POP DX
```

```
JC ShowErrMsg2
```

LOC\_30:

```
STI ; Enable interrupts
```

```
MOV BX,DX
```

```
RET
```

GetFAT32Sector ENDP

```
; Properly align the sector's boot signature at the end of
```

; the 3rd boot sect0r.

ORG 512 \* 3 - 2

DW 0AA55H ; 3rd Boot Signature

CODE ENDS

END

; - - - - -  
-  
;  
; CONCLUSION - THE END - HASTA LA VISTA - SLUTT - DET VAR ALT FOR I DAG.  
;  
; OK, Folks. that was quite a bit of work. It got pretty simple after some  
; hours.  
;  
; I would like to thank the following people...  
;  
; \* V Communications for Sourcer.  
; \* Ralf Brown for the Interrupt List.  
; \* Uriah Heep, The Who and Blind Guardian, for providing music.  
;  
; - - - - -  
-