

# An Examination of the Standard MBR ( Master Boot Record )

[ Embedded in Microsoft's FDISK Program  
from MS-DOS 4.01 through MS-Windows™ 95 (A) ]  
Daniel B. Sedory

This page examines the **Standard MBR** code which had been used in Microsoft's FDISK utility for many years; it's also the same as the Ranish Partition Manager's **Standard IPL** code (Initial Program Loader). However, since the introduction of the FAT32 file system (Windows 95B), the code created by FDISK is *no longer* as simple as what you'll find on this page; the MBR code presented here is also OS-independent.

**IMPORTANT:** One of the first things that any PC user should do after setting up a new hard disk (or creating a new partition with a utility such as *Partition Magic*) is to **make a copy of its MBR; especially if you have more than one partition on the disk!** Why? If you accidentally overwrite this sector, or are infected by a **Boot sector virus**, you may never be able to access some or even all of your disk again! Even the most expensive HD utility might not **correctly** restore the **Partition Table** of a **multi-partitioned** hard disk!

**Some advice:** Save the Partition Table data on floppy disks or even on paper(!); it does no good to have the data you need to access your HD on the *un-accessible* HD itself! There are many ways you can do this. Any good **Disk Editor** will allow you to manually enter data you've written down under an easy to use **Partition Table View**, or you could use the Ranish Partition Manager to save the binary data to a file and later restore the MBR from the saved file.

## Introduction

NOTE: Even though we're examining code created by an MS FDISK utility, the MBR is OS-independent: this code can be used to start the bootup process for any operating system's Boot Record on an x86-based computer. There have been many MBR or IPL (Initial Program Loader) programs created for booting any or even multiple OSs.

This page examines the "standard" code (prior to Windows 95B) that is written to Cylinder 0, Head 0, Sector 1 of your first Hard Drive by the so-called "undocumented" DOS command: 'FDISK /MBR'.

Here's a disk editor view of how the MBR is stored on your hard disk's first sector; that's Absolute (Physical) Sector 0 or CHS 0,0,1. (See Examination of the Code below to find out where this data ends up in the Memory of your computer.)

```
Absolute Sector 0 (Cylinder 0, Head 0, Sector 1)
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000:  FA 33 C0 8E D0 BC 00 7C 8B F4 50 07 50 1F FB FC  .3.....|..P.P...
0010:  BF 00 06 B9 00 01 F2 A5 EA 1D 06 00 00 BE BE 07  .....
0020:  B3 04 80 3C 80 74 0E 80 3C 00 75 1C 83 C6 10 FE  ...<.t.<.u.....
0030:  CB 75 EF CD 18 8B 14 8B 4C 02 8B EE 83 C6 10 FE  .u.....L.....
0040:  CB 74 1A 80 3C 00 74 F4 BE 8B 06 AC 3C 00 74 0B  .t.<.t.....<.t.
0050:  56 BB 07 00 B4 0E CD 10 5E EB F0 EB FE BF 05 00  V.....^.....
0060:  BB 00 7C B8 01 02 57 CD 13 5F 73 0C 33 C0 CD 13  ..|...W..._s.3...
0070:  4F 75 ED BE A3 06 EB D3 BE C2 06 BF FE 7D 81 3D  Ou.....}.=
0080:  55 AA 75 C7 8B F5 EA 00 7C 00 00 49 6E 76 61 6C  U.u.....|..Inval
0090:  69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61 62  id partition tab
00A0:  6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67  le.Error loading
00B0:  20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65  operating syste
00C0:  6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74  m.Missing operat
00D0:  69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00  ing system.....
00E0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
```

# An Examination of the Standard MBR ( Master Boot Record )

[ Embedded in Microsoft's FDISK Program  
from MS-DOS 4.01 through MS-Windows™ 95 (A) ]  
Daniel B. Sedory

```

00F0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0100:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0110:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0120:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0130:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0140:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0150:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0160:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0170:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0180:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0190:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
01A0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
01B0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 01  .....
01C0:  01 00 0B 7F BF FD 3F 00 00 00 C1 40 5E 00 00 00  .....?.....@^...
01D0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
01E0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
01F0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  .....U.
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

```

The first 139 bytes (00h through 8Ah) of the 512-byte sector are **executable code**, and the next 80 bytes (8Bh through DAh) contain **error messages**. The last 66 bytes of the sector contain the 64-byte **Partition Table** (1BEh through 1FDh); data in the Table area will depend upon the size, structure and file systems on each hard disk. The sector ends with the *Word-sized signature ID* of **AA55h** (often called the sector's *Magic* number. (On Intel CPU systems, hex Words are stored with the Low-byte first and the High-byte last.) The remaining 227 bytes (from DBh to 1BDh) are all *padding* which FDISK fills with bytes of zeros!

**NOTE:** Not all utilities are created alike! Using an MBR writing utility other than FDISK *may or may not* actually fill-in the *Non-used or padded section* of the MBR with zero-bytes. For example, if you use the Ranish Partition Manager to overwrite an MBR with the Standard IPL, it will only write 256 bytes (00h through FFh); the IPL plus 37 bytes of zeros. This leaves the rest of the MBR rather messy looking if it had already been filled with more than just 256 bytes of code.

After executing the POST (Power-On Self Test), the BIOS code loads this sector into memory at 0000:7C00 then executes it by carrying out a simple jump instruction to the copied code: JMP 0000:7C00 Unlike an OS boot sector though, this code must first copy itself to 0000:0600. This is necessary because the MBR code will later load the Boot Sector of the Active Partition into the same area of memory that **it** was first loaded into!

## An Examination of the Assembly Code

Here's a disassembled copy of the code (; with comments) after being loaded into memory by the BIOS at 0000:7C00 ( the 0000: Segment notation has been dropped from all the Memory locations listed below):

```

7C00 FA          CLI          ; Disable maskable Interrupts
7C01 33C0       XOR AX,AX    ; Zero out both the Accumulator and
7C03 8ED0       MOV SS,AX   ; the Stack Segment register.
7C05 BC007C    MOV SP,7C00 ; Set Stack Pointer to 0000:7C00
7C08 8BF4       MOV SI,SP   ; Source Index: Copy from here...
7C0A 50          PUSH        AX

```

# An Examination of the Standard MBR ( Master Boot Record )

[ Embedded in Microsoft's FDISK Program  
from MS-DOS 4.01 through MS-Windows™ 95 (A) ]  
Daniel B. Sedory

```
7C0B 07          POP ES          ; Zero Extra Segment
7C0C 50          PUSH          AX
7C0D 1F          POP DS          ; Zero Data Segment
7C0E FB          STI              ; Enable Interrupts again

7C0F FC          CLD              ; Clear Direction Flag
7C10 BF0006      MOV DI,0600     ; Destination Index: New copy of
                          ; code will begin at: 0000:0600
7C13 B90001      MOV CX,0100     ; Copy 256 Words (512 bytes)
                          ; (100 hex = 256 decimal)
7C16 F2          REPNZ          ; REPEAT unless Non-Zero (CX times;
                          ; unless ZF=1 after an iteration).
7C17 A5          MOVSW          ; Copy two bytes at a time.
7C18 EA1D060000 JMP 0000:061D   ; Jump to new copy of code...
```

*; Since the preceding routine copies itself and all of the following  
; code to 0000:0600 and then jumps to 0000:061D before continuing to  
; run, the following addresses have been changed to reflect the  
; code's actual location in memory at the time of execution.*

*; This next bit of code tries to find an entry in the partition table  
; that indicates at least one of them is ACTIVE (i.e., bootable). The  
; first byte of a partition entry is used as the indicator: If it's a  
; 0x80h, yes; if 0x00h then no it's not bootable. If none of the four  
; possible partitions is active, then an error message is displayed.*

```
061D BEBE07      MOV SI,07BE     ; Location of first entry
                          ; in the partition table
                          ; (see Sample Table below).
0620 B304        MOV BL,04       ; Maximum of 4 entries
0622 803C80      CMP BYTE PTR [SI],80 ; Is this one bootable?
0625 740E        JZ 0635        ; Yes, jump to next test!
0627 803C00      CMP BYTE PTR [SI],00 ; No; is it a 0x00? If not:
062A 751C        JNZ 0648        ; -> "Invalid partition table"
062C 83C610      ADD SI,+10     ; Checking the next entry...
                          ; (10h = 16 bytes per entry)
062F FECB        DEC BL         ; SUB 1 from Entry counter.
0631 75EF        JNZ 0622        ; Have all entries been tested?
0633 CD18        INT 18         ; Yes, and NONE of them were
                          ; bootable, so start...
                          ; ROM-BASIC (only available on
                          ; some IBM machines!) Many BIOS
                          ; simply display "PRESS A
                          ; KEY TO REBOOT" when an
                          ; Interrupt 18h is executed.
```

*; We found an Active partition, so all the other entries are checked  
; for being non-bootable (first byte = 0x00), or there's an error!  
; (Only one entry in the Partition Table can be marked as 'Active.')*

# An Examination of the Standard MBR ( Master Boot Record )

[ Embedded in Microsoft's FDISK Program  
from MS-DOS 4.01 through MS-Windows™ 95 (A) ]  
Daniel B. Sedory

; Before doing so, we load the Head, Drive, Cylinder and Sector data  
; into **DX** and **CX** for use by the DOS **Interrupt 13** commands later.

```
0635 8B14          MOV DX,[SI]          ; Drive -> DL / Head -> DH
                          ; For the standard MBR code,
                          ; DL will always be 80h; which
                          ; means ONLY the first drive
                          ; can be bootable! [ This part
                          ; of the code is often changed
                          ; by MBR replacements! ]

0637 8B4C02       MOV CX,[SI+02]      ; Sector -> CL / Cylinder -> CH

063A 8BEE         MOV BP,SI           ; Save offset of active entry
063C 83C610       ADD SI,+10         ; Do next entry
063F FECB        DEC BL           ; Is this the last entry?
0641 741A        JZ 065D           ; -> Jump to Boot-routine!
0643 803C00       CMP BYTE PTR [SI],00 ; Non-bootable entry?
0646 74F4        JZ 063C           ; Yes, check the next entry
```

; If there was an error, then this next routine displays the message that  
; **SI** points to. After printing the ASCII-Z string (null terminated), the  
; program 'hangs up' by going into an infinite loop (at 065B):

```
0648 BE8B06       MOV SI,068B        ; -> "Invalid partition table"

064B AC          LODSB           ; Load byte at [SI] into AL ...
                          ; and increment the SI value.

064C 3C00        CMP AL,00         ; Is it a zero-byte yet ?
064E 740B        JZ 065B           ; If yes, were done. If not ...
0650 56          PUSH SI          ; Store string pointer on stack.
0651 BB0700       MOV BX,0007       ; Use Function 0E (Write Text) of
0654 B40E        MOV AH,0E         ; DOS Interrupt 10 to send the
0656 CD10        INT 10           ; character in AL to the screen.
0658 5E          POP SI
0659 EBF0        JMP 064B

065B EBFE        JMP 065B        ; Infinite Loop. You must
                          ; power-down or Reboot!
```

; Now we can load the first sector of the Active Partition (on most drives,  
; this would be Absolute Sector 63 for the first or only partition of your  
; Hard Drive. Absolute Sectors 2 thru 62 are normally empty, unless you've  
; installed a large MBR replacement, disk translation software for a very  
; large HD or some kind of multi-OS or security/encryption boot code).

; The first two **words** of the partition entry are the drive/head  
; **and** the sector/cylinder numbers of the first partition sector.  
; This data is in the format required by the INT 13 calls below.



# An Examination of the Standard MBR ( Master Boot Record )

[ Embedded in Microsoft's FDISK Program  
from MS-DOS 4.01 through MS-Windows™ 95 (A) ]  
Daniel B. Sedory

```

07BE                                80 01
07C0 01 00 06 3F 3F C4 3F 00-00 00 81 1E 0C 00 00 00 ...???.?.....
07D0 01 C5 05 3F 7F 47 C0 1E-0C 00 40 0F 08 00 00 00 ...?.G....@.....
07E0 41 48 82 3F 7F 53 00 2E-14 00 00 BD 00 00 00 00 AH.?.S.....
07F0 41 54 83 3F BF 0F 00 EB-14 00 00 91 0B 00 55 AA AT.?......U.

```

And this is how it would be seen in a disk editor that can interpret Partition Table data:

| Partition Type | Active Boot | Starting Loc Cyl Head Sec | Ending Loc Cyl Head Sec | Relative sectors | Number of sectors |
|----------------|-------------|---------------------------|-------------------------|------------------|-------------------|
| DOS FAT-16     | Yes         | 0 1 1                     | 196 63 63               | 63               | 794241            |
| Extended       | No          | 197 0 1                   | 327 63 63               | 794304           | 528192            |
| LINUX Swap     | No          | 328 0 1                   | 339 63 63               | 1322496          | 48384             |
| LINUX Ext2FS   | No          | 340 0 1                   | 527 63 63               | 1370880          | 758016            |

**Note:** The sector must have a 'signature' of **0xAA55**. It's located at the very end of the partition table (remember that low-bytes appear first and high-bytes last). The BIOS checks for the signature and if it's not there, you'll get an error message like " **Drive not ready.**"

During boot-up, these locations are later replaced by communications parameters for COM2 thru COM4 and other data. Here's a listing of my own computer's memory after boot-up for these same memory locations:

```

07BE                                7A 02                                z.
07C0 AC 02 43 4F 4D 32 20 20 20 20 DC 00 70 00 00 80 ..COM2 ..p..€
07D0 7A 02 B2 02 43 4F 4D 33 20 20 20 00 00 0D 0E z...COM3 ....
07E0 00 80 7A 02 B8 02 43 4F 4D 34 20 20 20 E8 D2 .€z...COM4 ..
07F0 01 D7 07 38 CD 01 3B 0E 00 00 7C 04 18 00 59 04 .....i...|...Y.

```