The Standard Master Boot Record

Introduction

This page examines the "standard" code (used by *Microsoft* 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 (or 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 00  ing
system......
 00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 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 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 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 (**8B**h through **DA**h) contain **error messages**. The last 66 bytes of the sector contain the **64-byte Partition Table** (**1BE**h through **1FD**h); 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 **AA55**h (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!

If you ever see four non-zero Hex bytes at offsets 1B8h through 1BBh, you might like to know this is often the result of having placed the drive in a computer that was booted up with Windows™ **NT**, **2000** or **XP**. These bytes are called an **NT Drive Serial Number**.
[If you ever happen to replace a Win95B/98/98SE/ME **MBR** with this **Standard MBR**, those systems will change bytes **0DC**h through **0DF**h of the Standard MBR the next time you boot the OS. For all the details, see: Mystery Bytes.]

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!

This page examines the **Standard MBR** code which had been used in *Microsoft's* FDISK utility since **MS-DOS 3.30** (as a matter of fact only a small portion of its code was changed since it was first created for **IBM DOS** *version* **2.00** at the beginning of **1983**. So we've got a very good reason for using the phrase ***Standard*** **MBR** when discussing this code!).

This MBR is also the same as the **Standard IPL** (**I**nitial **P**rogram **L**oader) code for the Ranish Partition Manager and the "write new MBR code" function in many other utilities (TestDisk used this MBR code until version 5.7, in 2005, when we decided to use only *free* Open Source GNU GPL code, replacing the "standard MBR i386 boot sector code" by mbr-1.1.8). However, since the introduction of the FAT32 File System (in Win 95B), the code created by FDISK is *more complex* than what you'll find here. **Note:** the MBR code presented here is also OS-independent (you can use it to boot any OS on a **PC** *).

An Examination of the Standard MBR (Master Boot Record) [Embedded in Microsoft's FDISK Programs from MS-DOS 3.30 through MS-Windows™ 95 (A)]

Other *Microsoft* **MBR** pages:
The **MBR** created by **Windows 95B/98/98SE and ME**'s **FDISK**
The **MBR** created by **Windows 2000/XP/2003** Installs or **Disk Management** Utility
The **MBR** created by **Windows Vista OS** Installs or **Disk Management** Utility
The **MBR** created by **Windows 7** Installs or **Disk Management** Utility

You can learn a great deal about the instructions used here by obtaining the **x86 Opcode *Windows Help file*** and ***Ralf Brown's* Interrupt List** from my Intro to Assembly page. 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):

This shows a sample partition table and where it would appear in memory during the boot-up process. See the References Page for links to complete listings of Partition Identifiers. For a detailed explanation of how to interpret the data in the table, see the file PTGUIDE.TXT in my Mbrdemo.zip download below. This Sample Table contains entries for all four records; each record is 16 bytes long. I've underlined all the bytes in the first (7BEh through 7CDh) and last (7EEh through 7FDh) of the 4 records here:

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

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.**" (usually when using a floppy diskette), or "Operating System not found." (The message being dependent upon the BIOS code; most PhoenixBIOS, including those

modified for VMWare, display this one. But under BOCHS, you would see: "Boot failed: not a bootable disk" and on a PC using Award BIOS 6.00PG, it actually displays: DISK BOOT FAILURE, INSERT SYSTEM DISK AND PRESS ENTER.)

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:

Technically, Intel's F2 machine code byte will be ***disassembled*** as either REPNZ (by MS-DEBUG's U command) or REPNE (most other debuggers). I've used the simple REP *mnemonic* in my disassembly of the code above, because that's its real function here! Even though Intel CPUs end up performing a simple repetition of any MOVSW (or similar class of) instruction following an F2 byte (for as many times as the value found in the **CX** register), an ideal Assembly program should **never** use an F2 byte in such cases. Why? Well, apart from the fact that all respectable x86 assemblers encode a simple REP instruction only as an **F3** machine code byte, it confuses a lot of beginners trying to figure out how the repetition can continue when the Zero flag bit is often **NZ** (**not zero**) the whole time! The reason is because the CPU does **not** check the flag bits while repeating Move String, Byte or Word (MOVSB/MOVSW) instructions! It only checks the Direction Flag (df) to see if the Source (SI) and Destination (DI) registers should be incremented when df=0 ('cleared') or decremented when df=1 ('set') for the number in the CX (Count) register. Since it was preceded by the **CLD** instruction at **7C0F**, it will be incrementing the locations. The mnemonics REPE, REPZ, REPNE and REPNZ are supposed to apply ***only*** to the Compare (CMPSB/CMPSW) and Scan (SCASB/SCASW) String functions. Furthermore, since Intel has simply stated that: