
An Examination of the
Windows™ 7 or 8 or 8.1
MBR (Master Boot Record)

[Also embedded in **vdsutil.dll**,
winsetup.dll ***and various other***
System files]

This page examines the **Windows 7/ 8 / 8.1 MBR code**; it's the same exact bytes for all of these OS versions. We'll not only examine some interesting facts about this MBR sector's code and display each assembly instruction; as we did with our previous MBR/VBR pages, but also discuss some differences in how these OSs install on your computers.

Whenever you install **Windows 7/8** to a hard disk, even one with an existing MBR, its first sector may be overwritten with the **Windows 7/8 MBR** code. **(**Note: If necessary, it will also *change* the Volume Boot Record of an existing Active Windows boot OS; usually found in the first partition of a PC's first hard disk.**)** This code is also installed on ***blank*** hard drives when using **Windows 7/8**'s **Disk Management** utility.

NOTE: On our Windows **7** RC Install DVD, all 512 bytes of this **MBR (**including the *zero-bytes* in the partition table**)** were found in **boot\bootsect.exe**, **sources\upgdriver.dll** and **sources\winsetup.dll**; see below for more files containing this code.

Like all other MBRs presented in this series, this MBR code could still be used to boot any OS on some x86 **PC**s if it meets the conditions listed here*.

Other *Microsoft* **MBR** pages:
The **MBR** created by **Windows Vista** Installs or **Disk Management** Utility
The **MBR** created by **Windows 2000/XP/2003** Installs or **Disk Management** Utility
The **MBR** created by **Windows 95B/98/98SE** and ME's **FDISK**
An Examination of the **Standard MBR** created by **MS-DOS FDISK**

And *Microsoft* OS **Volume Boot Records**:
A Comparison of Windows™ Vista, 7 and 8 VBR Code
An Examination of the **Windows 8 / 10 OS Volume Boot Record**
An Examination of the **Windows 7 OS Volume Boot Record**
An Examination of the **Windows Vista OS Volume Boot Record**
An Examination of the **Windows 2000/XP OS Boot Record (NTFS)**
An Examination of the **Windows 95B/98/98SE/Me OS Boot Record (MSWIN4.1)**

Confused? Send us an email if you have a specific question about the MBR or any Boot Records...

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 write it down 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... See our MBR Tools Page. 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 can use a utility program, such as "MbrFix" (for Win *NT/2K, XP, 2003, PE, Vista & 7; even 64-bit versions!*) to save the binary data to a file on say a thumb drive, and later on restore the MBR from that saved file.

---

* NOTE: Even though we're examining code created by a *Microsoft* Operating System, this MBR can also be used to start the boot process for *any* operating system's Boot Record on an x86-CPU based (PC) computer ***as long as*** that OS is: **1)** on the Primary Master hard drive, **2)** set to be the only *Active* partition and **3)** it has a boot loader in the first sector of that partition. Most **Linux** OS distributions *can* install **LILO** or **GRUB** as a Boot Record rather than in the MBR and following sectors, so even the oldest MBR by Microsoft could still be used to boot Linux; as long as its boot code was *at or under the* ***1024 cylinder limit*** that is.

Furthermore, the processor must be an **80386** *or later* in order to use the Win7 MBR code, since it includes the "Operand-Size Prefix" (**66**h which can only be executed by an 80386 or later CPU. See Code, location 0659). When certain 16-bit assembly instructions, such as PUSHA (PUSH All registers onto the Stack), are prefixed by the byte 0x**66**, it forces that instruction to act as if it were running in 32-bit mode. So PUSHA effectively becomes a PUSHAD instruction, pushing all the 32-bit registers onto the Stack.

Windows 7 can also boot multiple OSs using its console (Command Prompt) program **BCDEDIT** (Boot Configuration Data Editor; "Sets properties in boot database to control boot loading") and **BOOTMGR** files, so for systems with multiple OSs, this would be a far more practical approach than changing the Active partition in the MBR each time you want to boot up a different OS. If you intend to install a Linux OS, it would probably be best to do so after installing Win7, then use the GRUB boot manager as the first boot code to load the others from. By the way, BCDEDIT (like any program that affects system files) must first have the Command Prompt window opened in Administrator mode (right-click on the icon or program name and choose to open it as Administrator). Running BCDEDIT without any switches will display a few facts about BOOTMGR and the Windows Boot Loader (another program first created for Vista); which is the Windows 7 OS Loader: `\Windows\system32` **`\winload.exe`**.

There have been many MBRs or **IPL**s (**I**nitial **P**rogram **L**oaders) created for booting an OS and even for booting multiple OSs. See Multi-OS Booting on our "Tools and References" Page for some alternative code and Boot Managers.

Introduction

Like Vista, if you install Windows 7/8 on a hard disk with no existing partitions, the first partition will start at Absolute Sector **2048** (counting from ***zero****;* Sector 0 is where the MBR is located). This is an offset of exactly 1 Binary Megabyte (2048 * 512 = 1,048,576 bytes) into the disk. In hexadecimal, this is an offset of 100,000 hex (100000h = 1 MiB). The main reason *Microsoft* gave for doing this is found in their article, KB-923332; with the number of sectors given only in hex: **0x800** = 2048 and **0x3F** = 63.

[Basically, since the starting offset for many disks, including the majority of Windows XP OS installs, was **63** (an *odd number*), they chose a starting offset that should give an *even number* of sectors for any *large-sector drive* manufacturers produce. It would cause performance issues on large-sector drives if there were a "misalignment" between the size of a physical sector and the partition(s). Western Digital and other drive manufacturers have been producing such hard disks for some time now; calling them *Advanced Format* drives, with physical sectors **8** times the size of a 512-byte sector (8 * 512 = 4096 bytes). But even with new disks using **4 KiB-sized** sectors, the Win7 offset of **1 MiB** still gives an equivalent offset of **256** such sectors (1048576/4096 or 2048/8). ***If*** Microsoft had picked an offset of any *even number* of sectors divisible by the size of a new large-sector, wouldn't that have solved any "misalignment" issue**?** So why not simply pick an offset of 32, 64 or even 128 KiB**?** Did Microsoft really want to be sure you could continue to use your Win7 OS on drives with *even much larger sector sizes***?** Whatever their thoughts on the issue may have been, technicians working with Win7 OS disks (on either current or future models) now have a much larger sized ***reserved space*** **(**1,048,576 bytes **vs.** 32,256 bytes**)** they must deal with. **(**Note: Do **not** confuse this generally ***unused*** *reserved space* with the new Windows 7 "**System Reserved**" **partition** described below.**)]**

However, *unlike Vista,* Microsoft added a further complication for those who must deal with software designed to make image copies of Windows 7 hard disks: For each standard install of Windows 7/8, the install DVD defaults to creating **two partitions! (NOTE:** This is **not** true if you use an Upgrade DVD where Win 7/8 must be installed onto a disk with an existing Windows partition; whether you choose to keep your data *or overwrite the whole partition,* its files will only be installed into the OS partition.**)**

It's also important to note: Some 'name brand' computer manufacturers insist on adding their own special partition(s), either *before* or *after* the OS partition(s); or *both*! For example, DELL PCs often include a small FAT16 formatted partition at the very beginning of the disk drive (type 0x**DE**), but 'name brand' PCs may also include a partition for restoring the entire OS partition to the state it was in when shipped from the factory.

Note: For the standard Windows 7 install, *to a disk that has **no** existing partitions,* the first partition will always be set to a size of only **100 MiB** and labeled "**System Reserved**". **[Note:** For a **Windows 8** OS install, the first partition is set to a size of **350 MiB** (i.e., 716,800 sectors).**]** Users may also be confused by the fact that although this partition is set as the Active partition, it's often hidden from them due to having **no drive letter** assigned to it; in which case, you need to use **Disk Management** (see Figure 3 below; if running Win 7, enter: **`diskmgmt`** into the "*Search programs and files*" box to open it) or some other utility to see the PC's partitions. Otherwise, if it is assigned a drive letter, it will be volume **E:**, since the DVD drive has traditionally been assigned to **D:**. So here's a case where under Microsoft Windows, a simple clean OS install not only has **two partitions**, but also has the drive letter **C:** assigned to the ***second partition*** on the disk; not the first.

So be aware of this when examining the MBR of a Win7/8 OS disk.
The typical (default) **Windows 7** OS partition table (with its **100 MiB** System partition as the first entry) will appear as:

```
===========================================================================
| B | FS TYPE |      START     |        END      |           |            |
| F |  (hex)  |  C     H     S |   C     H     S |  RELATIVE |    TOTAL   |
===========================================================================
| * |   07    |  0     32   33 |   12  223    19 |      2048 |     204800 |
|   |   07    | 12    223   20 | 1023  254    63 |    206848 |   nnnnnnnn |
|   |   00    |  0      0    0 |    0    0     0 |         0 |          0 |
|   |   00    |  0      0    0 |    0    0     0 |         0 |          0 |
===========================================================================
```

Figure 1. A Typical Windows 7 Partition Table.

where "nnnnnnnn" simply represents that partition's actual capacity in sectors for the main OS partition. The "RELATIVE" offset of the first partition is 2048 sectors; instead of the usual 63. For technicians, it may take some time getting used to seeing both a Starting CHS triple of 0,32,33 (instead 0,1,1) and an Ending CHS triple of 12,223,19 (for disks with 255 heads) rather than the 1023,254,63 we had become so familiar with seeing on many user's computers. The whole first entry above will appear as follows in a disk editor (showing the actual hex bytes rather than decimal values in the table above): "80 **20 21 00** 07 **DF 13 0C** **00 08** 00 00 **00 20 03** 00" (see below), where the Head and Sector values are **20**h and **21**h (in Cylinder 00h) for the Starting Sector. And **DF**h, **13**h and **0C**h for the Head, Sector and Cylinder values of the Ending Sector.

NOTE: For a Laptop/Notebook PC, the BIOS may use a different pseudo-CHS *geometry translation* for its 'Head' value. For example, if a Windows 7 PC's BIOS decides its hard disk should have only **240** Heads (instead of 255), the values you will find in your Partition Table's first entry should be: "80 **20 21 00** 07 **A3 13 0D** **00 08** 00 00 **00 20 03** 00" for an Ending CHS Triple of 13,163,19 which still results in a total of **204800** (32,000 hex) sectors (a capacity of 100 MiB) for the first partition.

The typical (default) **Windows 8** OS partition table (with its **350 MiB** System partition as the first entry) will appear as:

```
===========================================================================
| B | FS TYPE |      START     |        END      |           |            |
| F |  (hex)  |  C     H     S |   C     H     S |  RELATIVE |    TOTAL   |
===========================================================================
| * |   07    |  0     32   33 |   44  190    18 |      2048 |     716800 |
|   |   07    | 44    190   19 | 1023  254    63 |    718848 |   nnnnnnnn |
|   |   00    |  0      0    0 |    0    0     0 |         0 |          0 |
|   |   00    |  0      0    0 |    0    0     0 |         0 |          0 |
===========================================================================
```

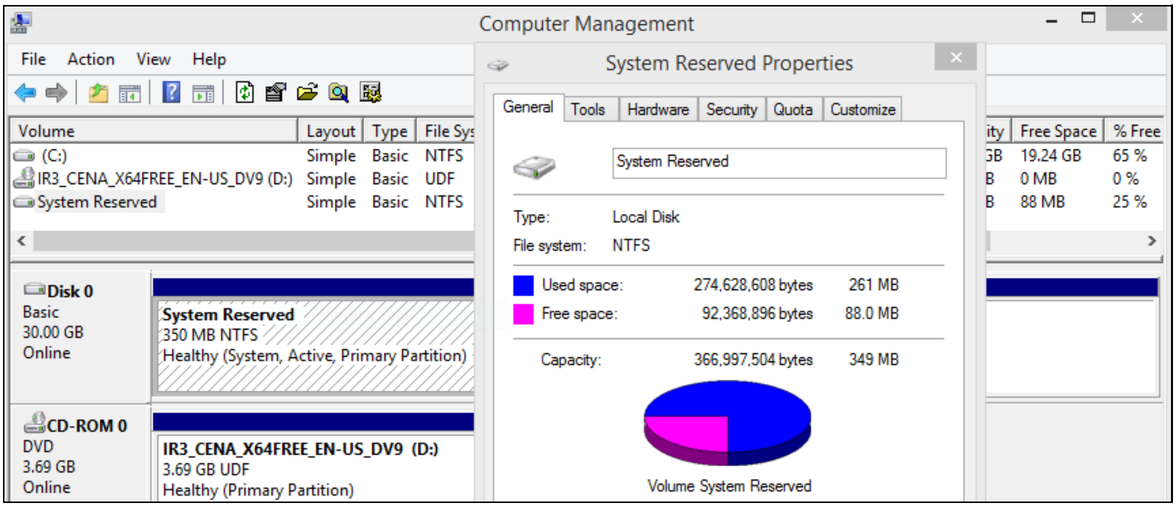
Figure 2. A Typical Windows 8 Partition Table.

where "nnnnnnnn" simply represents that partition's actual capacity in sectors for the main OS partition. The "RELATIVE" offset of the first partition is 2048 sectors; instead of the usual 63. For technicians, it may take some time getting used to seeing both a Starting CHS triple of 0,32,33 (instead 0,1,1) and an Ending CHS triple of 44,190,18 (for disks with 255 heads) rather than the 1023,254,63 we had become so familiar with seeing on many user's computers. The whole first entry above will appear as follows in a disk editor (showing the actual hex bytes rather than decimal values in the table above): "80 **20 21 00** 07 **BE 12 2C** **00 08** 00 00 **00 F0 0A** 00", where the Head and Sector values are **20**h and **21**h (in Cylinder 00h) for the Starting Sector. And **BE**h, **12**h and **2C**h for the Head, Sector and Cylinder values of the Ending Sector.

NOTE: Just as we stated above for Windows 7, the BIOS of a Laptop/Notebook PC may use a different pseudo-CHS *geometry translation* for its 'Head' value with a Windows 8 OS install.

Why is the Windows 8 / 8.1 *System Reserved* Partition 350 MiB (since it's only 100 MiB for Windows 7)?

Quick answer: Because the Windows 8 / 8.1 *System Reserved* Partition needs the room! With Windows 8, they decided to install the *Recovery Environment* right on the physical drive; of which the **`Winre.wim`** file (found in this drive's **`Recovery\WindowsRE\`** folder) is 225 MiB, so uses most of the additional space. Here we see that **75% of that 350 MiB is in use**:



Volume	Layout	Type	File Sys	ity	Free Space	% Free
(C:)	Simple	Basic	NTFS	GB	19.24 GB	65 %
IR3_CENA_X64FREE_EN-US_DV9 (D:)	Simple	Basic	UDF	B	0 MB	0 %
System Reserved	Simple	Basic	NTFS	B	88 MB	25 %

Disk 0
Basic
30.00 GB
Online

System Reserved
350 MB NTFS
Healthy (System, Active, Primary Partition)

CD-ROM 0
DVD
3.69 GB
Online

IR3_CENA_X64FREE_EN-US_DV9 (D:)
3.69 GB UDF
Healthy (Primary Partition)

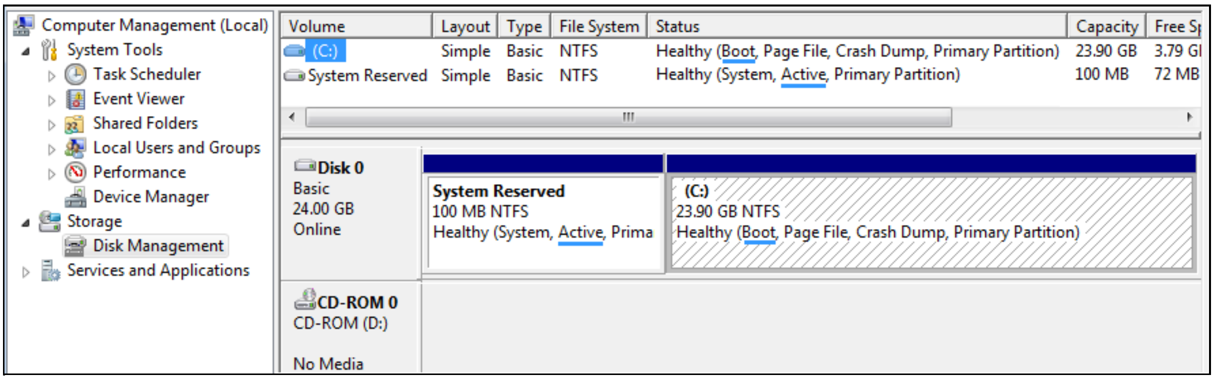
System Reserved

Type: Local Disk
File system: NTFS

🟦	Used space:	274,628,608 bytes	261 MB
🟪	Free space:	92,368,896 bytes	88.0 MB
	Capacity:	366,997,504 bytes	349 MB

Figure 3. Shows the *Properties* window of a Windows 8.1 System Reserved partition shortly after being installed.

The Terms *Active* and *Boot* under Windows 7 / 8



Volume	Layout	Type	File System	Status	Capacity	Free Sp
(C:)	Simple	Basic	NTFS	Healthy (Boot, Page File, Crash Dump, Primary Partition)	23.90 GB	3.79 GI
System Reserved	Simple	Basic	NTFS	Healthy (System, Active, Primary Partition)	100 MB	72 MB

Disk 0
Basic
24.00 GB
Online

System Reserved
100 MB NTFS
Healthy (System, Active, Prima

(C:)
23.90 GB NTFS
Healthy (Boot, Page File, Crash Dump, Primary Partition)

CD-ROM 0
CD-ROM (D:)

No Media

Figure 4. Disk Management view of small 25 GB Win 7 OS drive. The terms *Active* and *Boot* are no longer the same.

Prior to Windows 7, we often used the terms *Active* or *Bootable* as synonyms when discussing the partition that the Master Boot Record (MBR) code would load and execute in Memory from its Boot Sector, ***if*** it found the first byte of its Partition Table entry to be **80**h. However, when Microsoft programmer(s) created a more elaborate booting scheme, which could place the BOOTMGR code and BCD database in a separate partition from that of the Windows 7 OS, they decided to refer to the partition that contains the operating system as the ***Boot** partition* and the one that execution is initially passed to as the ***Active** partition.*

Windows 7 / 8 *Shrink* and *Expand* Utilities

Windows 7 does have the same useful feature related to boot records and booting which first appeared under Vista:
Its **Disk Management** utility has the ability to both ***shrink*** and ***expand*** partitions; similar to what *Partition Magic* could do for previous Windows versions. *We may present a detailed page about this in the future, **but** note:* All studies so far, have led us to the conclusion that no matter how much empty space you have

remaining within your last partition, this utility will allow you to shrink it to **only about 50% (just *half of*) the capacity of the physical disk drive!** Example: If the full capacity of a 320 GB disk drive was partitioned when installing the Windows 7 OS, this utility allows you to **shrink** the main OS partition to only about 160 GB; *even if* there is only 4 or 5 GB, or any other small number of GB, of that partition *in use!* And this will **not** change even if you run the utility again on the shrunk partition!

This page examines the **MBR code** most likely to be found in a *Microsoft*® Windows **7** OS installation. Win7's various editions, such as *Home Premium* or *Ultimate,* all contain the same exact MBR code. When partitioning a disk without an MBR sector, this code will be written to Cylinder 0, Head 0, Sector 1 (that's *Absolute* or **LBA** Sector **0**) of the **Disk Drive** by various OS routines, such as Win7's **Disk Management** utility. But even in the case of a drive that already has a functioning Windows MBR, the Win7 install DVD will overwrite the existing MBR **code** of the boot disk as part of the process. [As with Windows XP and Vista, Windows 7 itself will write *data* to an existing MBR sector (e.g., of a slave drive connected to the system), when necessary (compare *Disk Signature* comments for the Windows XP MBR).]

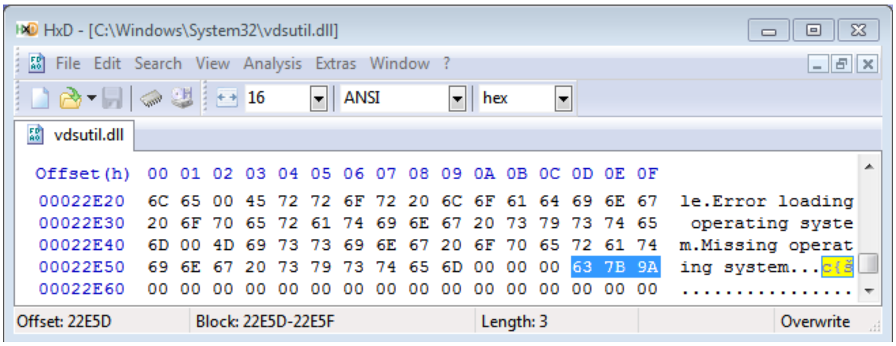
Where Copies of the MBR Code can be Found:

This link shows where copies of the MBR can be found for a **Windows 8.1** ISO file or DVD, or the files on its installed OS drive.

For our **Windows 7** install, all the bytes of Win7's MBR code were also contained inside the following files (listed by location, alphabetically; with offset to first byte of the code). In each case, there will be a full **512** bytes that comprise the MBR code (the location for the *NT Disk Signature* and the 64-byte Partition Table are all zero-filled, the last two bytes being 55h followed by AAh):

1. `C:\Windows\System32\`**`RelPost.exe`** [Offset: 12CD0h]
("Windows Diagnosis and Recovery"; File version: "6.1.7600.16385 (win7_rtm.090713-1255)"; 182,784 bytes; Modification Date: "07/14/2009 1:14 AM"). There's also a second copy here: `C:\Windows\winsxs\x86_microsoft-windows-`**`reliability-postboot`**`_31bf3856ad364e35_6.1.7600.16385_none_4d97265566a66f7e\`**`RelPost.exe`**.
2. `C:\Windows\System32\`**`vdsutil.dll`** [Offset: 22CA8h]
("Virtual Disk Service Utility Library"; File version: "6.1.7600.16385 (win7_rtm.090713-1255)"; 151,040 bytes; Modification Date: "07/14/2009 1:16 AM"). There's also a second copy here: `C:\Windows\winsxs\Backup\x86_microsoft-windows-`**`virtualdiskservice`**`_31bf3856ad364e35_6.1.7600.16385_none_6ac128c35c0231aa_`**`vdsutil.dll`**`_f2ef43cf`.
3. `C:\Windows\System32\`**`vssapi.dll`** [Offset: E20D0h]
("Volume Shadow Copy Requestor/Writer Services API DLL"; File version: "6.1.7600.16385 (win7_rtm.090713-1255)"; 1,123,328 bytes; Modification Date: "07/14/2009 1:16 AM").
There's also a second copy here: `C:\Windows\winsxs\Backup\x86_microsoft-windows-`**`vssapi`**`_31bf3856ad364e35_6.1.7600.16385_none_d4bd3473e31540bf_`**`vssapi.dll`**`_51f72c64`.
4. `C:\Windows\System32\`**`VSSVC`**`.exe` [Offset: E1BA8h]
("Volume Shadow Copy Service"; File version: "6.1.7600.16385 (win7_rtm.090713-1255)"; 1,025,536 bytes; Modification Date: "07/14/2009 1:15 AM").
There's also a second copy here: `C:\Windows\winsxs\x86_microsoft-windows-`**`vssservice`**`_31bf3856ad364e35_6.1.7600.16385_none_5aa3249a792b0938\`**`VSSVC.exe`**
5. `C:\Windows\System32\oobe\`**`winsetup.dll`** [Offset: 184220h]
("Windows System Setup"; File version: "6.1.7600.16385 (win7_rtm.090713-1255)"; 1,794,048 bytes; Modification Date: "07/14/2009 1:16 AM").
There's also a second copy here: `C:\Windows\winsxs\x86_microsoft-windows-`**`setup-component`**`_31bf3856ad364e35_6.1.7600.16385_none_3202d4720e95de08\`**`winsetup.dll`**.

Using the file "`C:\Windows\System32\`**`vdsutil.dll`**" of "151,040 bytes" with a Modification Date of "Tuesday, July 14, 2009 01:16:17 AM" as an example, the MBR in this file was found at offsets **`22CA8`**h through **`22EA7`**h (of which only 80 of its 512 bytes are shown here):



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00022E20  6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67  le.Error loading
00022E30  20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65   operating syste
00022E40  6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74  m.Missing operat
00022E50  69 6E 67 20 73 79 73 74 65 6D 00 00 00 63 7B 9A  ing system...c{š
00022E60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Offset: 22E5D
Block: 22E5D-22E5F
Length: 3
Overwrite

Figure 5. Showing the bytes "63 7B 9A"which are part of the Win7 MBR's *code.*

Disk Editor View of the Windows 7 / 8 MBR

The following is a ***disk editor view*** of how the bytes of this MBR are stored on a 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 Memory when it's executed.)

```
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   33 C0 8E D0 BC 00 7C 8E C0 8E D8 BE 00 7C BF 00   3.....|......|..
0010   06 B9 00 02 FC F3 A4 50 68 1C 06 CB FB B9 04 00   .......Ph.......
0020   BD BE 07 80 7E 00 00 7C 0B 0F 85 0E 01 83 C5 10   ....~..|........
0030   E2 F1 CD 18 88 56 00 55 C6 46 11 05 C6 46 10 00   .....V.U.F...F..
0040   B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55 AA 75 09   .A..U..]r...U.u.
0050   F7 C1 01 00 74 03 FE 46 10 66 60 80 7E 10 00 74   ....t..F.f`.~..t
0060   26 66 68 00 00 00 00 66 FF 76 08 68 00 00 68 00   &fh....f.v.h..h.
0070   7C 68 01 00 68 10 00 B4 42 8A 56 00 8B F4 CD 13   |h..h...B.V.....
0080   9F 83 C4 10 9E EB 14 B8 01 02 BB 00 7C 8A 56 00   ............|.V.
0090   8A 76 01 8A 4E 02 8A 6E 03 CD 13 66 61 73 1C FE   .v..N..n...fas..
00A0   4E 11 75 0C 80 7E 00 80 0F 84 8A 00 B2 80 EB 84   N.u..~..........
00B0   55 32 E4 8A 56 00 CD 13 5D EB 9E 81 3E FE 7D 55   U2..V...]...>.}U
00C0   AA 75 6E FF 76 00[E8 8D 00 75 17 FA B0 D1 E6 64   .un.v....u.....d
00D0   E8 83 00 B0 DF E6 60 E8 7C 00 B0 FF E6 64 E8 75   ......`.|....d.u
00E0   00 FB B8 00 BB CD 1A 66 23 C0 75 3B 66 81 FB 54   .......f#.u;f..T
00F0   43 50 41 75 32 81 F9 02 01 72 2C 66 68 07 BB 00   CPAu2....r,fh...
0100   00 66 68 00 02 00 00 66 68 08 00 00 00 66 53 66   .fh....fh....fSf
0110   53 66 55 66 68 00 00 00 00 66 68 00 7C 00 00 66   SfUfh....fh.|..f
0120   61 68 00 00 07 CD 1A]5A 32 F6 EA 00 7C 00 00 CD   ah.....Z2...|...
0130   18 A0 B7 07 EB 08 A0 B6 07 EB 03 A0 B5 07 32 E4   ..............2.
0140   05 00 07 8B F0 AC 3C 00 74 09 BB 07 00 B4 0E CD   ......<.t.......
0150   10 EB F2 F4 EB FD[2B C9 E4 64 EB 00 24 02 E0 F8   ......+..d..$...
0160   24 02 C3]49 6E 76 61 6C 69 64 20 70 61 72 74 69   $..Invalid parti
0170   74 69 6F 6E 20 74 61 62 6C 65 00 45 72 72 6F 72   tion table.Error
0180   20 6C 6F 61 64 69 6E 67 20 6F 70 65 72 61 74 69    loading operati
0190   6E 67 20 73 79 73 74 65 6D 00 4D 69 73 73 69 6E   ng system.Missin
01A0   67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74   g operating syst
01B0   65 6D 00 00 00 63 7B 9A D4 34 A0 2E 00 00 80 20   em...c{..4.....
01C0   21 00 07 DF 13 0C 00 08 00 00 00 20 03 00 00 DF   !.......... ....
01D0   14 0C 07 FE FF FF 00 28 03 00 hh hh hh hh 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
```

Figure 6.

The first 355 bytes (000h through 162h) of this 512-byte sector are **executable code** and the next 80 bytes (**163**h through **1B2**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 will depend upon the size, structure and file systems on the hard disk. [See our pages on Partition Tables, for notes on how to interpret the data in a particular disk's partition table.] The sector ends with the *Word-sized* signature ID of **AA55**h (sometimes called the MBR's *Magic* number). **Note:** On PCs using an Intel (or x86 compatible) CPU, hex Words are stored with the Low-byte first and the High-byte last.

The programmers of this MBR had to use almost every bit of space available, because all **110** bytes of the code shown between brackets in **Figure 6** (offsets **C6**h through **126**h *and* offsets **156**h through **162**h) are either related to, or directly involved in, determining if the hardware supports **TPM** (Trusted Platform Module) version **1.2**; which can then be used to provide extra functionality for the Microsoft Windows BitLocker™ Drive Encryption**.** The letters "TCPA" at offsets **EF**h through **F2**h are ***not*** coincidental; they stand for "Trusted Computing Platform Alliance" and are part of the code which tests for the existence of a **TPM** chip (see comments below).

The remaining 11 bytes (***between*** the **Error Messages** and the **Partition Table**; 1B3h through 1BDh) begin with only two zero-bytes as *padding*; followed by the three bytes (**63 7B 9A**) at **1B5**h through **1B7**h for a Win7 install with English messages (see below for all the details about this). If you stop the installation before any NT-type Operating Systems starts to boot-up, the next four bytes may remain as they were; usually zero-bytes. But once Windows has begun running, it will write a ***Disk Signature*** in the **MBR**. These **four** bytes from offsets **1B8**h through **1BB**h are called the Windows **Disk Signature** *or* NT Drive Serial Number. See here for details on Disk Signature use in the Windows Registry!

The **three bytes** at offsets **1B5**h through **1B7**h ("**63 7B 9A**") are used by *Microsoft* Windows for a very specific purpose; for **English** versions of Windows **7**, you'll ***always*** see these same Hex values ("63 7B 9A") in the **MBR**. They're used by the MBR code to display **Error Messages** on your screen. But for those using Windows 7 in a different language, their MBRs may have different values in the **second** and **third** bytes depending upon how many characters are in each of the three messages. **If** you look in the code section below, starting at offset **0731**h (instruction: "MOV AL,[0**7B7**]"), you'll see these **three** bytes are used to reference the ***offset*** in Memory of the first byte of each **Error Message** that can be displayed on screen at boot up: **0763**h, **077B**h and **079A**h. Since the **code** portion above the messages will always be the same, **the first offset** (**0763**h) ***will never change*** no matter what languages (and string lengths) are used.

Now that you know ***what*** the bytes at offsets **1B5**h through **1B7**h are used for, you could change these error messages to display whatever you wish (***as long as they all fit into the space*** between offsets **163**h and **1B4**h) by counting their character lengths and using a disk editor on the MBR sector to make the appropriate changes.

After executing the POST (Power-On Self Test), the BIOS loads this sector into memory at 0000:**7C00** (as it does any MBR) then transfers control to this code.

But this code must first copy itself into another area of Memory. This is *necessary* because the code must also load the Boot Sector of the *Active* Partition into the same area of Memory that **it** occupies just after being loaded! Unlike the Windows 2000/XP MBR, this code copies all 512 of its bytes to the new location, starting at: 0000:0**600**. Only the first three instructions are the same as the Windows 2000/XP MBR, so keep your eyes sharp if you're comparing the two.

An Examination of the Assembly Code

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 our Intro to Assembly page.

> **NOTE:** We've begun some new pages called Pathways through the Windows 7 MBR which graphically display all registers (and include detailed comments) for each step of the MBR code as it is being executed. These pages refer to the Bochs *Enhanced* Debugger as the tool chosen to provide these illustrative steps (We would appreciate any email comments or questions you might have concerning these pages).

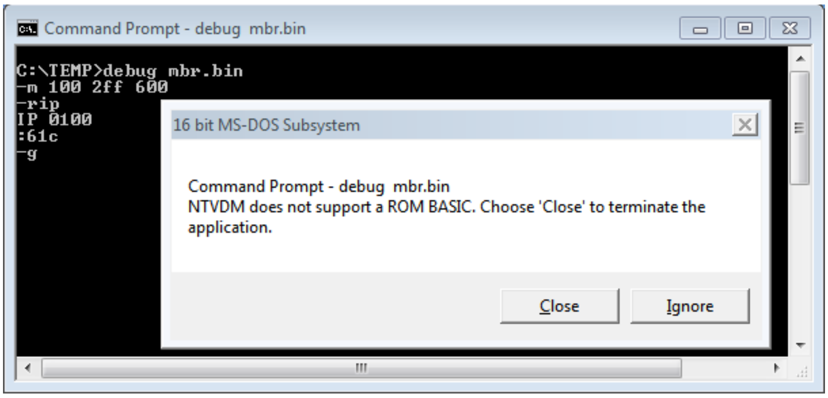
Here's a Listing of the disassembled code (; with comments) after first being loaded into Memory at **0000**:7C00 by the **BIOS** (all Memory locations listed below are in Segment **0000:**). If you see an asterisk (*) next to an instruction, it means that MS-DEBUG can ***not*** disassemble that code.

Note: If you compare this code to that of the Windows Vista MBR, you'll find there's only a slight variation due to changing two jump instructions from 32-bit to 16-bit at Memory locations 06A2 and 06C9; allowing for the insertion of a **CLI** instruction immediately after it, and an **STI** instruction at 06E1. Lastly, they also decided to add a **HLT** (Halt) instruction at 0753. Due to these changes, which added 12 more bytes to the Windows 7 MBR code, various offset bytes also needed to be adjusted. (The first byte difference from the Vista MBR code occurs at Memory location 062B, where the location to jump to changes from 073D to 073B. The next adjustment doesn't occur until 069E; just a few bytes before all the same code bytes become shifted by 1 to 4 bytes compared to Vista's MBR.)

```
7C00 33C0          XOR   AX,AX         ; Zero out the Accumulator and
7C02 8ED0          MOV   SS,AX         ;   Stack Segment register.
```

```
7C04 BC007C        MOV   SP,7C00       ; Set Stack Pointer to 0000:7C00
7C07 8EC0          MOV   ES,AX         ; Since AX=0, zero-out Extra Segment,
7C09 8ED8          MOV   DS,AX         ;  and zero-out Data Segment.
7C0B BE007C        MOV   SI,7C00       ; Source Index: Copy from here...
7C0E BF0006        MOV   DI,0600       ; Destination Index: Copy to here:
                                       ;  Code will begin at: 0000:0600
7C11 B90002        MOV   CX,0200       ; Set up Counter (CX) to copy all
                                       ; (200h) 512 bytes of the code.
7C14 FC            CLD                 ; Clear Direction Flag
7C15 F3            REP                 ;/ REPeat the following MOVSB
                                       ;| instruction for 'CX' times;
7C16 A4            MOVSB               ;\ copying one byte at a time.

; Note: Some debuggers disassemble the last two instructions above as:
;                  REP MOVSB BYTE PTR ES:[DI], BYTE PTR DS:[SI]
; making clear the source [SI] and destination [DI] of the bytes being copied.
;  CX will count down from 200h to zero while DI increases in step up to 800h.

7C17 50            PUSH  AX            ; Set up Segment(AX) and Offset(DI)
7C18 681C06      * PUSH  061C          ;  for jump to 0000:061C.
7C1B CB            RETF                ; Use RETF to do Jump into where we
                                       ;  copied all the code: 0000:061C.


; Since the preceding routine not only copies the MBR code to a new location, but
; also jumps there to continue its execution, the following addresses have been
; changed to reflect the code's actual location in memory at the time of execution.

; This next section of code tries to find an ACTIVE (i.e., bootable) entry in the
; Partition Table. The first byte of an entry indicates if it's bootable (an 80h)
; or not(a 00h); any other values in these locations means the Table is invalid!

; If none of the four entries in the Table is active, the 'Invalid' error message
; is displayed. [Microsoft MBR code prior to 2000/XP used the SI register instead
; of BP; offsets 0620, 0623 and 062D below show how BP can be used.]

061C FB            STI                   ; Enable Interrupts
061D B90400        MOV   CX,0004         ; Maximum of four entries.
0620 BDBE07        MOV   BP,07BE         ; Location of first entry in the
                                         ; partition table (see Sample Table here).

0623 807E0000      CMP   BYTE PTR [BP+00],00  ;    CoMPare first byte of entry at
                                              ; SS:[BP+00] to Zero. Anything from
                                     ; 80h to FFh will be less than, which means:
0627 7C0B          JL    0634        ;  We found a possible boot entry, so we
                                     ;  check on it in more detail at 0634h
0629 0F850E01    * JNZ   073B        ; But if it's not zero (and greater than),
                                     ; then we have an Error! (Because we must
                                     ; have found a 01h through 79h byte.) So:
                                     ; -> "Invalid partition table"
                                     ; Otherwise, we found a zero; so we keep on
062D 83C510        ADD   BP,+10      ; searching: Check the next entry ...
                                     ; (there are 10h = 16 bytes per entry)
0630 E2F1          LOOP  0623        ; Go back & check next Entry ...
                                     ;  unless CL = 0 (we tried all four).

0632 CD18          INT   18          ; Checked all 4; 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.
```

Just like most of the MBR code we've studied here, if you were to load a copy of the Win7 MBR with an empty partition table (or one that has no Active Boot Flag for any of its entries) as a *.bin file into MS-DEBUG (e.g., **`debug mbr.bin`**), move all the code to offset 0x0600 (**`-m 100 2ff 600`**), set the IP to 0x061C (**`-rip`** then **`61c`**) and run it (**`-g`**; Note: Although MS-DEBUG cannot step through the code at 0x0629 since it doesn't understand instructions beyond the 8086 through 8088 processors, when you enter 'g' it will simply pass all the code it encounters to Win XP/Vista/Win7's NTVDM program; under which you are actually running DEBUG, without trying to disassemble it), you would then see the following error message on your screen:



```
C:\TEMP>debug mbr.bin
-m 100 2ff 600
-rip
IP 0100
:61c
-g
```

16 bit MS-DOS Subsystem

Command Prompt - debug mbr.bin
NTVDM does not support a ROM BASIC. Choose 'Close' to terminate the application.

Note: Although this was possible under Windows 7 RC, the retail Windows 7 OS no longer includes the MS-DEBUG program! We recommend installing VMWare Player and creating your own Windows XP (or Windows 98 or even MS-DOS), *virtual computer* to run MS-DEBUG inside of.

Because NTVDM was never programmed to handle an **18**h Interrupt any further than displaying this message. If you Ignore it, the NTVDM program might warn you one more time about accessing the hard disk, but it will eventually freeze or go off into oblivion if you Ignore it again; and you'll have to use Task Manager to close the DOS Prompt window gracefully.

```
0634 885600        MOV   [BP+00],DL          ; DL is already set to 80h.
                                             ; (Presumably by PC's BIOS.)
0637 55            PUSH  BP                  ; Save Base Pointer on Stack.
0638 C6461105      MOV   BYTE PTR [BP+11],05 ; Data storage for possible use
                                             ; by instruction at 069F.
063C C6461000      MOV   BYTE PTR [BP+10],00 ; Used as a flag and/or counter
                                             ; for the INT13 Extensions being
                                     ;  installed (see 0656 and 065B below).
0640 B441          MOV   AH,41       ;/
0642 BBAA55        MOV   BX,55AA     ;|
0645 CD13          INT   13          ;| INT13, Function 41h (with BX=55AAh):
                                     ;| Check for Int 13 Extensions in BIOS.
;|  If CF flag cleared and [BX] changes to AA55h, they are installed;  Major
;| version is in AH: 01h=1.x; 20h=2.0/EDD-1.0; 21h=2.1/EDD-1.1; 30h=EDD-3.0.
;| CX = API subset support bitmap. If bit 0 is set (CX = 1, 3, 5, etc.; 'odd'),
;|      then extended disk access functions (AH=42h-44h,47h,48h) are supported.
;\      Only if no extended support is available, will it fail TEST at 0650.

0647 5D            POP   BP                ; Get back original Base Pointer.
0648 720F          JB    0659              ; Below? If so, CF=1 (not cleared)
                                           ;   so no INT 13 Ext. & do jump!
064A 81FB55AA      CMP   BX,AA55           ; Did contents of BX change?  If
064E 7509          JNZ   0659              ;   not, jump to offset 0659.
0650 F7C10100      TEST  CX,0001           ; Final test for INT 13 Extensions!
                                           ; If bit 0 not set, this will fail,
0654 7403          JZ    0659              ;   then we jump over next line...
0656 FE4610        INC   BYTE PTR [BP+10]  ; or increase [BP+10h] by one.

0659 6660        * PUSHAD                  ; Save all 32-bit Registers on the
                                           ; Stack in this order: eax, ecx,
                                           ; edx, ebx, esp, ebp, esi, edi.
065B 807E1000      CMP   BYTE PTR [BP+10],00  ;/ CoMPare [BP+10h] to zero;
065F 7426          JZ    0687                 ;\ if 0, can't use Extensions.


; The following code uses INT 13, Function 42h ("Extended Read") to read the
; first sector (VBR) of the bootable partition into Memory at location 0x7c00.
; It does this by first pushing what's called the "Disk Address Packet" onto
; the Stack in reverse order of how it will read the data, so 00h (Reserved)
; and 10h bytes are the last to be pushed onto the Stack at location 0674:
;
; Offset Size          Description of DISK ADDRESS PACKET's Contents
; ------ -----  ------------------------------------------------------------
;   00h  BYTE   Size of packet (10h or 18h; 16 or 24 bytes).
;   01h  BYTE   Reserved (00).
;   02h  WORD   Number of blocks to transfer (Only 1 sector for this code).
;   04h  DWORD  Points to -> Transfer Buffer (0000 7C00 for this code).
;   08h  QWORD  Starting Absolute Sector (get from Partition Table entry:
;               (00000000 + DWORD PTR [BP+08]). Remember, the Partition
;               Table Preceding Sectors entry can only be a max. of 32 bits!
;   10h  QWORD  NOT USED HERE. (EDD-3.0; optional) 64-bit flat address
;               of transfer buffer; only used if DWORD at 04h is FFFF:FFFF.

0661 666800000000  * PUSH    00000000  ; Push 4 zero-bytes (32-bits) onto
                                       ; Stack to pad VBR's Starting Sector.
0667 66FF7608      * PUSH    DWORD PTR [BP+08]   ; Location of VBR Sector.
066B 680000        * PUSH    0000      ; \ Segment then Offset parts, so:
066E 68007C        * PUSH    7C00      ; / Copy Sector to 0x7c00 in Memory.
0671 680100        * PUSH    0001      ;   Copy only 1 sector.
0674 681000        * PUSH    0010      ; Reserved and Packet Size (16 bytes).
0677 B442            MOV     AH,42      ; Function 42h.
0679 8A5600          MOV     DL,[BP+00] ; Drive Number.
067C 8BF4            MOV     SI,SP      ; DS:SI must point to -> "Disk
                                        ;        Address Packet" on Stack.
067E CD13            INT     13         ; Try to get VBR Sector from disk.

; If successful, CF (Carry Flag) is cleared (0) and AH set to 00h.
; If any errors, CF              is set to 1    and AH = error code. In either
; case, DAP's block count field is set to number of blocks actually transferred.

0680 9F            LAHF                ; Load Status flags into AH.
0681 83C410        ADD    SP,+10       ; Effectively removes all the DAP bytes
                                       ; from Stack by changing Stack Pointer.
0684 9E            SAHF                ; Save AH into flags register, so we do
                                       ;  not change Status flags by doing so!
0685 EB14          JMP    069B

; The MBR uses the standard INT 13 "Read Sectors" function here, because
; no INT 13 Extended functions were found in the BIOS code above (065F):

0687 B80102        MOV    AX,0201      ; Function 02h, read only 1 sector.
068A BB007C        MOV    BX,7C00      ; Buffer for read starts at 7C00.
068D 8A5600        MOV    DL,[BP+00]   ; DL = Disk Drive
0690 8A7601        MOV    DH,[BP+01]   ; DH = Head number (never use FFh).
0693 8A4E02        MOV    CL,[BP+02]   ; Bits 0-5 of CL (max. value 3Fh)
                                       ; make up the Sector number.
0696 8A6E03        MOV    CH,[BP+03]   ; Bits 6-7 of CL become highest two
                                       ; bits (8-9) with bits 0-7 of CH to
                                       ; make Cylinder number (max. 3FFh).
0699 CD13          INT    13           ; INT13, Function 02h: READ SECTORS
```

```
                                       ; into Memory at ES:BX (0000:7C00).
```

The following code is missing some comments, but all the instructions are here for you to study.

```
; Whether Extensions are installed or not, both routines end up here:

069B 6661        * POPAD               ; Restore all 32-bit Registers from
                                       ; the Stack, which we saved at 0659.
069D 731C          JNB    06BB
069F FE4E11        DEC    BYTE PTR [BP+11]     ; Begins with 05h from 0638.
06A2 750C          JNZ    06B0                 ; If 0, tried 5 times to read
                                               ; VBR Sector from disk drive.
06A4 807E0080      CMP    BYTE PTR [BP+00],80
06A8 0F848A00    * JZ     0736                 ;  -> "Error loading
                                               ;   operating system"
06AC B280          MOV    DL,80
06AE EB82          JMP    0634

06B0 55            PUSH   BP
06B1 32E4          XOR    AH,AH
06B3 8A5600        MOV    DL,[BP+00]
06B6 CD13          INT    13

06B8 5D            POP    BP
06B9 EB9E          JMP    0659

06BB 813EFE7D55AA  CMP    WORD PTR [7DFE],AA55
06C1 756E          JNZ    0731           ; If we don't see it, Error!
                                         ; -> "Missing operating system"
06C3 FF7600        PUSH   WORD PTR [BP+00] ; Popped into DL again at 0727
                                           ; (contains 80h if 1st drive).


; =====================================================================
;   All of the code from 06C6 through 0726 is related to discovering if
; TPM version 1.2 interface support is operational on the system, since
; it could be used by  BitLocker for validating the integrity of a PC's
; early startup components before allowing the OS to boot. The spec for
; the TPM code below states "There MUST be no requirement placed on the
; A20 state on entry to these INT 1Ah functions." (p.83) We assume here
; Microsoft understood this to mean access to memory over 1 MiB must be
; made available before entering any of the TPM's INT 1Ah functions.

; The following code is actually a method for gaining access to Memory
; locations above 1 MiB (also known as enabling the A20 address line).
;
; Each address line allows the CPU to access ( 2 ^ n ) bytes of memory:
; A0 through A15 can give access to 2^16 = 64 KiB. The A20 line allows
; a jump from 2^20 (1 MiB) to 2^21 = 2 MiB in accessible memory.  But
; our computers are constructed such that simply enabling the A20 line
; also allows access to any available memory over 1 MiB if both the CPU
; and code can handle it (once outside of "Real Mode"). Note: With only
; a few minor differences, this code at 06C6-06E1 and the Subroutine at
; 0756 ff. are the same as rather old sources we found on the Net.

06C6 E88D00        CALL   0756
06C9 7517          JNZ    06E2

06CB FA            CLI           ; Clear IF, so CPU ignores maskable interrupts.
06CC B0D1          MOV    AL,D1
06CE E664          OUT    64,AL
06D0 E88300        CALL   0756

06D3 B0DF          MOV    AL,DF
06D5 E660          OUT    60,AL
06D7 E87C00        CALL   0756

06DA B0FF          MOV    AL,FF
06DC E664          OUT    64,AL
06DE E87500        CALL   0756
06E1 FB            STI           ; Set IF, so CPU can respond to maskable interrupts
                                 ; again, after the next instruction is executed.

; Comments below checked with the document, "TCG PC Client Specific
; Implementation Specification For Conventional BIOS" (Version 1.20
; FINAL/Revision 1.00/July 13, 2005/For TPM Family 1.2; Level 2), §
; 12.5, pages 85 ff.  TCG and "TCG BIOS DOS Test Tool" (MSDN).

06E2 B800BB        MOV    AX,BB00   ; With AH = BBh and AL = 00h
06E5 CD1A          INT    1A        ; Int 1A ->  TCG_StatusCheck

06E7 6623C0      * AND    EAX,EAX  ;/   If EAX does not equal zero,
06EA 753B          JNZ    0727     ;\ then no BIOS support for TCG.

06EC 6681FB544350+  * CMP  EBX,41504354   ; EBX must also return ..
                                         ; the numerical equivalent
; of the ASCII character string "TCPA" ("54 43 50 41") as a further
; check. (Note: Since hex numbers are stored in reverse order on PC
; media or in Memory, a TPM BIOS would put 41504354h in EBX.)

06F3 7532             JNZ    0727       ;  If not, exit TCG code.
```

```
06F5 81F90201         CMP    CX,0102    ; Version 1.2 or higher ?
06F9 722C             JB     0727       ;  If not, exit TCG code.

; If TPM 1.2 found, perform a: "TCG_CompactHashLogExtendEvent".

06FB 666807BB0000   * PUSH   0000BB07   ; Setup for INT 1Ah AH = BB,
                                        ; AL = 07h command (p.94 f).
0701 666800020000   * PUSH   00000200   ;
0707 666808000000   * PUSH   00000008   ;
070D 6653           * PUSH   EBX        ;
070F 6653           * PUSH   EBX        ;
0711 6655           * PUSH   EBP        ;
0713 666800000000   * PUSH   00000000   ;
0719 6668007C0000   * PUSH   00007C00   ;
071F 6661           * POPAD             ;
0721 680000         * PUSH   0000       ;
0724 07               POP    ES         ;
0725 CD1A             INT    1A

; On return, "(EAX) = Return Code as defined in Section 12.3" and
;            "(EDX) = Event number of the event that was logged".
; =====================================================================

0727 5A            POP     DX          ; From [BP+00] at 06C3; often 80h.
0728 32F6          XOR     DH,DH       ; (Only DL matters)
072A EA007C0000    JMP     0000:7C00   ; Jump to Volume Boot Record code
                                       ; loaded into Memory by this MBR.

072F CD18          INT     18          ; Is this instruction here to meet
                                       ; some specification of TPM v 1.2 ?
; The usual 'INT18 if no disk found' is in the code above at 0632.

; Note: When the last character of any Error Message has been displayed, the
; instructions at offsets 0748, 0753 and 0754 lock computer's execution into
; a never ending loop! You must reboot the machine.  INT 10, Function 0Eh
; (Teletype Output) is used to display each character of these error messages.

0731 A0B707        MOV     AL,[07B7]   ; ([7B7] -> 9A) + 700 = 79A h
0734 EB08          JMP     073E        ; Displays: "Missing operating system"
0736 A0B607        MOV     AL,[07B6]   ; ([7B6] -> 7B) + 700 = 77B h
0739 EB03          JMP     073E        ; Displays: "Error loading operating
                                       ;                              system"
073B A0B507        MOV     AL,[07B5]   ; ([7B5] -> 63) + 700 = 763 h
                                       ; which will display: "Invalid
                                       ;                     partition table"
073E 32E4          XOR     AH,AH       ; Zero-out AH.
0740 050007        ADD     AX,0700     ; Add 700h to offsets from above.
0743 8BF0          MOV     SI,AX       ; Offset of message -> Source Index Reg.
0745 AC            LODSB               ; Load character into AL from [SI].

0746 3C00          CMP     AL,00       ;/ Have we reached end of message
                                       ;|   marker?(00) If so, then
0748 7409          JZ      0753        ;

074A BB0700        MOV     BX,0007     ; Display page 0, normal white on black
                                       ;   characters.
074D B40E          MOV     AH,0E       ;/ Teletype Output.. displays only
074F CD10          INT     10          ;\   one character at a time.
0751 EBF2          JMP     0745        ; Go back for another character...

0753 F4            HLT
0754 EBFD          JMP     0753        ; And just in case an NMI occurs,
                                       ; we jump right back to HLT again!

; -----------------------------------------------------------------------
;   SUBROUTINE - Part of A20 Line Enablement code (see 06C6 ff. above);
;                This routine checks/waits for access to KB controller.
; -----------------------------------------------------------------------
0756 2BC9          SUB     CX,CX      ; Sets CX = 0.
0758 E464          IN      AL,64      ; Check port 64h.
075A EB00          JMP     075C       ; Seems odd, but this is how it's done.
075C 2402          AND     AL,02      ; Test for only 'Bit 1' *not* set.
075E E0F8          LOOPNE  0758       ; Continue to check (loop) until
                                      ; CX = 0 (and ZF=1); it's ready.
0760 2402          AND     AL,02
0762 C3            RET
```

Location of *English* Error Messages and Message Offsets in Memory

```
                3  4  5  6  7  8  9  A  B  C  D  E  F
0763           49 6E 76 61 6C 69 64 20 70 61 72 74 69  Invalid parti
0770  74 69 6F 6E 20 74 61 62 6C 65 00 45 72 72 6F 72  tion Table.Error
0780  20 6C 6F 61 64 69 6E 67 20 6F 70 65 72 61 74 69   loading operati
0790  6E 67 20 73 79 73 74 65 6D 00 4D 69 73 73 69 6E  ng system.Missin
07A0  67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74  g operating syst
```

```
07B0  65 6D 00 00 00 63 7B 9A                         em...c{.
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

Location of Sample Disk Signature and Partition Table in Memory

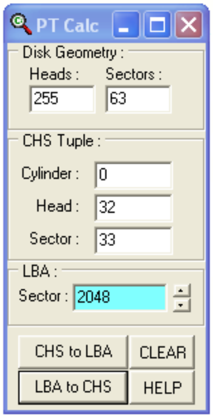
```
                               8  9  A  B  C  D  E  F
07B8                          D4 34 A0 2E 00 00 80 20         .4.....
07C0  21 00 07 DF 13 0C 00 08 00 00 00 20 03 00 00 DF  !.......... ....
07D0  14 0C 07 FE FF FF 00 28 03 00 hh hh hh hh 00 00  ................
07E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07F0  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
```

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

```
  Partition    Active  Starting Loc   Ending Loc    Relative   Number of
    Type        Boot   Cyl Head Sec  Cyl  Head Sec   sectors     sectors
-------------  ------  ------------  -------------  --------   ---------
     NTFS       Yes      0   32  33    12  223  19      2048      204800
     NTFS        No     12  223  20  1023  254  63    206848   nnnnnnnnn

[Note: Cylinders and heads start counting at ZERO; sectors at 1. So, a CHS of
 0,32,33 gives us 32 'full' heads of 63 sectors each, or 32 x 63 = 2016; the
 33rd head including only up to sector 33, for a total of: 2016 + 33 = 2049, or
 Absolute Sector 2048 as the sector where Win7 partitions begin on the disk;
 with 2048 sectors preceding.

 12,223,20 gives us 12 'full' cylinders of 255 x 63 = 16065 sectors each, or
 12 x 16065 = 192780 sectors. The 13th cylinder being only 223 'full' heads or
 14049; the 224th head including only up to sector 20, for a total of:
 192780 + 14049 + 20 = 206849, or Absolute Sector 206848 as the sector where
 this disk's second primary partition begins.]
```

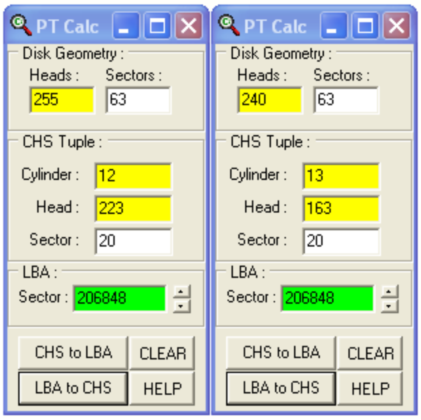
Three views of PT Calc showing: **1)** The CHS values for a Vista or Windows 7 OS install's first partition (at Absolute Sector **2048**) for either a typical desktop or notebook disk, and **2)** How disks which are assigned a different number of Heads in their BIOS (**255** and **240** are shown) can have different CHS values for exactly the same sector (**206848**).

(1)



Disk Geometry:
Heads: 255
Sectors: 63
CHS Tuple:
Cylinder: 0
Head: 32
Sector: 33
LBA:
Sector: 2048

(2)



Disk Geometry:
Heads: 255
Sectors: 63
CHS Tuple:
Cylinder: 12
Head: 223
Sector: 20
LBA:
Sector: 206848

Disk Geometry:
Heads: 240
Sectors: 63
CHS Tuple:
Cylinder: 13
Head: 163
Sector: 20
LBA:
Sector: 206848

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 see an error message such as "**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**.)

First Published: **12 MAR 2011** (12.03.11).

Updated: 22 MAR 2011 (22.03.2011); 8 MAY 2011 (08.05.2011); 14 MAY 2011 (14.05.2011); 28 MAR 2012 (28.03.2012); 18 APR 2012 (18.04.2012); 12 MAY 2013 (12.05.2013); 29 JUN 2013 (29.06.2013); 11 JUL 2013 (11.07.2013); 17 FEB 2015 (17.02.2015); 1 MAY 2015 (01.05.2015); 1 JUN 2015 (01.06.2015); 21 JUN 2015 (21.06.2015).

Last Update: **10 AUG 2015**. (10.09.2015)

You can write to me using this: online reply form. (It opens in a new window.)

The Starman's FREE TOOLS Page ☞

☜ MBR and Boot Records Index

☜ **The Starman's Realm Index Page**