

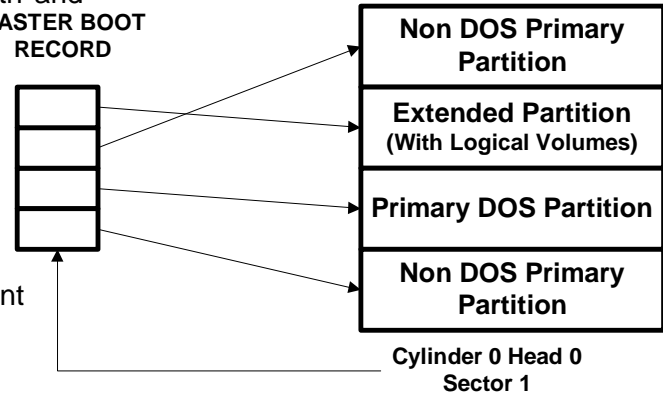
The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

INTRODUCTION

The two sectors critical to starting your computer are the *master boot record* (MBR), which is always located at sector 1 of cylinder 0, head 0, the first sector of a hard disk, and the *boot sector*, which resides at sector 1 of each volume. These sectors contain both executable code and the data required to run the code. This white paper will take an in-depth and detailed look at both of these data structures and how they affect the boot process and data storage integrity of your computer. Both the master boot record and the boot sector will be broken down to their most basic components. Each component will be explained in terms of what it is and how it relates to the other components of the structure. I challenge anyone to find a more detailed description or discussion of these important system structures.



MASTER BOOT RECORD

The MBR, the most important data structure on the disk, is created when the disk is partitioned. During partitioning, no matter what file system is specified or operating system used, the partitioning software (such as *fdisk*) writes a special boot program and partition table to the first sector of the disk called the **Master Boot Record** or **MBR**. Sector. The purpose of the MBR is to provide the system BIOS with the information it needs to locate and load the computer operating system. Recall, the **ROM BIOS Int19** routine, when it is ready to load the operating system, searches for the **Master Boot Record** at *cylinder 0, head 0, sector 1 (the very first sector)* of the hard disk, and loads it into memory at address 0000:7C00.

Now, the MBR (512 bytes in total length) is divided into 3 parts:

1. The first part, which is 446 bytes, contains the **Boot Code** that actually loads the operating system.
2. The second part, which is 64 bytes long, contains the data which defines where the various partitions on your hard drive start and stop, and what operating system they are used by. This is known as the **Partition Table**. Each partition's information occupies 16 bytes for a maximum total of 4 partitions.

BASIC MBR (512 Bytes Total)

Boot Code 0x0000 to 0x01BE 446 Bytes
Partition Table 0x01BE to 0x1FE (64 Bytes)
Signature Word/Magic Number (set to 0xAA55) 0x01FE to 0x1FF (2 Bytes)

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

- The last part, which is two bytes long and often referred to as the **Magic Number**, contains the characters 55 AA, which indicates that the partition table is executable. If the last two bytes are anything but these values, the BIOS will not load the MBR.

BOOT CODE

The 446 byte boot code section of the MBR actually contains four distinct areas within itself. These are a **Near Jump** (three bytes), the physical drive **Disk Parameters** (59 bytes), the **Boot Program Code** (378 bytes) itself, and a **Disk Signature** (six bytes).

The **Disk Signature**, a unique number at offset 0x01B8, identifies the disk to the operating system. For instance, Windows 2000 uses the disk signature as an index to store and retrieve information about the disk in the registry subkey HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices.

The master boot code performs the following activities:

- Scans the partition table for the active partition.
- Finds the starting sector of the active partition.
- Loads a copy of the boot sector from the active partition into memory.
- Transfers control to the executable code in the boot sector.

If the master boot code cannot complete these functions, the system displays one of the following error messages:

- Invalid partition table
- Error loading operating system
- Missing operating system

Note: There is no MBR on a floppy disk. The first sector on a floppy disk is the boot sector. Although every hard disk contains an MBR, the master boot code is used only if the disk contains the active, primary partition.

BOOT CODE (446 Bytes Total)

Near Jump Into Boot Program Code 0x0000 to 0x0003 3 Bytes
Disk Parameters 0x0003 to 0x003E 59 Bytes
Boot Program Code 0x003E to 0x01B8 378 Bytes
Disk Signature 0x01B8 to 0x01BE 6 Bytes

PARTITION TABLE

The partition table, a 64-byte data structure used to identify the type and location of partitions on a hard disk, conforms to a standard layout independent of the operating system. Each partition table entry is 16

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

bytes long, with a maximum of four entries. Each entry starts at a predetermined offset from the beginning of the sector, as follows:

- Partition 1 0x01BE (446)
- Partition 2 0x01CE (462)
- Partition 3 0x01DE (478)
- Partition 4 0x01EE (494)

The following example shows a partial printout of an MBR revealing the partition table from a computer with three partitions. When there are fewer than four partitions on a disk, the remaining partition table fields are set to the value 0:

```

000001B0: 80 01 ..
000001C0: 01 00 07 FE BF 09 3F 00 - 00 00 4B F5 7F 00 00 00 .....?...K....
000001D0: 81 0A 07 FE FF FF 8A F5 - 7F 00 3D 26 9C 00 00 00 .....=&....
000001E0: C1 FF 05 FE FF FF C7 1B - 1C 01 D6 96 92 00 00 00
.....
000001F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 .....

```

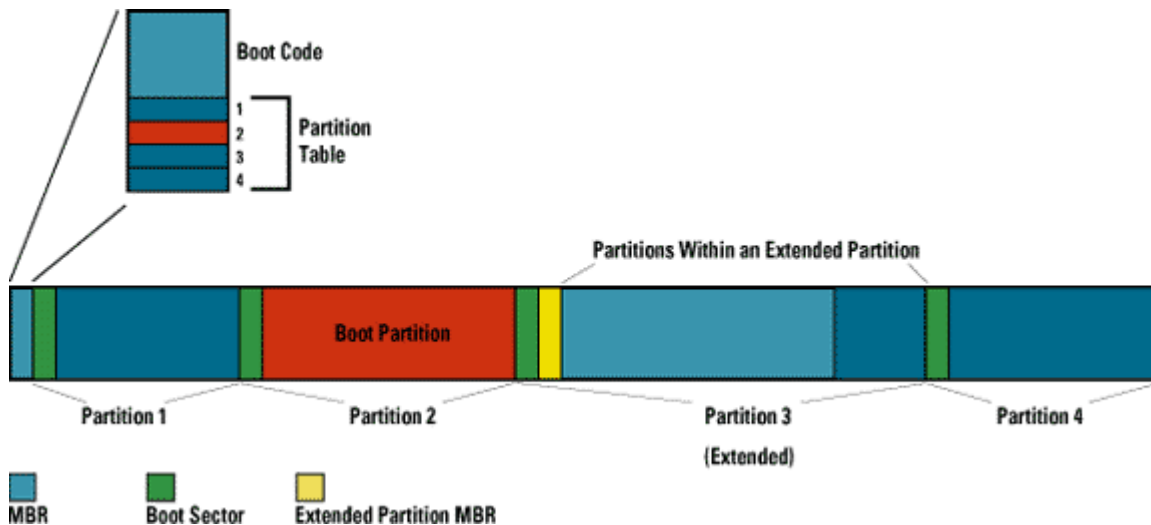


Table 1 describes the fields in each entry in the partition table. The sample values correspond to the first partition table entry shown in the preceding example. The Byte Offset values correspond to the addresses of the first partition table entry.

TABLE 1: PARTITION TABLE FIELDS

Byte Offset	Field Length	Sample Value	Field Name and Definition
0x01BE	BYTE	0x80	Boot Indicator: Indicates whether the volume is the active partition. Legal values include 00 (do not use for booting) and 80 (active partition).
0x01BF	BYTE	0x01	Starting Head

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

0x01C0	6 bits	0x01	Starting Sector: Only bits 0-5 are used. The upper two bits are used by the Starting Cylinder field.
0x01C2	10 bits	0x00	Starting Cylinder: Uses 1 byte in addition to the upper 2 bits from the Starting Sector field to make up the cylinder value. The Starting Cylinder is a 10 bit number with a maximum value of 1023.
0x01C2	BYTE	0x07	System ID: Defines the volume type (see Table 2).
0x01C3	BYTE	0xFE	Ending Head
0x01C4	6 bits	0xBF	Ending Sector: Only bits 0-5 are used. The upper two bits are used by the Ending Cylinder field.
0x01C5	10 bits	0x09	Ending Cylinder: Uses one byte in addition to the upper two bits from the Ending Sector field to make up the cylinder value. The Ending Cylinder is a 10 bit number with a maximum value of 1023.
0x01C6	DWORD	0x3F000000	Relative Sectors: The offset from the beginning of the disk to the beginning of the volume, counting by sectors.
0x01CA	DWORD	0x4BF57F00	Total Sectors: The total number of sectors in the volume.

Numbers larger than one byte are stored in little endian format, or reverse-byte ordering. Little endian format is a method of storing a number so that the least significant byte appears first in the hexadecimal number notation. For example, the sample value for the Relative Sectors field in the previous table, 0x3F000000, is a little endian representation of 0x0000003F. The decimal equivalent of this little endian number is 63.

Boot Indicator Field

The first element of the partition table, the **Boot Indicator** field, indicates whether or not the volume is the active partition. Only one primary partition on the disk can have this field set. It is possible to have different operating systems and different file systems on different volumes. By using disk configuration tools such as the Windows 2000-based Disk Management or the MS-DOS-based Fdisk to designate a primary partition as active, the **Boot Indicator** field for that partition is set in the partition table.

System ID Field

Another element of the partition table is the **System ID** field. It defines which file system, such as FAT16, FAT32, or NTFS, was used to format the volume. The **System ID** field also identifies an extended

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

partition, if one is defined. Windows 2000 uses the **System ID** field to determine which file system device drivers to load during startup. Table 2 identifies the values for the **System ID** field.

TABLE 2: SYSTEM ID VALUES

Partition Type	ID Value
0x01	FAT12 primary partition or logical drive (fewer than 32, 680 sectors in the volume).
0x04	FAT16 partition or logical drive (32, 680 to 65, 535 sectors).
0x05	Extended Partition
0x06	BIGDOS FAT16 partition or logical drive (33 MB to 4 GB).
0x07	Installable File System (NTFS) partition or logical drive.
0x0B	FAT32 partition or logical drive.
0x0C	FAT32 partition or logical drive using BIOS INT 13h extensions.
0x0E	BIGDOS FAT16 partition or logical drive using BIOS INT 13h extensions.
0x0F	Extended partition using BIOS INT 13h extensions.
0x12	EISA partition
0x42	Dynamic disk volume
0x86	Legacy FT FAT16 disk.
0x87	Legacy FT NTFS disk.
0x8B	Legacy FT volume formatted with FAT32.
0x8C	Legacy FT volume using using BIOS INT 13h extensions formatted with FAT32.

Note: MS-DOS can only access volumes that have a **System ID** value of 0x01, 0x04, 0x05, or 0x06. However, you can delete volumes that have the other values listed in Table 2 using MS-DOS tools such as Fdisk. If you use a low-level disk editor, such as DiskProbe or Disk Doctor, you can read and write to any sector, including ones that are in NTFS volumes.

Starting and Ending Cylinder, Head, and Sector Fields

The **Starting** and **Ending Cylinder, Head,** and **Sector** fields (collectively known as the **CHS** fields) are additional elements of the partition table. These fields are essential for starting the computer. The master boot code uses these fields to find and load the boot sector of the active partition. The **Starting CHS** fields for non-active partitions point to the boot sectors of

Revised January 11, 2009

PARTITION TABLE (64 Bytes Total)

Partition 1 0x01BE to 0x01CE 16 Bytes
Partition 2 0x01CE to 0x01DE 16 Bytes
Partition 3 0x01DE to 0x01EE 16 Bytes
Partition 4 0x01EE to 0x01FE 16 Bytes

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

the remaining primary partitions and the EBR of the first logical drive in the extended partition.

Knowing the starting sector of an extended partition is very important for low-level disk troubleshooting. If your disk fails, you need to work with the partition starting point (among other factors) to retrieve stored data.

The **Ending Cylinder** field in the partition table is 10 bits long, which limits the number of cylinders that can be described in the partition table to a range of 0–1,023. The **Starting Head** and **Ending Head** fields are each one byte long, which limits the field range to 0–255. The **Starting Sector** and **Ending Sector** fields are each six bits long, which limits the range of these fields to 0–63. However, the enumeration of sectors starts at 1 (not 0, as for other fields), so the maximum number of sectors per track is 63.

Because all hard disks are low-level formatted with a standard 512-byte sector, the maximum disk capacity described by the partition table is calculated as follows:

Maximum capacity = sector size x cylinders (10 bits) x heads (8 bits) x sectors per track (6 bits)

Using the maximum possible values yields:

512 x 1024 x 256 x 63 (or 512 x 2²⁴) = 8,455,716,864 bytes or 7.8 GB

The calculation results in a maximum capacity of slightly less than 8 gigabytes (GB). Before BIOS INT 13h extensions drive geometry translation (also known as logical block addressing, or LBA) were introduced, the active, primary partition could not exceed 7.8 GB, regardless of the file system used.

Important: When using the standard 512-byte sector, the maximum cluster size that you can use for FAT16 volumes while running Windows 2000 is 64 kilobytes (KB). Therefore, the maximum size for a FAT16 volume is 4 GB.

If you use a multiple-boot configuration with Windows 95, Windows 98, or MS-DOS, FAT16 volumes must be limited to 2 GB to be accessed from these operating systems. In addition, a Macintosh computer that accesses volumes on a computer running Windows 2000 cannot access a FAT16 volume that is larger than 2 GB. If you try to use a FAT16 volume larger than 2 GB when running MS-DOS, Windows 95, or Windows 98, or try to access such a volume from a Macintosh computer, you might get a message that zero bytes are available.

The maximum FAT16 volume size that you can use on a computer depends on the disk geometry and the maximum values that fit in the partition table entry

Revised January 11, 2009

STRUCTURE OF PARTITION TABLE ENTRY (16 Bytes Total)

BOOT Boot Flag: 0 = not active or 0x080 active 1 Byte
HD Begin: head number 1 Byte
SEC CYL Begin: sector & cylinder number of boot sector 2 Bytes
SYS System Code: OS specific code 1 Byte
HD End: head number 1 Byte
SEC CYL End: sector & cylinder number of last sector 2 Bytes
low byte - high byte relative sector number of start sector 4 Bytes
low byte - high byte number of sectors in partition 4 Bytes

The Master Boot Record The Partition Table & The Boot Sector

By Mark E. Donaldson

fields. The number of cylinders in both cases is 1,024 (0–1,023). When a primary partition or logical drive extends beyond the 1,023rd cylinder, all fields described in this section contain the maximum values.

Relative Sectors and Total Sectors Fields

The **Relative Sectors** field represents the offset from the beginning of the disk to the beginning of the volume, counting by sectors, for the volume described by the partition table entry. The **Total Sectors** field represents the total number of sectors in the volume.

Using the **Relative Sectors** and **Total Sectors** fields (resulting in a 32-bit number) provides eight more bits than the CHS scheme to represent the total number of sectors. This allows partitions containing up to 2^{32} sectors to be defined. With a standard sector size of 512 bytes, the 32 bits used to represent the **Relative Sectors** and **Total Sectors** fields translates into a maximum partition size of 2 terabytes (or 2,199,023,255,552 bytes).

Windows 2000 uses the fields in the partition table entries to access all partitions. A partition that is formatted while Windows 2000 is running puts data into the **Starting** and **Ending CHS** fields to have compatibility with MS-DOS, Windows 95, and Windows 98, and to maintain compatibility with the BIOS INT 13h for startup.

Extended Boot Record

A special partition type, extended partition, or Extended Boot Record (EBR), contains another MBR with its own partition table. By using extended partitions, operating systems overcome the apparent limit of four partitions per disk, as an MBR's partition table defines. ***In general, the recursion that extended partitions permit can continue indefinitely, which means that no upper limit exists to the number of possible partitions on a disk.***

An EBR, which consists of an extended partition table and the signature word for the sector, exists for each logical drive in the extended partition. It contains the only information on the first side of the first cylinder of each logical drive in the extended partition. The boot sector in a logical drive is usually located at either Relative Sector 32 or 63. However, if there is no extended partition on a disk, there are no EBRs and no logical drives.

The first entry in an extended partition table for the first logical drive points to its own boot sector. The second entry points to the EBR of the next logical drive. If no further logical drives exist, the second entry is not used and is recorded as a series of zeroes. If there are additional logical drives, the first entry of the extended partition table for the second logical drive points to its own boot sector. The second entry of the extended partition table for the second logical drive points to the EBR of the next logical drive. The third and fourth entries of an extended partition table are never used.

The **Relative Sectors** field in an extended partition table entry shows the number of bytes that are offset from the beginning of the extended partition to the first sector in the logical drive. The number in the **Total Sectors** field refers to the number of sectors that make up the logical drive. The value of the **Total Sectors** field equals the number of sectors from the boot sector defined by the extended partition table entry to the end of the logical drive.

The Master Boot Record The Partition Table & The Boot Sector

By Mark E. Donaldson

BOOT SECTOR

The boot sector, located at sector 1 of each volume, is a critical disk structure for starting your computer. It contains executable code and data required by the code, including information that the file system uses to access the volume. The boot sector is created when you format a volume. At the end of the boot sector is a two-byte structure called a signature word or end of sector marker, which is always set to 0x55AA. On computers running Windows 2000, the boot sector on the active partition loads into memory and starts Ntldr, which loads the operating system.

The Windows 2000 boot sector consists of the following elements:

- An x86-based CPU jump instruction.
- The original equipment manufacturer identification (OEM ID).
- The BIOS parameter block (BPB), a data structure.
- The extended BPB.
- The executable boot code (or bootstrap code) that starts the operating system.

Boot Sector Startup Processes

Computers use the boot sector to run instructions during startup. The initial startup process is summarized in the following steps:

1. The system BIOS and the CPU initiate the power-on self test (POST).
2. The BIOS searches for a boot device (typically a disk).
3. The BIOS loads the first physical sector of the boot device into memory and transfers CPU execution to that memory address.

If the boot device is on a hard disk, the BIOS loads the MBR. The master boot code in the MBR loads the boot sector of the active partition, and transfers CPU execution to that memory address. On computers that are running Windows 2000, the executable boot code in the boot sector finds Ntldr, loads it into memory, and transfers execution to that file.

If there is a floppy disk in drive A, the system BIOS loads the first sector (the boot sector) of the disk into memory. If the disk is startable, the boot sector loads into memory and uses the executable boot code to transfer CPU execution to Io.sys, a core MS-DOS operating system file. If the floppy disk is not bootable, the executable boot code displays an error message such as:

- Non-System disk or disk error
- Replace and press any key when ready

Note: This error will not appear on normally functioning systems that are configured to look for the startup files on drive C first. On many computers, an option in the CMOS setup program allows the user

The Master Boot Record The Partition Table & The Boot Sector

By Mark E. Donaldson

to set the sequence of installed disks that the system searches for the startup files. If you get similar errors when trying to start the computer from the hard disk, the boot sector might be corrupted.

Initially, the startup process is independent of disk format and operating system. The unique characteristics of operating and file systems become important when the boot sector's executable boot code starts.

Components of a Boot Sector

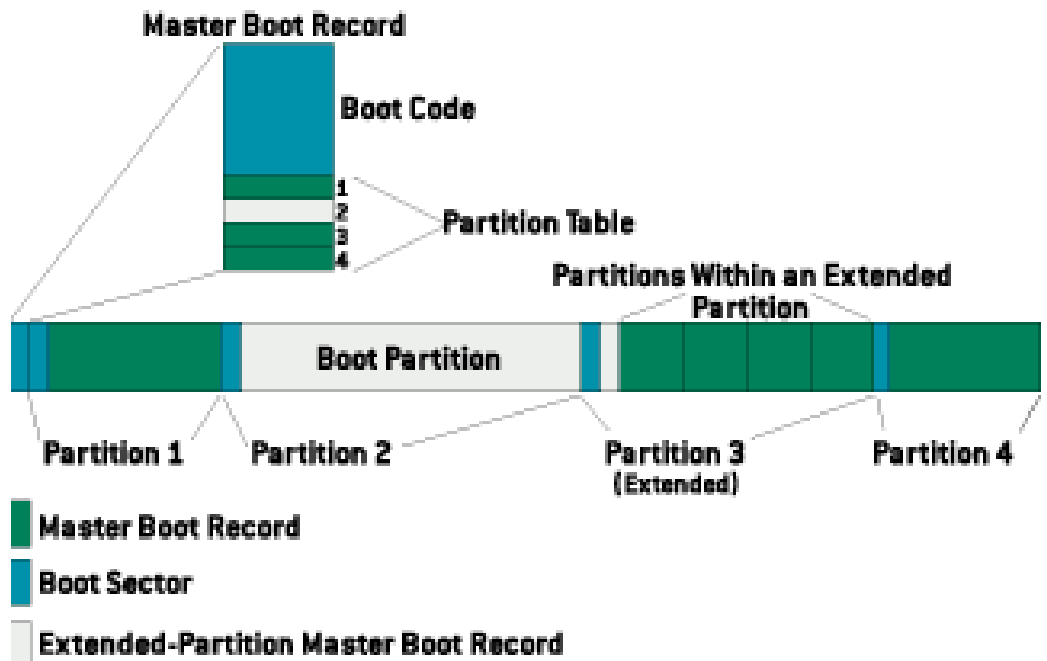
The MBR transfers CPU execution to the boot sector, so the first three bytes of the boot sector must be valid, executable x86-based CPU instructions. This includes a jump instruction that skips the next several nonexecutable bytes.

Following the jump instruction is the 8-byte OEM ID, a string of characters that identifies the name and version number of the operating system that formatted the volume. To preserve compatibility with MS-DOS, Windows 2000 records "MSDOS5.0" in this field on FAT16 and FAT32 disks. On NTFS disks, Windows 2000 records "NTFS."

Following the OEM ID is the BPB, which provides information that enables the executable boot code to locate Ntldr. The BPB always starts at the same offset, so standard parameters are in a known location. Disk size and geometry variables are encapsulated in the BPB.

Because the first part of the boot sector is an x86 jump instruction, the BPB can be extended in the future by appending new information at the end. The jump instruction needs only a minor adjustment to accommodate this change.

The BPB is stored in a packed (unaligned) format.



WINDOWS NTLDR

The Windows 2000 bootstrap loader, NTLDR, can load any version of NT or Windows 2000 from any drive or partition on the local system (except some flavors of removable media drive), but it can load only one alternate operating system. It does so by storing the boot sector for the alternate OS in a file called BOOTSECT.DOS at the root of the boot partition. If the alternate OS is selected from the Windows 2000 BOOT menu, NTLDR shifts the processor back to Real mode, loads the image from BOOTSECT.DOS

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

into memory at 0x700h just as if it had been loaded by a standard INT13 call, and then turns control over to the executable code in the image.

If you need to boot more than two operating systems and one of them is not a Windows OS, use a partition manager, such as Partition Magic, System Commander, or Linux Loader (LILO). If you want to configure a triple-boot machine to load DOS/Windows, Windows 9x, and Windows 2000, you can create a triple-boot menu with two alternate BOOTSECT files by layering your installations. The basic steps are as follows:

1. Install DOS.
2. Install Windows 2000. This saves the DOS boot sector to BOOTSECT.DOS.
3. Toggle the read-only, hidden, and system attributes on BOOTSECT.DOS.
4. Rename BOOTSECT.DOS to BOOTSECT.622, for DOS version 6.22.
5. Install Windows 9x. This overwrites the Windows 2000 boot sector with its own boot sector.
6. Run the Windows 2000 Repair Console. Use it to restore a Windows 2000 boot sector, and save the existing boot sector as BOOTSECT.DOS.
7. Toggle the read-only attribute on BOOT.INI.
8. Edit BOOT.INI to add the following lines:

```
[Operating Systems]
c:\bootsect.622="DOS" /win95dos
c:\bootsect.dos="Win9x" /win95
```

A CLOSER LOOK AT THE DOS FLOPPY DISK BOOT SECTOR

The boot sector of a floppy disk is located at cylinder 0, head 0, sector 1. This sector is created by a floppy disk formatting program, such as the DOS FORMAT program. The boot sector of a FAT hard disk partition has a similar layout and function. Basically a bootable FAT hard disk partition looks like a big floppy during the early stages of the system's boot processing.

At the completion of your system's Power On Self Test (POST), INT 19 is called. Usually INT 19 tries to read a boot sector from the first floppy drive. If a boot sector is found on the floppy disk, the that boot sector is read into memory at location 0000:7C00 and INT 19 jumps to memory location 0000:7C00. However, if no boot sector is found on the first floppy drive, INT 19 tries to read the MBR from the first hard drive. If an MBR is found it is read into memory at location 0000:7c00 and INT 19 jumps to memory location 0000:7c00. The small program in the MBR will attempt to locate an active (bootable) partition in its partition table. If such a partition is found, the boot sector of that partition is read into memory at location 0000:7C00 and the MBR program jumps to memory location 0000:7C00. Each operating system has its own boot sector format. The small program in the boot sector must locate the first part of

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

the operating system's kernel loader program (or perhaps the kernel itself or perhaps a "boot manager program") and read that into memory.

INT 19 is also called when the CTRL-ALT-DEL keys are used. On most systems, CTRL-ALT-DEL causes an short version of the POST to be executed before INT 19 is called.

- The BIOS Parameter Block (BPB) starts at offset 0.
- The boot sector program starts at offset 3e.
- The messages issued by this program start at offset 19e.
- The DOS hidden file names start at offset 1e6.
- The boot sector signature is at offset 1fe.

Here is a summary of what all this does:

1. Copy Diskette Parameter Table which is pointed to by INT 1E.
2. Alter the copy of the Diskette Parameter Table.
3. Alter INT 1E to point to altered Diskette Parameter Table.
4. Do INT 13 AH=00, disk reset call.
5. Compute sector address of root directory.
6. Read first sector of root directory into 0000:0500.
7. Confirm that first two directory entries are for IO.SYS and MSDOS.SYS.
8. Read first 3 sectors of IO.SYS into 0000:0700 (or 0070:0000).
9. Leave some information in the registers and jump to IO.SYS at 0070:0000.

This program uses the CHS based INT 13H AH=02 to read the FAT root directory and to read the IO.SYS file. If the drive is >528MB, this CHS must be a translated CHS. Except for internal computations no addresses in LBA form are used, another reason why LBA doesn't solve the >528MB problem. Here is the entire sector in hex and ascii.

```

OFFSET 0 1 2 3 4 5 6 7 8 9 A B C D E F *0123456789ABCDEF*
000000 eb3c904d 53444f53 352e3000 02010100 *.<.MSDOS5.0.....*
000010 02e00040 0bf00900 12000200 00000000 *...@.....*
000020 00000000 0000295a 5418264e 4f204e41 *.....)ZT.&NO NA*
000030 4d452020 20204641 54313220 2020fa33 *ME FAT12 .3*
000040 c08ed0bc 007c1607 bb780036 c5371e56 *.....|...x.6.7.V*
000050 1653bf3e 7cb90b00 fcf3a406 1fc645fe *.S.>|.....E.*
000060 0f8b0e18 7c884df9 894702c7 073e7cfb *....|.M..G...>|. *
000070 cd137279 33c03906 137c7408 8b0e137c *..ry3.9..|t....|*
000080 890e207c a0107cf7 26167c03 061c7c13 *..|..|.&.|...|. *
000090 161e7c03 060e7c83 d200a350 7c891652 *..|...|....P|..R*
0000a0 7ca3497c 89164b7c b82000f7 26117c8b *|.I|..K|. ..&|. *
0000b0 1e0b7c03 c348f7f3 0106497c 83164b7c *..|..H....I|..K|*
0000c0 00bb0005 8b16527c a1507ce8 9200721d *.....R|.P|...r.*
0000d0 b001e8ac 0072168b fbb90b00 bee67df3 *.....r.....}. *
0000e0 a6750a8d 7f20b90b 00f3a674 18be9e7d *.u... ..t...}*
0000f0 e85f0033 c0cd165e 1f8f048f 4402cd19 *_.3...^....D...*
000100 585858eb e88b471a 48488a1e 0d7c32ff *XXX...G.HH...|2.*
000110 f7e30306 497c1316 4b7cbb00 07b90300 *....I|..K|.....*
000120 505251e8 3a0072d8 b001e854 00595a58 *PRQ.:.r....T.YZX*
000130 72bb0501 0083d200 031e0b7c e2e28a2e *r.....|.....*

```

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

```

000140 157c8a16 247c8b1e 497ca14b 7cea0000 *.|..$|..I|.K|...*
000150 7000ac0a c07429b4 0ebb0700 cd10ebf2 *p....t).....*
000160 3b16187c 7319f736 187cfec2 88164f7c *;...|s..6.|...O|*
000170 33d2f736 1a7c8816 257ca34d 7cf8c3f9 *3..6.|..%|.M|...*
000180 c3b4028b 164d7cb1 06d2e60a 364f7c8b *.....M|.....6O|. *
000190 ca86e98a 16247c8a 36257ccd 13c30d0a *.....$|.6%|.....*
0001a0 4e6f6e2d 53797374 656d2064 69736b20 *Non-System disk *
0001b0 6f722064 69736b20 6572726f 720d0a52 *or disk error..R*
0001c0 65706c61 63652061 6e642070 72657373 *eplace and press*
0001d0 20616e79 206b6579 20776865 6e207265 * any key when re*
0001e0 6164790d 0a00494f 20202020 20205359 *ady...IO      SY*
0001f0 534d5344 4f532020 20535953 000055aa *SMSDOS  SYS..U.*

```

The first 62 bytes of a boot sector are known as the BIOS Parameter Block (BPB). Here is the layout of the BPB fields and the values they are assigned in this boot sector:

```

db JMP instruction      at 7c00 size 2 = eb3c
db NOP instruction     7c02      1 90
db OEMname             7c03      8 'MSDOS5.0'
dw bytesPerSector      7c0b      2 0200
db sectPerCluster      7c0d      1 01
dw reservedSectors     7c0e      2 0001
db numFAT               7c10      1 02
dw numRootDirEntries   7c11      2 00e0
dw numSectors           7c13      2 0b40 (ignore numSectorsHuge)
db mediaType           7c15      1 f0
dw numFATsectors       7c16      2 0009
dw sectorsPerTrack     7c18      2 0012
dw numHeads            7c1a      2 0002
dd numHiddenSectors    7c1c      4 00000000
dd numSectorsHuge      7c20      4 00000000
db driveNum            7c24      1 00
db reserved            7c25      1 00
db signature           7c26      1 29
dd volumeID            7c27      4 5a541826
db volumeLabel         7c2b      11 'NO NAME '
db fileSysType         7c36      8 'FAT12 '

```

Here is the boot sector. The first 3 bytes of the BPB are JMP and NOP instructions.

```

0000:7C00 EB3C      JMP      START
0000:7C02 90          NOP

```

Here is the rest of the BPB.

```

0000:7C00 .....4d 53444f53 352e3000 02010100 * MSDOS5.0.....*
0000:7C10 02e00040 0bf00900 12000200 00000000 *...@.....*
0000:7C20 00000000 0000295a 5418264e 4f204e41 *.....)ZT.&NO NA*
0000:7C30 4d452020 20204641 54313220 2020.... *ME FAT12 *

```

The 11 bytes starting at 0000:7c3e are immediately overlaid by information copied from another part of memory. That information is the Diskette Parameter Table. This data is pointed to by INT 1E. This data is:

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

7c3e = Step rate and head unload time.
 7c3f = Head load time and DMA mode flag.
 7c40 = Delay for motor turn off.
 7c41 = Bytes per sector.
 7c42 = Sectors per track.
 7c43 = Intersector gap length.
 7c44 = Data length.
 7c45 = Intersector gap length during format.
 7c46 = Format byte value.
 7c47 = Head settling time.
 7c48 = Delay until motor at normal speed.

The 11 bytes starting at 0000:7c49 are also overlaid by the following data:

7c49 - 7c4c = diskette sector address (as LBA)
 of the data area.
 7c4d - 7c4e = cylinder number to read from.
 7c4f - 7c4f = sector number to read from.
 7c50 - 7c53 = diskette sector address (as LBA)
 of the root directory.

START:

START OF BOOT SECTOR PROGRAM

```
0000:7C3E FA          CLI          interrupts off
0000:7C3F 33C0       XOR          AX,AX      set AX to zero
0000:7C41 8ED0       MOV          SS,AX      SS is now zero
0000:7C43 BC007C     MOV          SP,7C00    SP is now 7c00
0000:7C46 16          PUSH        SS          also set ES
0000:7C47 07          POP         ES          to zero
```

The INT 1E vector is at 0000:0078.
 Get the address that the vector points to
 into the DS:SI registers.

```
0000:7C48 BB7800     MOV          BX,0078    BX is now 78
0000:7C4B 36          SS:
0000:7C4C C537       LDS          SI,[BX]    DS:SI is now [0:78]
0000:7C4E 1E          PUSH        DS          save DS:SI --
0000:7C4F 56          PUSH        SI          saves param tbl addr
0000:7C50 16          PUSH        SS          save SS:BX --
0000:7C51 53          PUSH        BX          saves INT 1E address
```

Move the diskette param table to 0000:7c3e.

```
0000:7C52 BF3E7C     MOV          DI,7C3E    DI is address of START
0000:7C55 B90B00     MOV          CX,000B    count is 11
0000:7C58 FC          CLD             clear direction
0000:7C59 F3          REPZ          move the diskette param
0000:7C5A A4          MOVSB         table to 0000:7c3e
0000:7C5B 06          PUSH        ES          also set DS
0000:7C5C 1F          POP         DS          to zero
```

Alter some of the diskette param table data.

```
0000:7C5D C645FE0F   MOV          BYTE PTR [DI-02],0F change head settle time
                                                at 0000:7c47
```

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

```

0000:7C61 8B0E187C  MOV  CX,[7C18]           sectors per track
0000:7C65 884DF9      MOV  [DI-07],CL         save at 0000:7c42

Change INT 1E so that it points to the
altered Diskette param table at 0000:7c3e.

0000:7C68 894702      MOV  [BX+02],AX         change INT 1E segment
0000:7C6B C7073E7C      MOV  WORD PTR [BX],7C3E change INT 1E offset

Call INT 13 with AX=0000, disk reset, so
that the new diskette param table is used.

0000:7C6F FB          STI          interrupts on
0000:7C70 CD13      INT 13      do diskette reset call
0000:7C72 7279      JB TALK     jmp if any error

Determine the starting sector address of
the root directory as an LBA.

0000:7C74 33C0      XOR  AX,AX      AX is now zero
0000:7C76 3906137C  CMP  [7C13],AX  number sectors zero?
0000:7C7A 7408      JZ  SMALL_DISK yes
0000:7C7C 8B0E137C  MOV  CX,[7C13]  number of sectors
0000:7C80 890E207C  MOV  [7C20],CX  save in huge num sects

SMALL_DISK:

0000:7C84 A0107C      MOV  AL,[7C10]   number of FAT tables
0000:7C87 F726167C  MUL  WORD PTR [7C16] number of fat sectors
0000:7C8B 03061C7C  ADD  AX,[7C1C]   number of hidden sectors
0000:7C8F 13161E7C  ADC  DX,[7C1E]   number of hidden sectors
0000:7C93 03060E7C  ADD  AX,[7C0E]   number of reserved sectors
0000:7C97 83D200      ADC  DX,+00      number of reserved sectors
0000:7C9A A3507C      MOV  [7C50],AX   save start addr
0000:7C9D 8916527C  MOV  [7C52],DX   of root dir (as LBA)
0000:7CA1 A3497C      MOV  [7C49],AX   save start addr
0000:7CA4 89164B7C  MOV  [7C4B],DX   of root dir (as LBA)

Determine sector address of first sector
in the data area as an LBA.

0000:7CA8 B82000      MOV  AX,0020     size of a dir entry (32)
0000:7CAB F726117C  MUL  WORD PTR [7C11] number of root dir entries
0000:7CAF 8B1E0B7C  MOV  BX,[7C0B]   bytes per sector
0000:7CB3 03C3      ADD  AX,BX
0000:7CB5 48          DEC  AX
0000:7CB6 F7F3      DIV  BX
0000:7CB8 0106497C  ADD  [7C49],AX   add to start addr
0000:7CBC 83164B7C00 ADC  WORD PTR [7C4B],+00 of root dir (as LBA)

Read the first root dir sector into 0000:0500.

0000:7CC1 BB0005      MOV  BX,0500     addr to read into
0000:7CC4 8B16527C  MOV  DX,[7C52]   get start of address
0000:7CC8 A1507C      MOV  AX,[7C50]   of root dir (as LBA)
0000:7CCB E89200      CALL CONVERT     call conversion routine

```

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

```

0000:7CCE 721D      JB      TALK      jmp is any error
0000:7CD0 B001      MOV     AL,01     read 1 sector
0000:7CD2 E8AC00    CALL   READ_SECTORS read 1st root dir sector
0000:7CD5 7216      JB      TALK      jmp if any error
0000:7CD7 8BFB      MOV     DI,BX     addr of 1st dir entry
0000:7CD9 B90B00    MOV     CX,000B   count is 11
0000:7CDC BEE67D    MOV     SI,7DE6   addr of file names
0000:7CDF F3        REPZ                    is this "IO.SYS"?
0000:7CE0 A6        CMPSB
0000:7CE1 750A      JNZ     TALK      no
0000:7CE3 8D7F20    LEA    DI,[BX+20]  addr of next dir entry
0000:7CE6 B90B00    MOV     CX,000B   count is 11
0000:7CE9 F3        REPZ                    is this "MSDOS.SYS"?
0000:7CEA A6        CMPSB
0000:7CEB 7418      JZ      FOUND_FILES they are equal

```

TALK:

Display "Non-System disk..." message,
wait for user to hit a key, restore
the INT 1E vector and then
call INT 19 to start boot processing
all over again.

```

0000:7CED BE9E7D    MOV     SI,7D9E   "Non-System disk..."
0000:7CF0 E85F00    CALL   MSG_LOOP   display message
0000:7CF3 33C0      XOR     AX,AX     INT 16 function
0000:7CF5 CD16      INT     16        read keyboard
0000:7CF7 5E        POP     SI        get INT 1E vector's
0000:7CF8 1F        POP     DS        address
0000:7CF9 8F04      POP     [SI]      restore the INT 1E
0000:7CFB 8F4402    POP     [SI+02]   vector's data
0000:7CFE CD19      INT     19        CALL INT 19 to try again

```

SETUP_TALK:

```

0000:7D00 58        POP     AX        pop junk off stack
0000:7D01 58        POP     AX        pop junk off stack
0000:7D02 58        POP     AX        pop junk off stack
0000:7D03 EBE8      JMP     TALK      now talk to the user

```

FOUND_FILES:

Compute the sector address of the first
sector of IO.SYS.

```

0000:7D05 8B471A    MOV     AX,[BX+1A] get starting cluster num
0000:7D08 48        DEC     AX        subtract 1
0000:7D09 48        DEC     AX        subtract 1
0000:7D0A 8A1E0D7C MOV     BL,[7C0D] sectors per cluster
0000:7D0E 32FF      XOR     BH,BH
0000:7D10 F7E3      MUL    BX        multiply
0000:7D12 0306497C ADD     AX,[7C49] add start addr of
0000:7D16 13164B7C ADC     DX,[7C4B] root dir (as LBA)

```

Read IO.SYS into memory at 0000:0700. IO.SYS

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

is 3 sectors long.

```
0000:7D1A BB0007    MOV    BX,0700    address to read into
0000:7D1D B90300    MOV    CX,0003    read 3 sectors
```

READ_LOOP:

Read the first 3 sectors of IO.SYS
(IO.SYS is much longer than 3 sectors).

```
0000:7D20 50          PUSH   AX          save AX
0000:7D21 52          PUSH   DX          save DX
0000:7D22 51          PUSH   CX          save CX
0000:7D23 E83A00    CALL   CONVERT     call conversion routine
0000:7D26 72D8      JB     SETUP_TALK  jmp if error
0000:7D28 B001      MOV    AL,01       read one sector
0000:7D2A E85400    CALL   READ_SECTORS read one sector
0000:7D2D 59          POP    CX          restore CX
0000:7D2E 5A          POP    DX          restore DX
0000:7D2F 58          POP    AX          restore AX
0000:7D30 72BB      JB     TALK        jmp if any INT 13 error
0000:7D32 050100   ADD    AX,0001     add one to the sector addr
0000:7D35 83D200   ADC    DX,+00      add one to the sector addr
0000:7D38 031E0B7C ADD    BX,[7C0B]   incr mem addr by sect size
0000:7D3C E2E2      LOOP  READ_LOOP    read next sector
```

Leave information in the AX, BX, CX and DX registers for IO.SYS to use. Finally, jump to IO.SYS at 0070:0000.

```
0000:7D3E 8A2E157C MOV    CH,[7C15]   media type
0000:7D42 8A16247C MOV    DL,[7C24]   drive number
0000:7D46 8B1E497C MOV    BX,[7C49]   get start addr of
0000:7D4A A14B7C    MOV    AX,[7C4B]   root dir (as LBA)
0000:7D4D EA00007000 JMP    0070:0000  JUMP TO 0070:0000
```

MSG_LOOP:

This routine displays a message using INT 10 one character at a time. The message address is in DS:SI.

```
0000:7D52 AC          LODSB             get message character
0000:7D53 0AC0      OR     AL,AL      end of message?
0000:7D55 7429      JZ     RETURN     jmp if yes
0000:7D57 B40E      MOV    AH,0E     display one character
0000:7D59 BB0700    MOV    BX,0007   video attributes
0000:7D5C CD10      INT    10        display one character
0000:7D5E EBF2      JMP    MSG_LOOP  do again
```

CONVERT:

This routine converts a sector address (an LBA) to a CHS address. The LBA is in DX:AX.

```
0000:7D60 3B16187C CMP    DX,[7C18]   hi part of LBA > sectPerTrk?
```

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

```

0000:7D64 7319      JNB     SET_CARRY      jmp if yes
0000:7D66 F736187C     DIV     WORD PTR [7C18]  div by sectors per track
0000:7D6A FEC2       INC     DL              add 1 to sector number
0000:7D6C 88164F7C     MOV     [7C4F],DL      save sector number
0000:7D70 33D2       XOR     DX,DX          zero DX
0000:7D72 F7361A7C     DIV     WORD PTR [7C1A]  div number of heads
0000:7D76 8816257C     MOV     [7C25],DL      save head number
0000:7D7A A34D7C      MOV     [7C4D],AX      save cylinder number
0000:7D7D F8         CLC                    clear carry
0000:7D7E C3         RET                    return

```

SET_CARRY:

```

0000:7D7F F9         STC                    set carry

```

RETURN:

```

0000:7D80 C3         RET                    return

```

READ_SECTORS:

The caller of this routine supplies:

AL = number of sectors to read
ES:BX = memory location to read into
and CHS address to read from in
memory locations 7c25 and 7c4d-7c4f.

```

0000:7D81 B402      MOV     AH,02          INT 13 read sectors
0000:7D83 8B164D7C  MOV     DX,[7C4D]      get cylinder number
0000:7D87 B106      MOV     CL,06          shift count
0000:7D89 D2E6      SHL     DH,CL          shift upper cyl left 6 bits
0000:7D8B 0A364F7C OR      DH,[7C4F]      or in sector number
0000:7D8F 8BCA      MOV     CX,DX          move to CX
0000:7D91 86E9      XCHG   CH,CL          CH=cyl lo, CL=cyl hi + sect
0000:7D93 8A16247C MOV     DL,[7C24]      drive number
0000:7D97 8A36257C MOV     DH,[7C25]      head number
0000:7D9B CD13      INT     13            read sectors
0000:7D9D C3         RET                    return

```

Data not used.

```

0000:7D90 ca86e98a 16247c8a 36257ccd 13c3.... *.....$.6%|... *

```

Messages here.

```

0000:7D90 ..... 0d0a * ..*
0000:7Da0 4e6f6e2d 53797374 656d2064 69736b20 *Non-System disk *
0000:7Db0 6f722064 69736b20 6572726f 720d0a52 *or disk error..R*
0000:7Dc0 65706c61 63652061 6e642070 72657373 *eplace and press*
0000:7Dd0 20616e79 206b6579 20776865 6e207265 * any key when re*
0000:7De0 6164790d 0a00.... ..... *ady... *

```

MS DOS hidden file names (first two root directory entries).

```

0000:7De0 ..... 494f 20202020 20205359 * IO SY*

```

The Master Boot Record The Partition Table & The Boot Sector

By Mark E. Donaldson

```
0000:7Df0 534d5344 4f532020 20535953 000055aa *SMSDOS SYS..U.*
```

The last two bytes contain a 55AAH signature.

```
0000:7Df0 ..... 55aa * U.*
```

A CLOSER LOOK AT THE MASTER BOOT RECORD

The MBR is the sector at cylinder 0, head 0, sector 1 of a hard disk. An MBR is created by the FDISK program. The FDISK program of all operating systems must create a functionally similar MBR. The MBR is first of what could be many partition sectors, each one containing a four entry partition table.

At the completion of your system's Power On Self Test (POST), INT 19 is called. Usually INT 19 tries to read a boot sector from the first floppy drive. If a boot sector is found on the floppy disk, the that boot sector is read into memory at location 0000:7C00 and INT 19 jumps to memory location 0000:7C00. However, if no boot sector is found on the first floppy drive, INT 19 tries to read the MBR from the first hard drive. If an MBR is found it is read into memory at location 0000:7c00 and INT 19 jumps to memory location 0000:7c00. The small program in the MBR will attempt to locate an active (bootable) partition in its partition table. If such a partition is found, the boot sector of that partition is read into memory at location 0000:7C00 and the MBR program jumps to memory location 0000:7C00. Each operating system has its own boot sector format. The small program in the boot sector must locate the first part of the operating system's kernel loader program (or perhaps the kernel itself or perhaps a "boot manager program") and read that into memory.

INT 19 is also called when the CTRL-ALT-DEL keys are used. On most systems, CTRL-ALT-DEL causes an short version of the POST to be executed before INT 19 is called.

- The MBR program code starts at offset 0000.
- The MBR messages start at offset 008b.
- The partition table starts at offset 00be.
- The signature is at offset 00fe.

Here is a summary of what all this does:

If an active partition is found, that partition's boot record is read into 0000:7c00 and the MBR code jumps to 0000:7c00 with SI pointing to the partition table entry that describes the partition being booted. The boot record program uses this data to determine the drive being booted from and the location of the partition on the disk.

If no active partition table entry is found, ROM BASIC is entered via INT 18. All other errors cause a system hang.

The first byte of an active partition table entry is 80. This byte is loaded into the DL register before INT 13 is called to read the boot sector. When INT 13 is called, DL is the BIOS device number. Because of this, the boot sector read by this MBR program can only be read from BIOS device number 80 (the first hard disk). This is one of the reasons why it is usually not possible to boot from any other hard disk.

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

The MBR program uses the CHS based INT 13H AH=02H call to read the boot sector of the active partition. The location of the active partition's boot sector is in the partition table entry in CHS format. If the drive is >528MB, this CHS must be a translated CHS. Here is the entire MBR record (hex dump and ascii).

```

OFFSET 0 1 2 3 4 5 6 7 8 9 A B C D E F *0123456789ABCDEF*
000000 fa33c08e d0bc007c 8bf45007 501ffbf0 * .3.....|..P.P...*
000010 bf0006b9 0001f2a5 ea1d0600 00bebe07 *.....*
000020 b304803c 80740e80 3c00751c 83c610fe *...<.t.<.u.....*
000030 cb75efcd 188b148b 4c028bee 83c610fe *.u.....L.....*
000040 cb741a80 3c0074f4 be8b06ac 3c00740b *.t.<.t.....<.t.*
000050 56bb0700 b40ecd10 5eebf0eb febf0500 *V.....^.....*
000060 bb007cb8 010257cd 135f730c 33c0cd13 *..|...W...s.3...*
000070 4f75edbe a306ebd3 bec206bf fe7d813d *Ou.....}.=*
000080 55aa75c7 8bf5ea00 7c000049 6e76616c *U.u.....|..Inval*
000090 69642070 61727469 74696f6e 20746162 *id partition tab*
0000a0 6c650045 72726f72 206c6f61 64696e67 *le.Error loading*
0000b0 206f7065 72617469 6e672073 79737465 * operating syste*
0000c0 6d004d69 7373696e 67206f70 65726174 *m.Missing operat*
0000d0 696e6720 73797374 656d0000 00000000 *ing system.....*
0000e0 00000000 00000000 00000000 00000000 *.....*
0000f0 TO 0001af SAME AS ABOVE
0001b0 00000000 00000000 00000000 00008001 *.....*
0001c0 0100060d fef83e00 00000678 0d000000 *.....>....x....*
0001d0 00000000 00000000 00000000 00000000 *.....*
0001e0 00000000 00000000 00000000 00000000 *.....*
0001f0 00000000 00000000 00000000 000055aa *.....U.*

```

Here is the disassembly of the MBR. This sector is initially loaded into memory at 0000:7c00 but it immediately relocates itself to 0000:0600.

	BEGIN:		NOW AT 0000:7C00, RELOCATE
0000:7C00	FA	CLI	disable int's
0000:7C01	33C0	XOR AX,AX	set stack seg to 0000
0000:7C03	8ED0	MOV SS,AX	
0000:7C05	BC007C	MOV SP,7C00	set stack ptr to 7c00
0000:7C08	8BF4	MOV SI,SP	SI now 7c00
0000:7C0A	50	PUSH AX	
0000:7C0B	07	POP ES	ES now 0000:7c00
0000:7C0C	50	PUSH AX	
0000:7C0D	1F	POP DS	DS now 0000:7c00
0000:7C0E	FB	STI	allow int's
0000:7C0F	FC	CLD	clear direction
0000:7C10	BF0006	MOV DI,0600	DI now 0600
0000:7C13	B90001	MOV CX,0100	move 256 words (512 bytes)
0000:7C16	F2	REPZ	move MBR from 0000:7c00
0000:7C17	A5	MOVSW	to 0000:0600
0000:7C18	EA1D060000	JMP 0000:061D	jmp to NEW_LOCATION
NEW_LOCATION:		NOW AT 0000:0600	
0000:061D	BE07	MOV SI,07BE	point to first table entry
0000:0620	B304	MOV BL,04	there are 4 table entries

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

```

SEARCH_LOOP1:                                SEARCH FOR AN ACTIVE ENTRY
0000:0622 803C80    CMP     BYTE PTR [SI],80  is this the active entry?
0000:0625 740E      JZ      FOUND_ACTIVE     yes
0000:0627 803C00    CMP     BYTE PTR [SI],00  is this an inactive entry?
0000:062A 751C      JNZ     NOT_ACTIVE       no
0000:062C 83C610    ADD     SI,+10           incr table ptr by 16
0000:062F FECB      DEC     BL               decr count
0000:0631 75EF      JNZ     SEARCH_LOOP1     jmp if not end of table
0000:0633 CD18      INT     18              GO TO ROM BASIC

FOUND_ACTIVE:                                FOUND THE ACTIVE ENTRY
0000:0635 8B14      MOV     DX,[SI]         set DH/DL for INT 13 call
0000:0637 8B4C02    MOV     CX,[SI+02]     set CH/CL for INT 13 call
0000:063A 8BEE      MOV     BP,SI          save table ptr

SEARCH_LOOP2:                                MAKE SURE ONLY ONE ACTIVE ENTRY
0000:063C 83C610    ADD     SI,+10           incr table ptr by 16
0000:063F FECB      DEC     BL               decr count
0000:0641 741A      JZ      READ_BOOT       jmp if end of table
0000:0643 803C00    CMP     BYTE PTR [SI],00  is this an inactive entry?
0000:0646 74F4      JZ      SEARCH_LOOP2     yes

NOT_ACTIVE:                                  MORE THAN ONE ACTIVE ENTRY FOUND
0000:0648 BE8B06    MOV     SI,068B        display "Invld prttn tbl"

DISPLAY_MSG:                                  DISPLAY MESSAGE LOOP
0000:064B AC        LODSB                    get char of message
0000:064C 3C00     CMP     AL,00           end of message
0000:064E 740B     JZ      HANG             yes
0000:0650 56       PUSH    SI              save SI
0000:0651 BB0700    MOV     BX,0007        screen attributes
0000:0654 B40E     MOV     AH,0E          output 1 char of message
0000:0656 CD10     INT     10             to the display
0000:0658 5E       POP     SI              restore SI
0000:0659 EBF0     JMP     DISPLAY_MSG     do it again

HANG:                                          HANG THE SYSTEM LOOP
0000:065B EBFE     JMP     HANG            sit and stay!

READ_BOOT:                                    READ ACTIVE PARTITION BOOT RECORD
0000:065D BF0500    MOV     DI,0005        INT 13 retry count

INT13RTRY:                                    INT 13 RETRY LOOP
0000:0660 BB007C    MOV     BX,7C00        read 1 sector
0000:0663 B80102    MOV     AX,0201        save DI
0000:0666 57       PUSH    DI              save DI
0000:0667 CD13     INT     13             read sector into 0000:7c00
0000:0669 5F       POP     DI              restore DI

```


The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

The last two bytes contain a 55AAH signature.

A CLOSER LOOK AT THE PARTITION TABLE

FDISK creates all partition records (sectors). The primary purpose of a partition record is to hold a partition table. The rules for how FDISK works are unwritten but so far most FDISK programs (DOS, OS/2, WinNT, etc) seem to follow the same basic idea.

First, all partition table records (sectors) have the same format. This includes the partition table record at cylinder 0, head 0, sector 1 -- what is known as the Master Boot Record (MBR). The last 66 bytes of a partition table record contain a partition table and a 2 byte signature. The first 446 bytes of these sectors usually contain a program but only the program in the MBR is ever executed (so extended partition table records could contain something other than a program in the first 466 bytes).

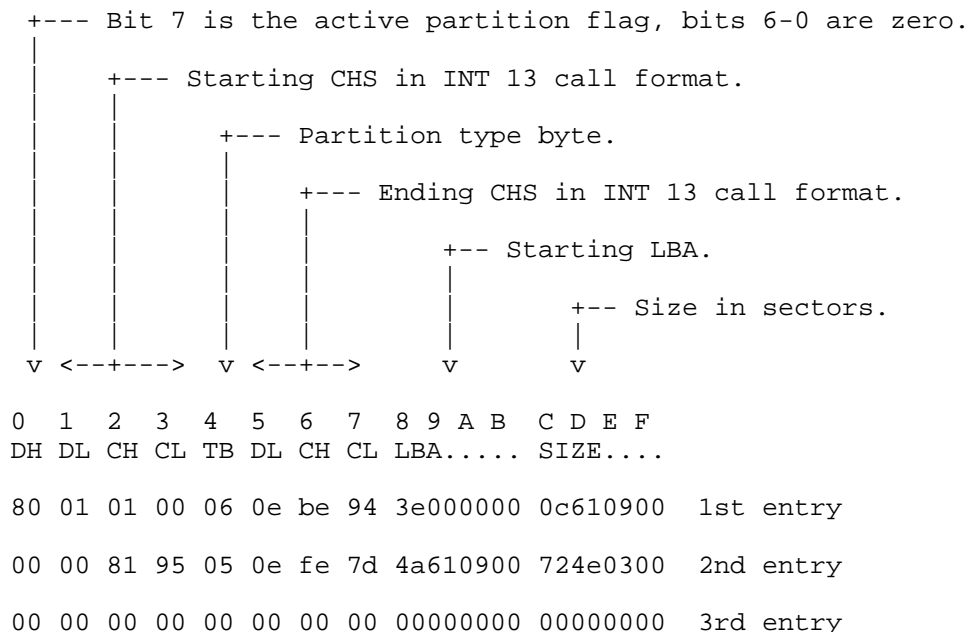
Second, extended partitions are "nested" inside one another and extended partition table records form a "linked list". I will attempt to show this in a diagram below.

Partition Table Entry Format

Each partition table entry is 16 bytes and contains things like the start and end location of a partition in CHS, the start in LBA, the size in sectors, the partition "type" and the "active" flag. Warning: older versions of FDISK may compute incorrect LBA or size values. And note: When your computer boots itself, only the CHS fields of the partition table entries are used (another reason LBA doesn't solve the >528MB problem). The CHS fields in the partition tables are in L-CHS format.

There is no central clearing house to assign the codes used in the one byte "type" field. But codes are assigned (or used) to define most every type of file system that anyone has ever implemented on the x86 PC: 12-bit FAT, 16-bit FAT, HPFS, NTFS, etc. Plus, an extended partition also has a unique type code.

The 16 bytes of a partition table entry are used as follows:



The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

00 00 00 00 00 00 00 00 00 00000000 00000000 4th entry

Bytes 0-3 are used by the small program in the Master Boot Record to read the first sector of an active partition into memory. The DH, DL, CH and CL above show which x86 register is loaded when the MBR program calls INT 13H AH=02H to read the active partition's boot sector.

These entries define the following partitions:

1. The first partition, a primary partition DOS FAT, starts at CHS 0H,1H,1H (LBA 3EH) and ends at CHS 294H,EH,3EH with a size of 9610CH sectors.
2. The second partition, an extended partition, starts at CHS 295H,0H,1H (LBA 9614AH) and ends at CHS 37DH,EH,3EH with a size of 34E72H sectors.
3. The third and fourth table entries are unused.

Partition Table Rules

Keep in mind that there are NO written rules and NO industry standards on how FDISK should work but here are some basic rules that seem to be followed by most versions of FDISK:

1. In the MBR there can be 0-4 "primary" partitions, OR, 0-3 primary partitions and 0-1 extended partition entry.
2. In an extended partition there can be 0-1 "secondary" partition entries and 0-1 extended partition entries.
3. Only 1 primary partition in the MBR can be marked "active" at any given time.
4. In most versions of FDISK, the first sector of a partition will be aligned such that it is at head 0, sector 1 of a cylinder. This means that there may be unused sectors on the track(s) prior to the first sector of a partition and that there may be unused sectors following a partition table sector.

For example, most new versions of FDISK start the first partition (primary or extended) at cylinder 0, head 1, sector 1. This leaves the sectors at cylinder 0, head 0, sectors 2...n as unused sectors. This same layout may be seen on the first track of an extended partition. See example 2 below.

Also note that software drivers like Ontrack's Disk Manager depend on these unused sectors because these drivers will "hide" their code there (in cylinder 0, head 0, sectors 2...n). This is also a good place for boot sector virus programs to hang out.

5. The partition table entries (slots) can be used in any order. Some versions of FDISK fill the table from the bottom up and some versions of FDISK fill the table from the top down. Deleting a partition can leave an unused entry (slot) in the middle of a table.

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

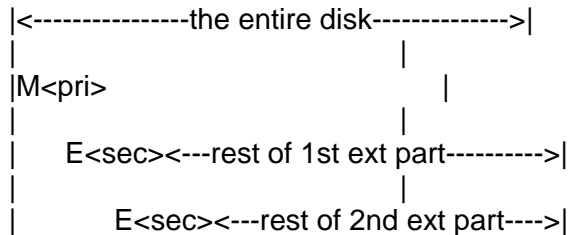
6. And then there is the "hack" that some newer OS's (OS/2 and Linux) use in order to place a partition spanning or passed cylinder 1024 on a system that does not have a CHS translating BIOS. These systems create a partition table entry with the partition's starting and ending CHS information set to all FFH. The starting and ending LBA information is used to describe the location of the partition. The LBA can be converted back to a CHS -- most likely a CHS with more than 1024 cylinders. Since such a CHS can't be used by the system BIOS, these partitions can not be booted or accessed until the OS's kernel and hard disk device drivers are loaded. It is not known if the systems using this "hack" follow the same rules for the creation of these type of partitions.

There are NO written rules as to how an OS scans the partition table entries so each OS can have a different method. For DOS, this means that different versions could assign different drive letters to the same FAT file system partitions.

Partition Nesting

What is meant by partitions being "nested" within each other? Lets look at this example:

- M = Master Boot Record (and any unused sectors on the same track)
- E = Extended partition record (and any unused sectors on the same track)
- pri = a primary partition (first sector is a "boot" sector)
- sec = a secondary partition (first sector is a "boot" sector)



The first extended partition is described in the MBR and it occupies the entire disk following the primary partition. The second extended partition is described in the first extended partition record and it occupies the entire disk following the first secondary partition.

Partition Table Linking

Partition table linking means that the MBR has an entry that describes (points to) the first extended partition, the first extended partition table has an entry that describes (points to) the second extended partition table, and so on. There is, in theory, no limited to out long this linked list is. When you ask FDISK to show the DOS "logical drives" it scans the linked list looking for all of the DOS FAT type partitions that may exist. Remember that in an extended partition table, only two entries of the four can be used (rule 2 above).

Within a partition, the layout of the file system data varies greatly. However, the first sector of a partition is expected to be a "boot" sector. A DOS FAT file system has: a boot sector, first FAT sectors, second FAT sectors, root directory sectors and finally the file data area.

The Master Boot Record

The Partition Table & The Boot Sector

By Mark E. Donaldson

CHS=x-1,n,n	user data area)	:	
CHS=x,0,1	Boot sector for the OS/2 HPFS file system partition	:	: an OS/2
CHS=x,0,2 to CHS=y-1,n,n	rest of the OS/2 HPFS file system partition	:	: HPFS file : system
CHS=y,0,1	Partition record for the extended partition containing a partition record program (never executed) and a partition table	:	
	+-----+ DOS FAT partition description	:	points to CHS=b+1
	+-----+ unused table entry	:	
	+-----+ unused table entry	:	
	+-----+ unused table entry	:	
CHS=y,0,2 to CHS=y,0,n	the rest of the first track of the extended partition	:	: normally : unused
CHS=y,1,1	Boot sector for the DOS FAT partition	:	: a DOS FAT
CHS=y,1,2 to CHS=n,n,n	rest of the DOS FAT partition (FAT table, root directory and user data area)	:	: file : system

EXAMPLE 3

Here is a partition record from an extended partition (the first sector of an extended partition). Note that it contains no program code. It contains only the partition table and the signature data.

```

OFFSET 0 1 2 3 4 5 6 7 8 9 A B C D E F *0123456789ABCDEF*
000000 00000000 00000000 00000000 00000000 *.....*
000010 TO 0001af SAME AS ABOVE
0001b0 00000000 00000000 00000000 00000001 *.....*
0001c0 8195060e fe7d3e00 0000344e 03000000 *.....}>...4N....*
0001d0 00000000 00000000 00000000 00000000 *.....*
0001e0 00000000 00000000 00000000 00000000 *.....*
0001f0 00000000 00000000 00000000 000055aa *.....U.*

```