Bootstrapping

In computing, bootstrapping refers to a process starting up a computer, in which a mechanism is needed to execute the software program that is responsible for executing software programs (the operating system).

Bootstrapping was shortened to booting, or the process of starting up any computer; the verb "boot" is similarly derived. A "bootstrap" most commonly refers to the simple program itself that actually begins the initialization of the computer's operating system - like GRUB, LILO or NTLDR. Modern personal computers have the ability of using their network interface card (NIC) for bootstrapping; on IA-32 (x86) and IA-64 (Itanium) this method is implemented by PXE and Etherboot.

The computer is regarded as starting in a "blank" condition - although magnetic core memory retains its state with the power off. The bootstrap part would be a short simple piece of code that loads the main code.

Thus the bootstrap sequence begins with a very simple hardware state, plus arrangements that cause something to be read into memory and executed. That something is the bootstrap loader, which is capable of loading an arbitrary program into an arbitrary part of memory, and then starting it. The initial state is half-way through the hardware's instruction cycle, as if the instruction had been read into the instruction register but before it is executed. This is followed by the execution of a program that is not in memory except that its instructions are read (from specially-prepared input) into memory just before they are executed, then that simple programm is executed, that (due to carefully-prepared input) places a program somewhere in memory, that finally is executed.

Master boot record

Structure of a Master Boot Record

Address		Description		Size in bytes
Hex	Dec			
0000	0	Code Area		max. **446**
01B8	440	Optional Disk signature		4
01BC	444	Usually Nulls; 0x0000		2
01BE	446	**Table of primary partitions** (Four 16-byte entries, IBM Partition Table scheme)		**64**
01FE	510	55h	MBR signature; 0xAA55[1]	**2**
01FF	511	Aah		
		MBR, total size: 446 + 64 + 2 =		**512**

A Master Boot Record (MBR), or partition sector, is the 512-byte boot sector that is the first sector ("Sector 0") of a partitioned data storage device such as a hard disk. (The boot sector of a non-partitioned device is a Volume Boot Record, these are usually different, although it's possible to create a record that acts as both, it's called Multi Boot Record.) The MBR may be used for one or more of the following:

· Holding a disk's primary partition table
· Bootstrapping operating systems, after the computer's BIOS passes execution to machine code instructions contained within the MBR.
· Uniquely identifying individual disk media, with a 32-bit disk signature; even though it may never be used by the machine the disk is running on.

Due to the broad popularity of IBM PC-compatible computers, this type of MBR is widely used, to the extent of being supported by, and incorporated into, other computer types, including newer cross-platform standards for bootstrapping and partitioning.

MBRs and disk partitioning

Layout of one 16-byte partition record

Offset	Description
0x00	(1 byte) Status (0x80 = bootable, 0x00 = non-bootable, other = malformed
0x01	(3 bytes) Cylinder-head-sector address of the first sector in the partition[5]
0x04	(1 byte) Partition type
0x05	(3 bytes) Cylinder-head-sector address of the last sector in the partition[6]
0x08	(4 bytes) Logical block address of the first sector in the partition
0x0C	(4 bytes) Length of the partition, in sectors

Where a data storage device has been partitioned with (what Microsoft terms) the MBR Partition Table scheme (i.e., the conventional IBM PC partitioning scheme), the master boot record contains the primary partition entries in its partition table. The partition table entries for other, secondary, partitions are stored in Extended Boot Records, BSD disklabels, and Logical Disk Manager metadata partitions that are described by those primary entries.

By convention, there are exactly four primary partition table entries in the MBR Partition Table scheme, although some DOS operating systems did extend this to five or even eight.

Where a data storage device has been partitioned with the GUID Partition Table scheme, the Master Boot Record will still contain a partition table, but its only purpose is to indicate the existence of the GUID Table and to prevent utility programs that only understand the MBR Partition Table scheme from creating any partitions in what they would see as only free space on the disk.

MBRs and Sysem Bootstrap

On IA-32 IBM PC compatible machines using the MBR Partition Table scheme, the bootstrapping firmware contained within the ROM BIOS loads and executes the master boot record. Because the i386 family of processors boot up in real mode, the code in the MBR is real mode machine language instructions. This code normally passes control by chain loading the Volume Boot Record of the active (primary) partition, although some boot managers replace that conventional code with their own.

The conventional MBR code expects the MBR Partition Table scheme to have been used, and scans the list of (primary) partition entries in its embedded partition table to find the only one that is marked with the active flag. It then loads and runs the Volume Boot Record for that partition. (Thus the master boot record, like other boot sectors, is a target for boot-sector infecting computer viruses. See boot sector.)

The MBR replacement code in some boot managers can perform a variety of tasks, and what those tasks are varies from boot manager to boot manager. In some, for example, it loads the remainder of the boot manager code from the first track of the disk, which it assumes to be "free" space that is not allocated to any disk partition, and executes it. In others, it uses a table of embedded disk locations to locate the remainder of the boot manager code to load and to execute. (Both approaches have problems. The first relies on behavior that is not universal across all disk partitioning utilities. The second requires that the embedded list of disk locations be updated when changes are made that would relocate the remainder of the code.)

On machines that do not use IA-32 processors, and on machines that use Extensible Firmware Interface (EFI) firmware, this design is unsuitable, and the MBR is not used as part of the system bootstrap. On the latter, the firmware is instead capable of directly understanding the GPT partitioning scheme and the FAT filesystem format, and loads and runs programs held as files in the EFI System Partition. The MBR will only be involved insofar as it might contain the partition table if the MBR Partition Table scheme has been used.

Editing/Replacing/Backing up MBR

Though it's possible to directly manipulate the bytes in the MBR sector using various Disk Editors, Microsoft-based operating systems have included the means to write fixed sets of functioning code to the MBR sector for some time. Since MS-DOS 5.0, the DOS-mode program FDISK has included the (undocumented, but widely used) switch /mbr, which will rewrite the MBR code.

Using the FDISK /MBR command from MS-DOS 6.22/Windows 95, or any earlier versions, may zero-out the partition table under some circumstances; especially if the last two bytes of the MBR sector are not 55h followed by AAh. If a DOS boot is necessary, use the FDISK program from a Windows 98, or other FAT32 capable boot disk, as a safer alternative; to ensure only the code area, not the partition table, is overwritten.) Under Windows 2000 or later, the Recovery Console can be used to write new MBR code to a hard disk using its fixmbr command.

Some third-party utilities may also be used for directly editing the contents of partition tables (without requiring any knowledge of hexadecimal or disk/sector editors)]

MS-DOS and Windows

For MS-DOS and Windows-based operating systems, a number of third-party utilities exist for backing-up the MBR and/or partition table data. (See the "External links" section below.)

Unix-like systems

The dd utility, which is available on most Unix-like systems, can be used to create backup images of the master boot record. Care should be taken when using the dd utility, as incorrect values for its parameters can cause data corruption. You must have the appropriate access to the target device in order to use these commands successfully.

To back up the MBR of a hard disk or other storage device, use the following command to create a file named mbr.bin containing the entire contents of the master boot record:

dd if=device of=mbr.bin bs=512 count=1

where device is the path of the device file corresponding to the entire contents of the storage device whose MBR is to be backed up. For example, the primary master IDE device (ATA bus 0, device 0) is typically represented by the device file /dev/hda under Linux.
To restore the MBR from a backup file to a storage device, use the following command, which replaces the first 512 bytes with the contents of the file named mbr.bin:

dd if=mbr.bin of=device bs=512 count=1

To back up or restore only the executable code area, assuming that the executable code uses at most 446 bytes, use the option bs=446 instead of bs=512 when executing the appropriate command. If a full 512-byte backup has already been made, a 446-byte partial restore can still be used, but the opposite is not true (dd cannot copy more bytes from a file than it contains). Using a partial restore can be useful if both the code area and the partition table have been changed and it is necessary to restore only the original code without losing the new partition table.