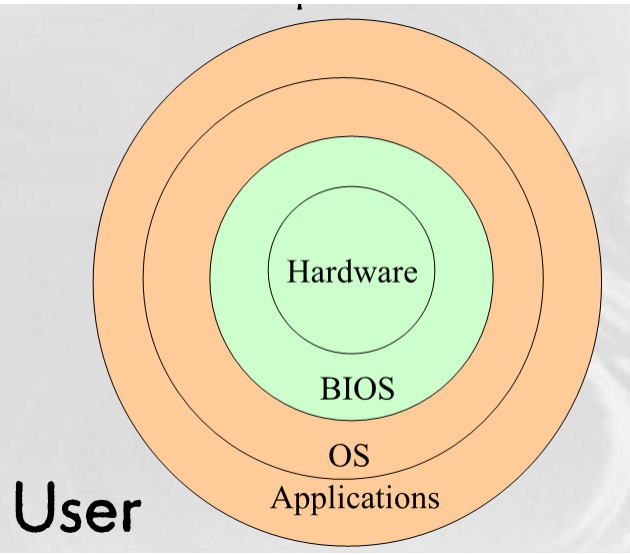
CS415 Module 1 Part A - Introduction

Athens State University

Contents

1 So you pressed the power button?

There's a lot going on here



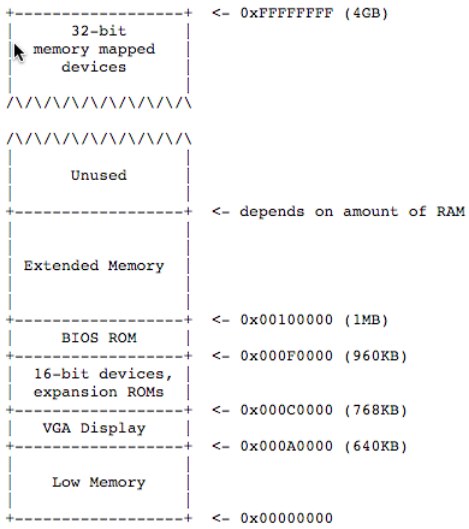
I've pressed the BOB

- Pressing the power button turns on the power supply
  - The power supply does a self-test
  - When the power supply has decided that it's got stable power, it turns on a signal to the processor that the power is good and applies current to the different parts of the motherboard
  - The takes between 0.1 and 0.5 seconds

I've pressed the BoB

- The CPU is hard-coded to start executing by reading the instruction at 0xFFFF:0x0000
  - This is the top of the memory map and is typically mapped into Read-Only Memory (ROM)
  - At this point, there will be a jump instruction that points the processor at the start of the "BIOS" code in ROM

IBM PC Physical Memory



- Early PCs had 1MB of physical memory
  - Only the first 640kB were available for use
- A section of memory was reserved to store the "Basic I/O System" or "BIOS"

Basic Input/Output System (BIOS)

- A set of small programs (or services) that are designed to operate each major PC subsystem
- Each service is invoked by a set of standard calls
- Interrupt driven in the PC BIOS
- The BIOS runs a Power-On-Self-Test (POST) program when the power is turned on
- BIOS is kept in read-only memory, called "firmware"

Modern versions of the BIOS added features

- Security settings
- Boot sequence options
- Time stamp changes
- Hardware management, thermal control
- Power management configuration
- Virus/intrusion detection

Power-On Self Test

The early IBM PC's would write a byte to a specific IO port on the system bus that indicated what happened in the case of a start-up failure. One could then use special test equipment to read this number.

As time progressed, BIOS vendors such as AMI would map these codes to the speaker port on the motherboard.

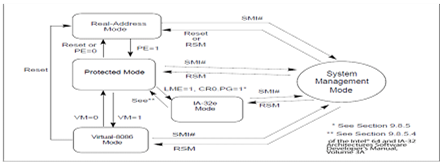
Each type of error generated a specific number of beeps:

Beeps	Meaning
1	Memory refresh timer error
2	Parity error in base memory (first 64 KiB block)
3	Base memory read/write test error
4	Motherboard timer not operational (check all PSU to MB connectors seated)
5	Processor failure
6	8042 Gate A20 test error (cannot switch to protected mode)
7	General exception error (processor exception interrupt error)
8	Display memory error (system video adapter)
9	AMI BIOS ROM checksum fix
10	CMOS shutdown register read/write fix
11	Cache memory test failed
12	Motherboard does not detect a RAM module (continuous beeping)

For those of you who may be thinking of taking the CompTIA A+ certification exam, that exam covers the following specific beep codes:

Beeps	Meaning
Steady, short beeps	Power supply issues
Long continuous tone	Memory failure
Steady, long beeps	Power supply failure
No beep	Plug that thing into the wall
One long, two short beeps	Video card failure

Processor Mode



- Intel processors operate in one of five modes: Real-address mode,Protected mode, System management mode, 32-bit extended mode, 64-bit extended mode
- CPU starts in real mode and kick starts itself into protected mode

BIOS extensions

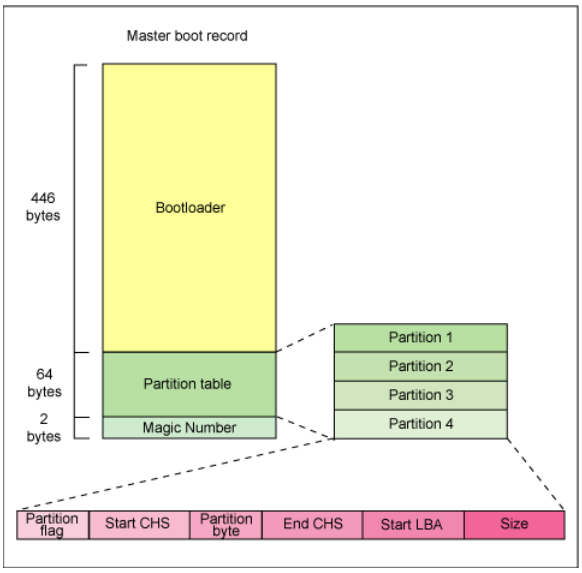
- Some devices need to load their own firmware routines into memory
- Video adapters have their firmware mapped into 0xc000:0x0000 to 0xC7000:0x000
  - The BIOS scans this memory to load the code in the video card ROM
  - This will include routines to kick-start the GPU
- Other devices: Disk controllers, Network adapters, and PXE network boot ROMS.

2 And now the fun really begins

Transfer of control to the OS

- The BIOS scans list of boot drives stored in the CMOS
- Each boot device will have a Master Boot Record stored in cyl. 0, head 0, sector 1 of the device
- Data is loaded from that location into 0x0000:0x7c00 and tested for a valid signature of 0x55aa in the last two bytes

Master Boot Record

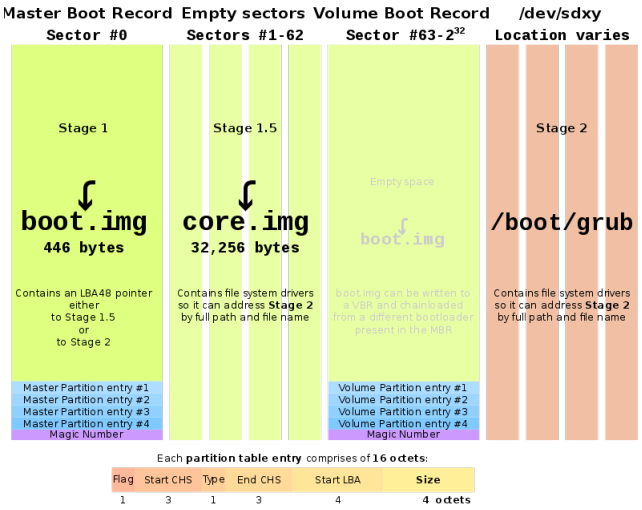


- MBR has two parts
  - The partition table of the fixed disk
  - Partition loader code with instructions on how to continue the boot process
- The partition load will find the active partition on the boot disk
  - Look at the boot record in the first sector of this disk
    - Boot record contains a partition table and location of the first OS file that needs to be loaded

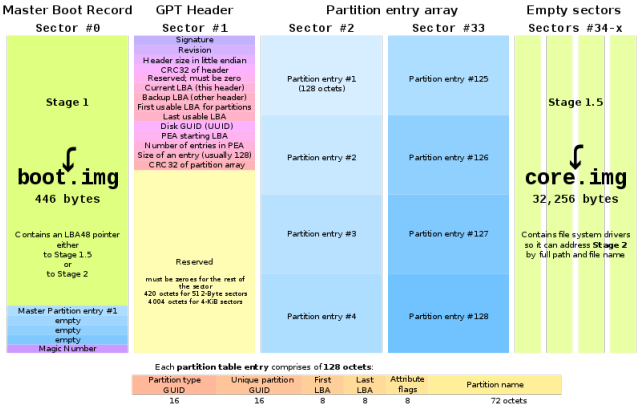
Stage 2 Boot Loader

- Program that does the real work of loading the operating system
  - This is GRUB: GNU GRand Unified Bootloader) for Linux
  - This is Windows Boot Manager in Windows Vista (and later versions)

Case Study: GRUB



Case Study: GRUB



The following quote from the Wikipedia article on GRUB explains the differences and some of the design issues that have to be addressed:

> The legacy MBR partition table supports a maximum of four partitions and occupies 64 bytes. Together with the optional disk signature (four bytes) and disk timestamp (six bytes), this leaves between 434 and 446 bytes available for the machine code of a boot loader. Although such a small space can be sufficient for very simple boot loaders,[7] it is not big enough to contain a boot loader supporting complex and multiple file systems, menu-driven selection of boot choices, etc. Boot loaders with bigger footprints are thus split into pieces, where the smallest piece fits into and resides within the MBR, while larger piece(s) are stored in other locations (for example, into empty sectors between the MBR and the first partition) and invoked by the boot loader's MBR code.
>
> Operating system kernel images are in most cases files residing on appropriate file systems, but the concept of a file system is unknown to the BIOS. Thus, in BIOS-based systems, the duty of a boot loader is to access the content of those files, so it can be loaded into the RAM and executed
>
> One possible approach for boot loaders to load kernel images is by directly accessing hard disk sectors without understanding the underlying file system. Usually, additional level of indirection is required, in form of maps or map files – auxiliary files that contain a list of physical sectors occupied by kernel images. Such maps need to be updated each time a kernel image changes its physical location on disk, due to installing new kernel images, file system defragmentation etc. Also, in case of the maps changing their physical location, their locations need to be updated within the boot loader's MBR code, so the sectors indirection mechanism continues to work. This is not only cumbersome, but it also leaves the system in need of manual repairs in case something goes wrong during system updates.
>
> Another approach is to make a boot loader aware of the underlying file systems, so kernel images are configured and accessed using their actual file paths. That requires a boot loader to contain a driver for each of the supported file systems, so they can be understood and accessed by the boot loader itself. This approach eliminates the need for hardcoded locations of hard disk sectors and existence of map files, and does not require MBR updates after the kernel images are added or moved around. Configuration of a boot loader is stored in a regular file, which is also accessed in a file system-aware way to obtain boot configurations before the actual booting of any kernel images. As a result, the possibility for things to go wrong during various system updates is significantly reduced. As a downside, such boot loaders have increased internal complexity and even bigger footprints.

3 Off to the FUTURE!

Beyond the PC

- Beyond the PC, system firmware can be become quite complicated
  - Out-of-band management (lights-out management): remote management of servers
  - Separate dedicated physical network for management
  - Intelligent Platform Management Interface (IPMI)
    - Server out-of-band management protocol standard
  - Baseboard management controller: separate co-processor or independent processor for managing servers

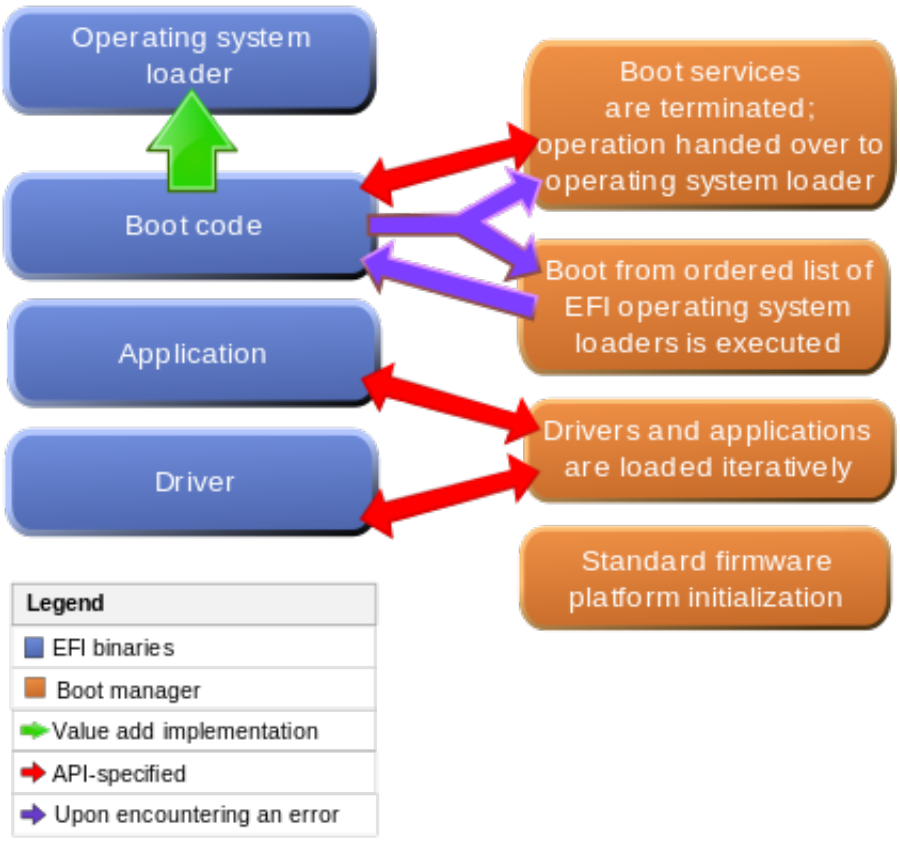
Need for more than just "BIOS"

- As PC market matured, the need for better "manageability" became a requirement for PC vendors
- More services required at boot time plus need more flexible boot architecture as devices like the GPU and fancy disk controllers became more common
- Vendors were extending BIOS in different ways to do the same thing
- Need for native support of 32 and 64-bit processors

Unified Extensible Firmware Interface (UEFI)

- Industry specification that defines interface between operating system and platform firmware
- Supports x86, x86-64, ARM, and ARM64 architectures
- Supported by Windows, Linux, and macOS
- Works with both MBR and GPT partition formats

The UEFI Boot Process



UEFI Secure Boot

- UEFI a secure boot process based on cryptographic hashing
  - Firmware vendors write a public "Platform Key" to the firmware
  - Additional "Key Exchange Keys" can be added to the keystore but one must know the private portion of the "Platform Key"
- Criticism
  - Starting with Windows 8, MSFT requires OEMs to enable secure boot using Microsoft's private key, but backed off on this requirement for Intel-based systems after industry complaint
  - Poor implementation of UEFI firmware has resulted in some OEMs having issues with bricking of machines