Overview of PC and Bootstraping

Hui Chen [a]

[a]CUNY Brooklyn College

January 27, 2020

Outline

1. PC Hardware
   - CPU and Memory
   - Registers, Memory, and Cache
   - Memory and Cache
   - Input and Output
2. x86 Instruction and Assembly Languages
3. Bootstrap

PC (Personal Computer)

- A PC is a computer that aheres to a few industry standards. The goal of the standards is that a given peice of software can run on PCs from multiple vendors.
- The standards have evolved over time.

Main Components

- CPU
- Memory
- Input/Output (I/O) devices

CPU and Memory

A CPU (central procesing unit, or processor) runs a conceptually simple loop,

1. it reads an address in the Program Counter,
2. it reads a machine instruction from that address in memory,
3. it advances the program counter past the instruction,
4. it executes the instruction, and
5. it repeats the above steps.

Note that some instructions do change the Program Counter including branches and function calls.

Storage

A program can access two types of storage directly, registers and memory.

Registers

- Registers are storage cells inside the CPU itself;
- are capable of holding a machine word-size value, e.g., 16, 32, or 64 bits, and
- the data stored in registers can be read or write written quickly, typically in a single CPU cycle, as such, they are the fastest storage for data in a computer system.

x86 Registers

- The modern x86 provides 8 general purpose 32-bit registers
  eax, ebx, ecx, edx, edi, esi, ebp, esp, and eip
- The bottom half of each of the 32-bit registers are the 8 "16-bit registers",
  ax, bx, cx, dx, di, si, bp, sp, and ip.
- The first 4 16-bit registers have names for the top half and bottom half of the registers, e.g.,
  - al and ah denote the low and the high bits of ax
  - similarly there are bl, bh, cl, ch, dl, dh.
- x86 has also 8 80-bit floating point registers and a few special-purpose registers, e.g.,
  - Control registers cr0, cr2, cr3, and cr4
  - Debug registers dr0, dr1, dr2, and dr3
  - Segment regsiters cs, ds, es, fs, gs, and ss
  - Global and local descriptor table pseudo-registers gdtr and ldtr.

Control and segment registers are important to any operating system.

Memory and Cache

- Registers are fast but expensive and few.
- Main memory (main random-access memory or RAM) are slower than registers but are many.
- Cache memory serves as a middle ground between registers and memory both in access time and size.
  - x86 processors store copies of recently-accessed sections of main memory in on-chip cache memory.
  - x86 processor typically have two-levels of cache, a small but fast first-level cache (L1 cache) and a larger but slower second-level cache (L2 cache)
- Note that x86 processors hide the cache from programmers.

x86 I/O Instructions

x86 processors provides special I/O that read and write values from device addresses (to memory or registers on the device)

- `in` and `out` instructions
- Device addresses are called I/O ports

Via the device's ports (or memory), programs can

- examine the device's status, and
- cause the device to take actions, e.g., by reading or writing to the I/O ports, the program cause the disk interface hardware to read and write sectors on the disk.

Memory-Mapped I/O

- On modern x86 architecture, a device can have designated memory addresses and the processor communicates with the device by reading and writing values to those addresses.
- Memory-mapped I/O are used for most high-speed devices such as netowrk, disk, and graphic controllers.
- For backwards compatibility, x86 still have the in and out instructions and some (legacy) devices use them.

x86 Instruction Set

- PCs have a processor that implements the x86 instruction set.
- The instruction set was originally designed by Intel and has become a standard.
- The standard evolves. Thus far, the new standard is backward compatible with the old standards.
- This backward compatibility creates a little complexity for boot loader as an x86 processor once powered on simulates an Intel 8088, the CPU chip in the original IBM PC released in 1981.
- The instruction set is the programmer's interface to the physical machine.

x86 Assembly Language

x86 assembly language has two main syntax branches,

- Intel syntax, originally used for documentation of the x86 platform. Intel Intel syntax is dominant in the MS-DOS and Windows world
- AT&T syntax is dominant in the Unix world, since Unix was created at AT&T Bell Labs.

Comparison between AT&T and Intel Syntax

Let's take a quick look at,
https://en.wikipedia.org/wiki/X86_assembly_language

BIOS

- A PC has a program called the BIOS (Basic Input/Output System) stored in non-volatile memory, e.g., read-only memory (ROM), on the motherboard.
- The program contains various low-level routines that are specific to the hardware supplied with the motherboard.
  - Among the routes are the POST. The POST (Power On Self Test) is a set of routines including the memory check, system bus check, and other low-level initialization so the CPU can set up the computer properly.
- One job of the BIOS is to prepare the hardware and then transfer control to the operating system, specifically, it loads the boot sector and executes the boot loader code.
  - The boot sector contains the boot loader, a short program that loads the operating system kernel into memory.
  - The boot sector is always the first 512-byte sector of the boot disk.
  - The BIOS always loads the boot sector at memory address 0x7c00 and the jumps to that address.

Bootstrapping PC

1. When the computer powers on, the processor's registers are set to some predefined values.
   - `eip` is initialized to `0xfffffff0`.
   - `cr0` is initialized to `0x00000000`, i.e, the PE (Protection Enabled) bit is 0 indicating the processor is running in 16-bit real mode.
2. The processor begins executing instructions at address 0xfffffff0, which really resides in the BIOS's memory.
3. Usually this address contains a jump instruction to the BIOS's POST (Power-on Self Test) routines.
4. The POST determines the boot device. Modern BIOS implementations permit the selection of a boot device, allowing booting from a floppy, CD-ROM, hard disk, or other devices.
5. The POST triggers a soft interrupt by calling the INT 0x19 instruction. The interrupt handler reads 512 bytes from the boot sector, i.e., the first sector of boot device into the memory at address 0x7c00.

Real Mode and Protected Mode

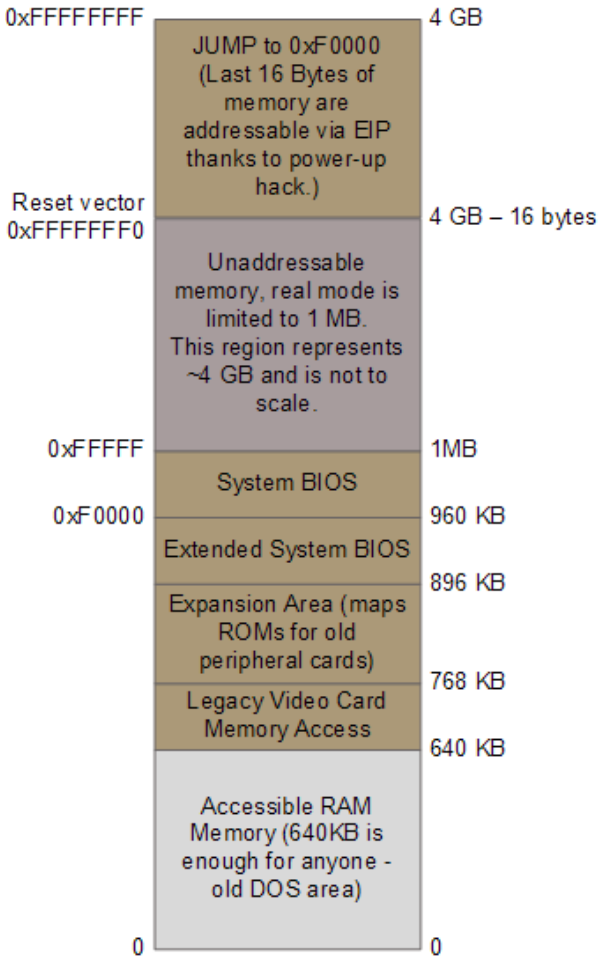
x86 processors have two modes, real mode and protected mode.

- When the processor is powered on, it enters the real mode, i.e., simulates an Intel 8088 processor, the processor on an IBM PC released in 1981
- So the boot sector code is 16-bit code.
- To use the features provided by modern x86 architecture, the boot loader needs to switch the processor to the protected mode.

Real Mode

"Real Mode" in x86-compatitle PC means memory addresses in real mode always correspond to real locations in memory.

PC Memory Map



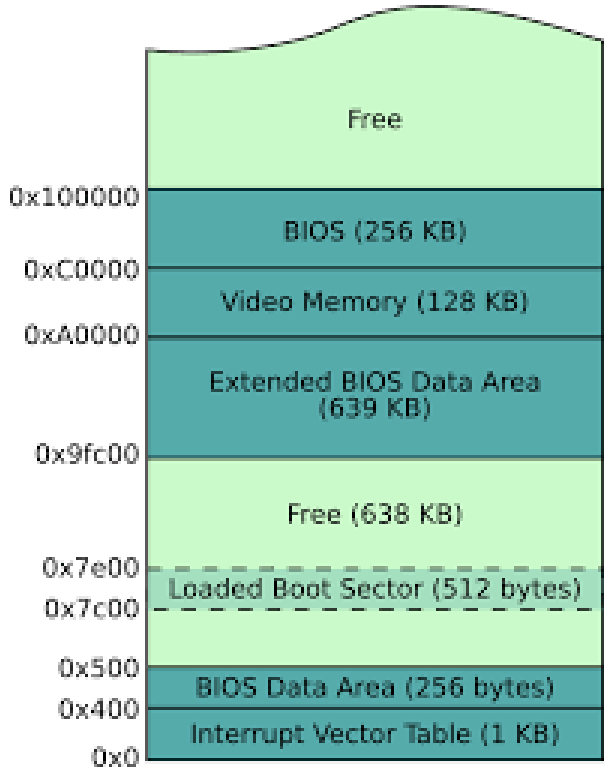


Figure: Source: https://manybutfinite.com/post/how-computers-boot-up/

Real Mode Address

Address bus is 20 bits. How does PC in Real Mode refer an address?

- Use two values, a 16-bit segment address, and a 16-bit offset address to compute the 20-bit address.

  ```
  Address = (Segment << 4) | Offset
  ```

- Example. A segment address is 0x1080, an offset address 0x0342.

  ```
  Address = (0x1080 << 4) | 0x0342
  ```

Real Mode Segments

In Real Mode, registers CS, DS, SS, and ES stores segment addresses.

- They point to the currently used program code segment (CS), the current data segment (DS), the current stack segment (SS), and one extra segment (ES), respectively.

Real Mode Segment Example

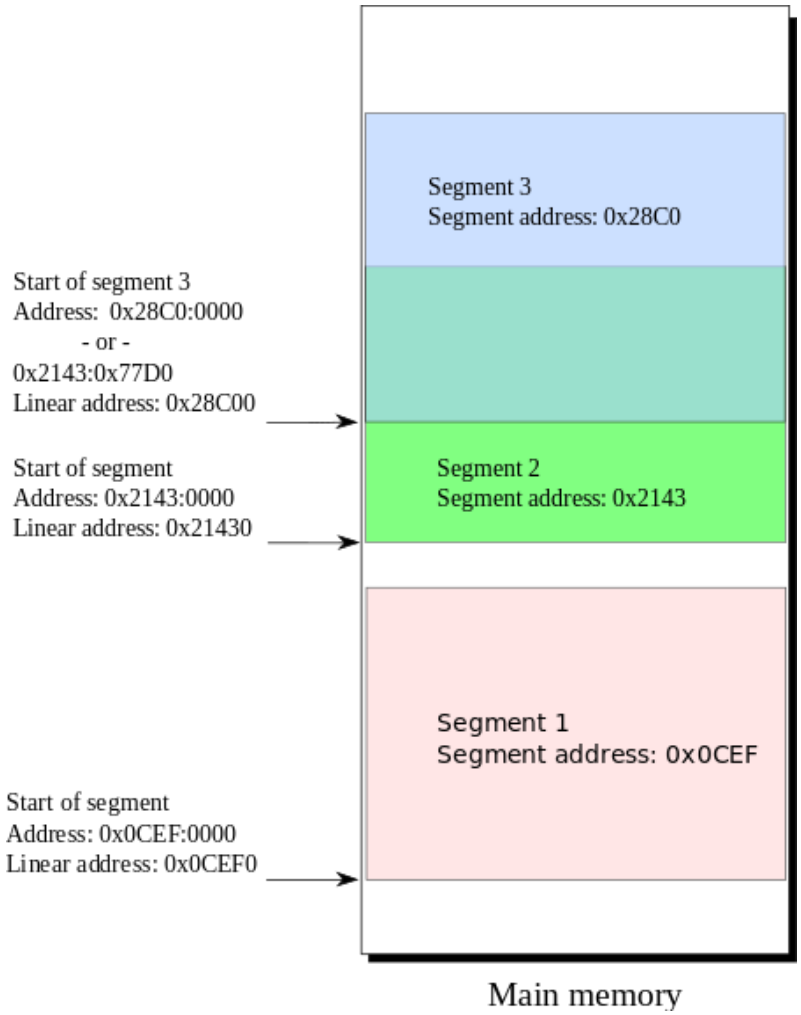


Figure: Source: https://en.wikipedia.org/wiki/X86_memory_segmentation

Boot Sector Program Examples

We discuss a few simple boot sector programs including a simple interrupt handler.

- To run conveniently the boot sector code, we use PC virtual machines and emulators.
  1. Set up a Debian Linux system on an Oracle VirtualBox virtual machine
  2. Install PC emulator and assembly language compiler on the Debian Linux system
  3. Enter a boot sector program, compile and run it using the PC emulator.
  4. See the tutorial for more details.