Operating Systems

Boot your OS

Fall 2020

```
vahid@DESKTOP-J2OMVJH:~$ cat os.logo
.............................Operating Systems.............................
...............K...................................................................
...............E...................................................................
........M E M O R Y..............................P R O C E S S.....
........A.......N........        Presented By:        ..................Y.......
........N.......E........  Professor Mohsen Sharifi  ..................S.......
........A.......L........    --------------------     ..................T.......
........G................          Tutors:            ..........T H R E A D...
........E................       Vahid Mohsseni        ..................M.......
........M................    Ehsan SeyedAliAkbar      ..................C.......
..S C H E D U L E R......      Farbod Shahinfar       ..................A.......
........N................................................M A L L O C...
........T..........Iran University of Science and Technology..........L.......
...................................Fall 2020...................................
vahid@DESKTOP-J2OMVJH:~$ curl -L https://os-course.github.io/fall20/ClassTime
>>>>     SCHEDULE     <<<<<br>
> Sundays  and  Tuesdays <<br>
>     10:30 - 12:00      <<br>
<img src="/fall20/_images/banner.png">
```

Groups Collaboration

- Are they determined?
- Pick a day for online meeting about 5 to 10 min for each group
- Deadline is next Tuesday

- Cheating in Exams and Quizzes!

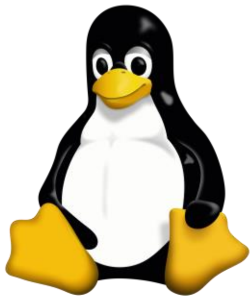
Intro

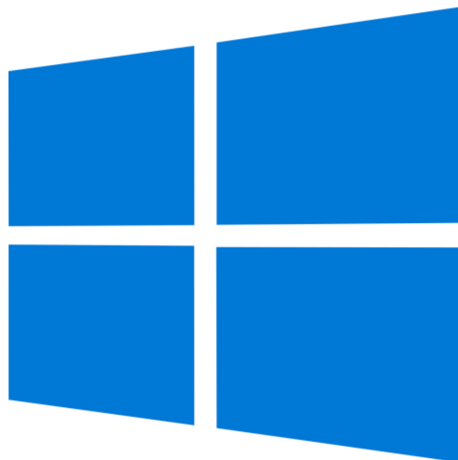
We have all used an Operating System before.

Linux

Windows

OS X





Intro

Many years ago….

develop your OS from scratch to

start your research

solve a problem

programming

Now…

wonderful, beautiful machines above layers of codes!

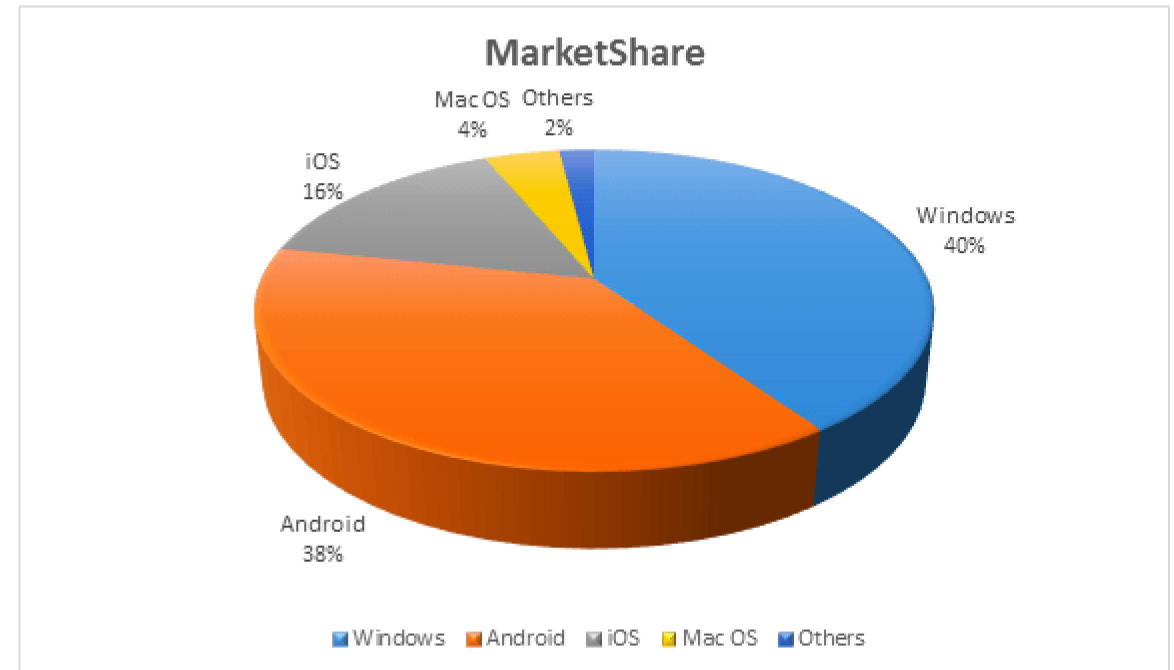
Intro

Operating Systems at the heart of it all…

- Make the incredible advance in the underlying technology available to rapid evolving body of applications.
  - Processing, Communications, Storages, Interaction, Protected sharing.

The key building blocks are

- Processes, Scheduling
- Concurrency, Coordination
- Address space, Translation
- Protection, Isolation, Sharing, Security
- Communication, Protocols
- Persistent storage, transactions, consistency
- Interfaces to all devices

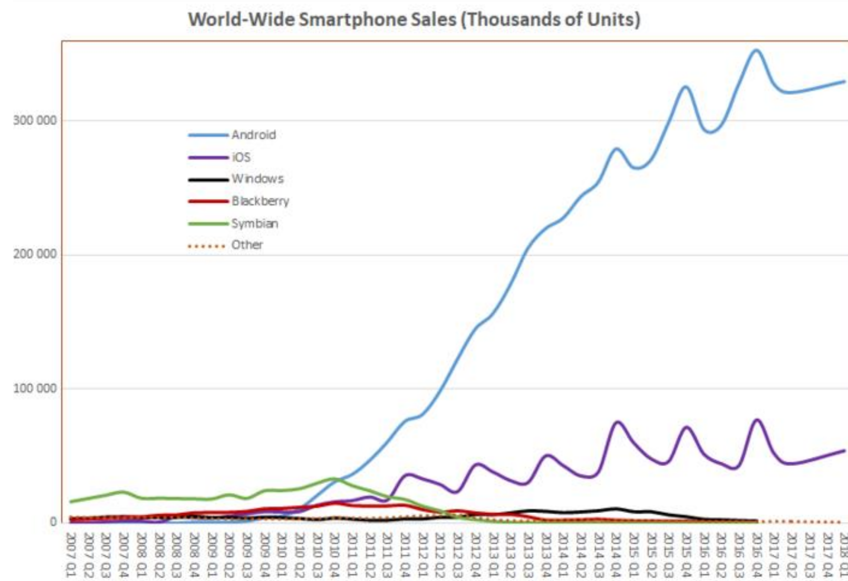


Intro

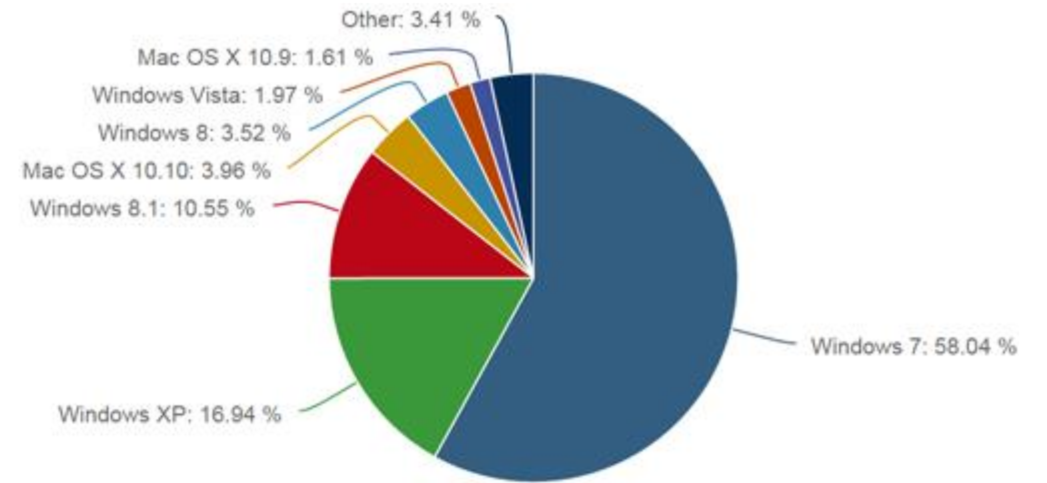
Year 2019

2.5 billion android devices

1.9 billion IOS devices



2018



2015

What is an Operating System?

An illusionist!

Provides clean, easy to use abstraction of physical resources.

- Infinite memory, dedicated machine
- Higher level objects: files, users, message, and etc.
- Masking limitations, virtualization

Intro

- How a computer boots?
- How to write a low-level program on bare hardware where there is no OS yet?
- How to configure a CPU so that utilize its functionality?
- How to bootstrap code written in a higher-level language to make progress towards our own OS?

Intro

Our goal is not to develop all functionality of OS!

We aim to introduce some low-level concepts and how early OS is written for now.

We will familiarize you to assembly language and machine codes.

BIOS

What happen when we turn on or reboot our computer?

How to load Data that are stored somewhere in permanent storage device.

e.g. USB, Hard-Disk, floppy!

At the start of the computer, the system offers little of services.

even there is no a simple file system!

BIOS

Hopefully, There is something!

Basic Input/Output Software known as BIOS.

- Is a collection of software routines that are initially loaded from a chip into memory and initialized when a computer switched on.
- Provides auto-detection and basic control of the computer's essential devices such as screen, keyboard, hard disks.
- Completes some low-level tests of the hardware, particularly checks whether memory is working or not.
- Can not simply load a file containing the Operating System. Has no idea!!

BIOS

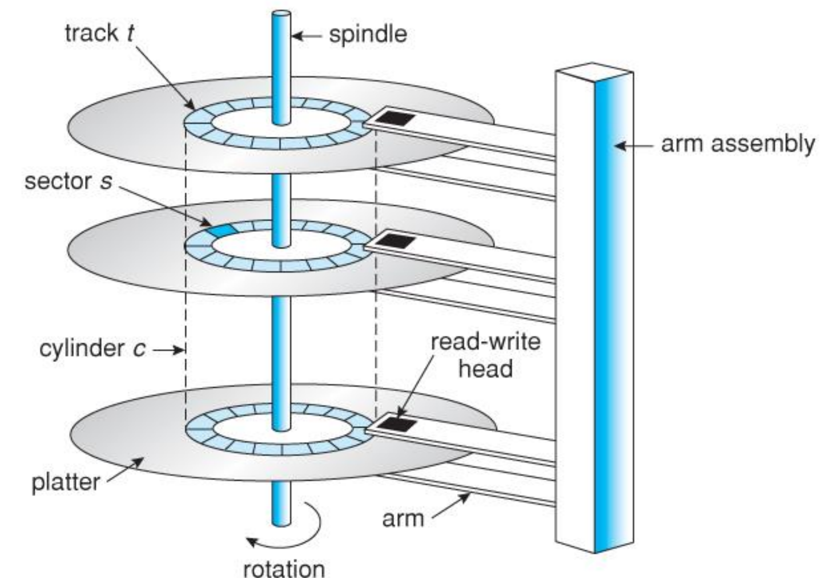
BIOS reads specific sectors of data (usually 512 bytes in size).

The easiest place for BIOS to find OS is the first sector of hard disk. i.e. Cylinder 0, Head 0, Sector 0

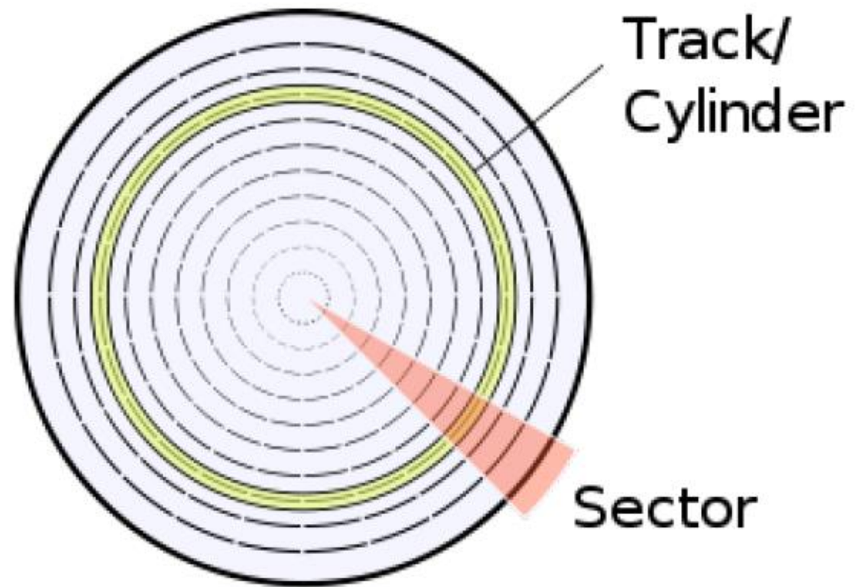
Known as Boot Sector.

Note that CPU does not differentiate between data and code.

Both interpreted as instructions.



Hard Disk Schema



Boot Sector and Magic Number

At the end of boot sector (last two bytes) must be set to the magic number `0xaa55`.

Size of boot sector? 512 bytes

- What is magic number?
- A magic number is a numeric or string constant that indicates the file type.
- More info at this url.

An example

Using a binary text editor, such as TextPad or GHex, allows us to read/write raw bytes to a file. Other editor convert characters into their ASCII values.

‘A’ -> 0x41

```
e9 fd ff 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
```

An actual machine code for boot sector

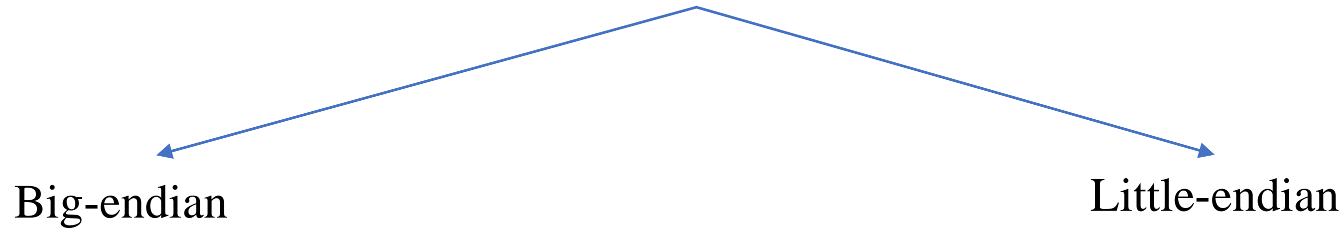
An example

```
e9 fd ff 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
```

- The initial three bytes, in hexadecimal, are machine codes instructions to perform an endless jump.
- The last two bytes make up the magic number for boot sector. But why not in their order?!
- The rest of the file is filled by zero. Asterisk (* character) indicates the repetition of zeroes.

Endianness

The order or sequence of bytes of a word of digital data in computer memory.

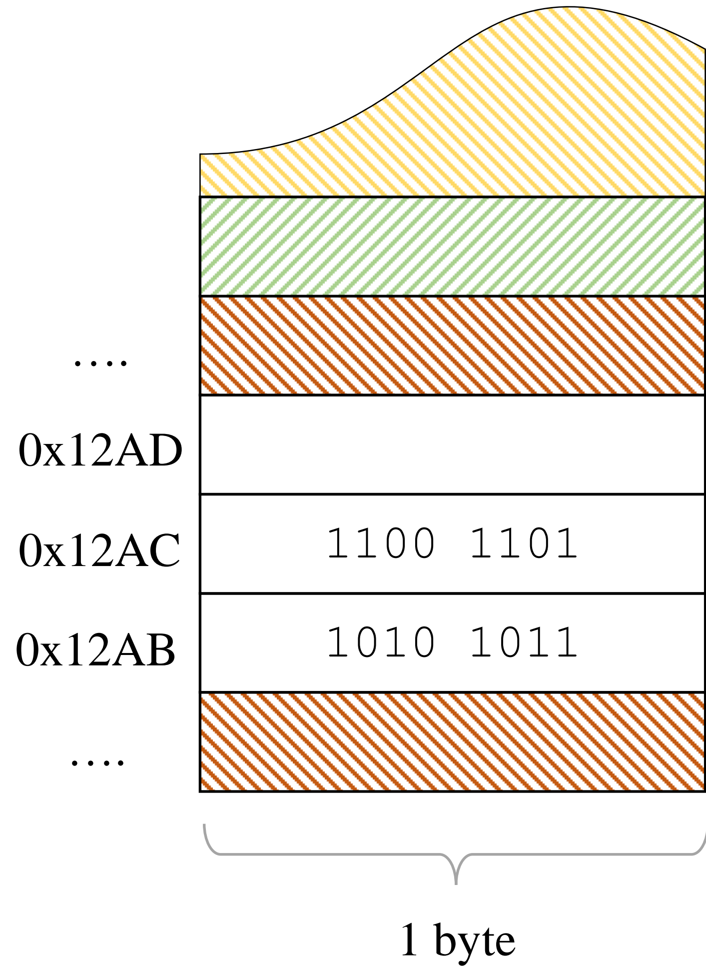


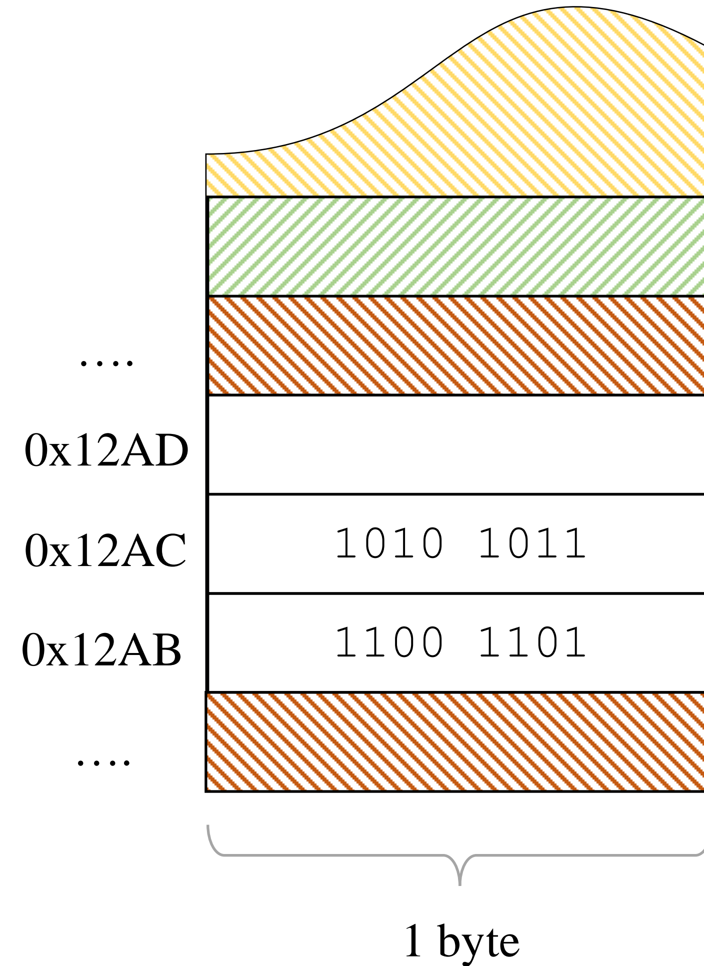
BE: stores the most-significant byte of a word at the smallest memory address and least significant byte at the largest.

LE: stores least-significant byte at the smallest address.

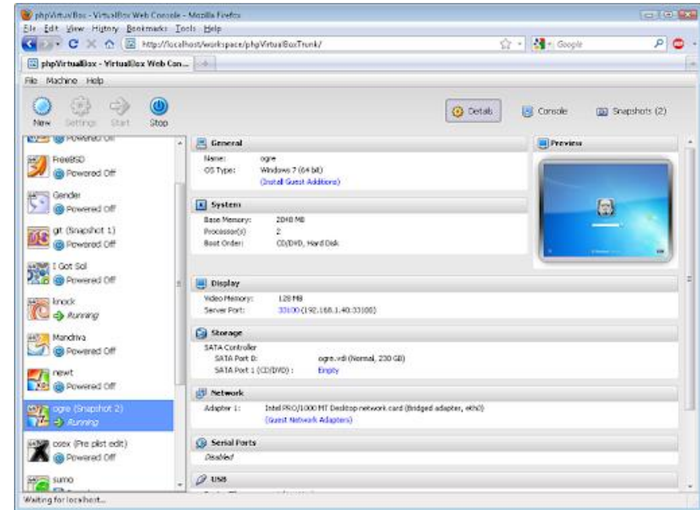
Endianness

The data to store is `0xABCD` (the binary is `1010 1011 1100 1101`).





Virtual machine software and Emulators



We can boot an OS on

- Real hardware
- Using Virtual machine tools
  - Virtual Box
  - VMWare
- Emulators
  - Boshs
  - QEmu

QEMU

```
Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok



iPXE 1.0.0+ -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP iSCSI DNS TFTP AoE bzImage COMBOOT ELF MBOOT PXE PXEXT

net0: 52:54:00:12:34:56 using rtl8139 on PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
DHCP (net0 52:54:00:12:34:56)...... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
Next server: 10.0.2.2
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

Booting from Floppy...
Boot failed: could not read the boot disk

No bootable device.
```

Why Hexadecimal?

Easy to convert to binary and 4-bit segments

Shorter than decimal

Easy to convert binary to Hex

1110001001001011

1
2
8
64
512
8192
16384
32768

57931

1110001001001011

1110 0010 0100 1011

8+2+1=11
4
2
8+4+2=14

E 2 4 B

Assembly

nasm is a compiler for assembly language.

```
$ nasm boot.asm -f bin -o boot.bin
```

- `-f bin` indicates that the `nasm` produce raw binary output file.

Boot Sector Again

```
; a simple boot sector assembly code


loop:
       jmp loop


times 510 – ($ - $$) db 0



dw 0xaa55
```

Hello World!

Too fast!

Real mode and Protected mode

As the CPU technology and new generations of CPUs come to the market, what should happen to all those codes written for old CPUs?

Simple solution: Emulates the oldest CPU in the family. So we can have backward compatibility.

Intel 8086 supports 16-bit instructions and had no notion of memory protection.

Because of backward compatibility, the CPU starts in 16-bit real mode.

In x86 family, we have 32-bit and 64-bit instructions.

After OS boots, it explicitly switches up to protected mode.

Interrupts

- Interrupts are a mechanism that allow a CPU temporarily to halt what it is doing and run some other.
- Each interrupt is represented by unique number which is an index to interrupt vector.
- BIOS initialize the table at the start of memory containing a pointer to Interrupt Service Routines (ISR).

For example: `0x10` causes screen related interrupts.

Registers

- Four general purpose registers
  - `ax`, `bx`, `cx`, and `dx`
- Load a data
  - `mov ax, 1234    ; loads 1234 decimal number into ax`
  - `mov bh, 0xab    ; loads 0xab hex number into 8 most-significant bits of bx`
  - `mov cl, 0x10    ; loads 0x10 hex number into 8 least-significant bits of cx`

Using Screen (Monitor) by BIOS

- Interrupt `0x10` is screen-related interrupt.
- To write a value in the screen, `ah` should be set to `0x0e`.
- It writes the value of `al` register onto the screen.

```
; an example for using interrupt

mov ah, 0x0e

mov al, 'V'

int 0x10
```

Hello!

```
; Hello world!

mov ah, 0x0e

mov al, 'H'
int 0x10

mov al, 'e'
int 0x10

mov al, 'l'
int 0x10

mov al, 'l'
int 0x10

mov al, 'o'
int 0x10


jmp $

times 510 - ($ - $$) db 0

dw 0xaa55
```

Compile with `nasm` and open binary

- `nasm hello.asm -f bin -o out.bin`
- `od -t x1 -A n out.bin`

`od` displays a file into a given format.

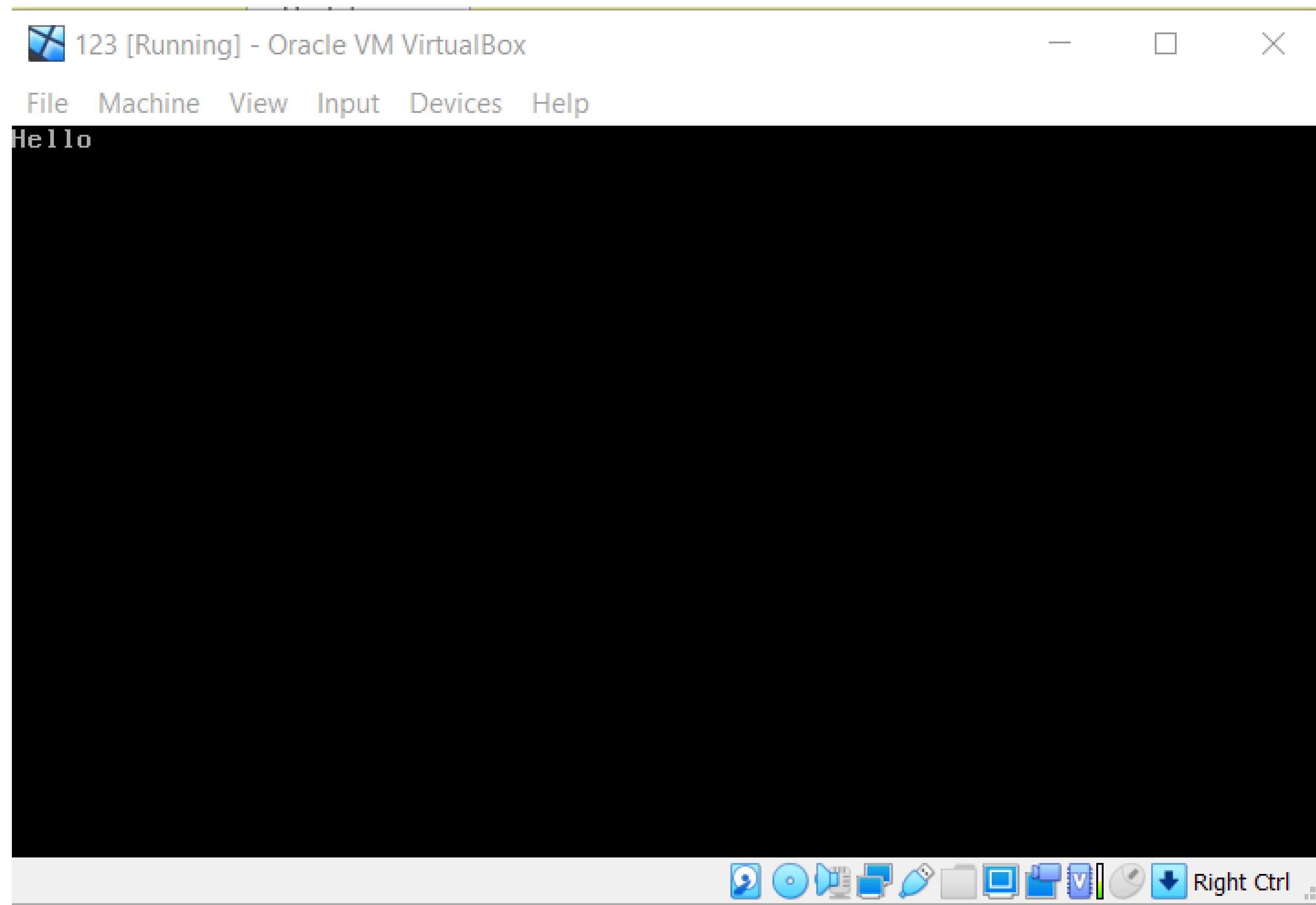
```
 b4 0e b0 48 cd 10 b0 65 cd 10 b0 6c cd 10 b0 6c
 cd 10 b0 6f cd 10 eb fe 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
```

Compile with `nasm` and open binary

- `od -t x1 -A n out.bin`

```
512 bytes copied, 0.0088102 s, 58.1 kB/s
 b4 0e b0 48 cd 10 b0 65 cd 10 b0 6c cd 10 b0 6c
 cd 10 b0 6f cd 10 eb fe 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
```

Running on Virtual Box



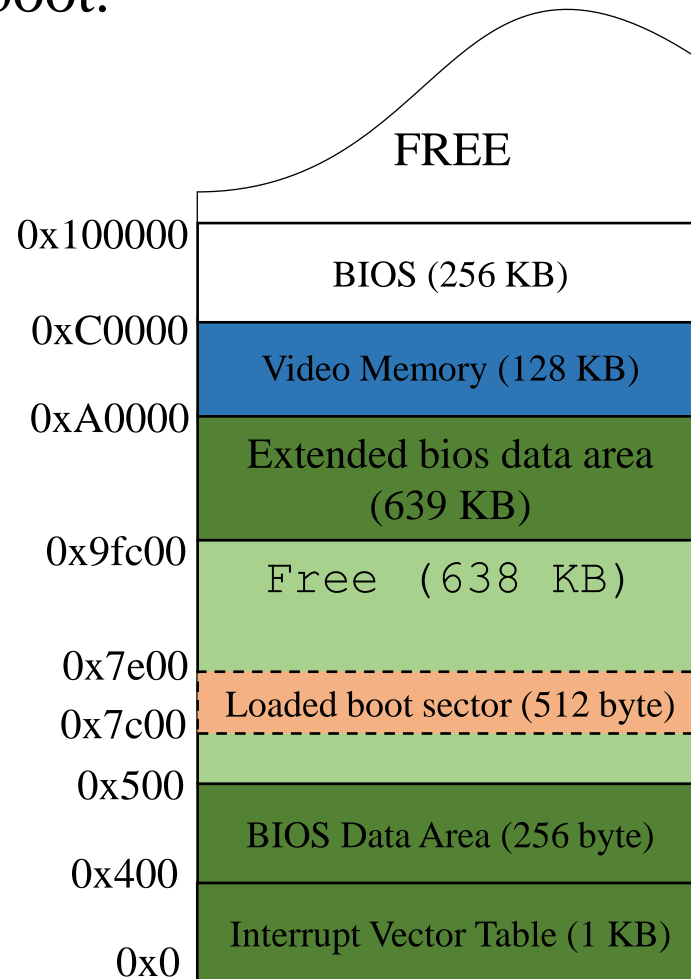
Memory

CPU always fetches and executes instructions from memory.

Our boot sector code is somewhere in memory. but where?

Memory

Memory layout after boot.



Load data from memory

```
; Hello world!

mov ah, 0x0e

mov al, x_char
int 0x10

mov al, [x_char]
int 0x10

mov bx, x_char
add bx, 0x7c00
mov al, [bx]
int 0x10

mov al, [0x7c1d]
int 0x10
```

```
jmp $


x_char:
        db "X"


times 510 - ($ - $$) db 0

dw 0xaa55
```

Origin

```
; Hello world!
[org 0x7c00]

mov ah, 0x0e

mov al, x_char
int 0x10

mov al, [x_char]
int 0x10

mov bx, x_char
add bx, 0x7c00
mov al, [bx]
int 0x10

mov al, [0x7c1d]
int 0x10
```

```
jmp $


x_char:
        db "X"


times 510 - ($ - $$) db 0

dw 0xaa55
```

ASCII-0

Problem: How to print a string?

start from the first address

then increase the address

until it reaches to ZERO

```
my_string:
        db “Hello World!”, 0
```

Stack

Is stack something special and complicated thing?

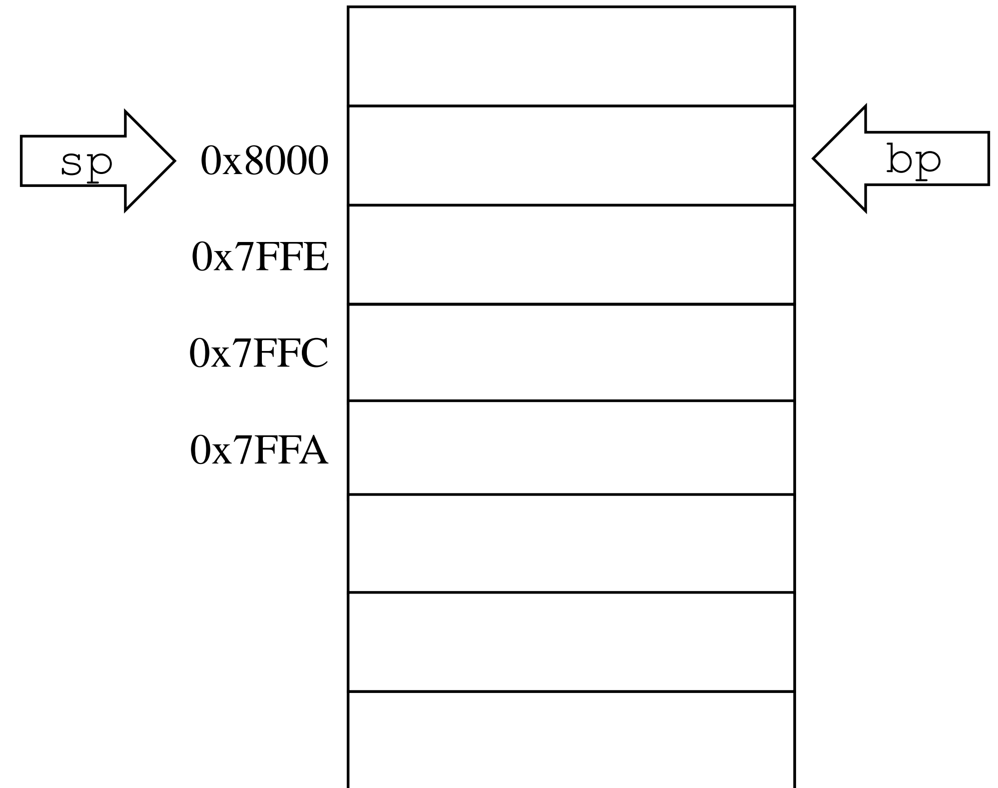
Stack is simple solution to a CPU inconvenience:

The limited number of registers.

Two simple instructions: `push` and `pop`.

Two special registers: `bp` and `sp`.

Stack grows downwards.



Stack

Is stack something special and complicated thing?

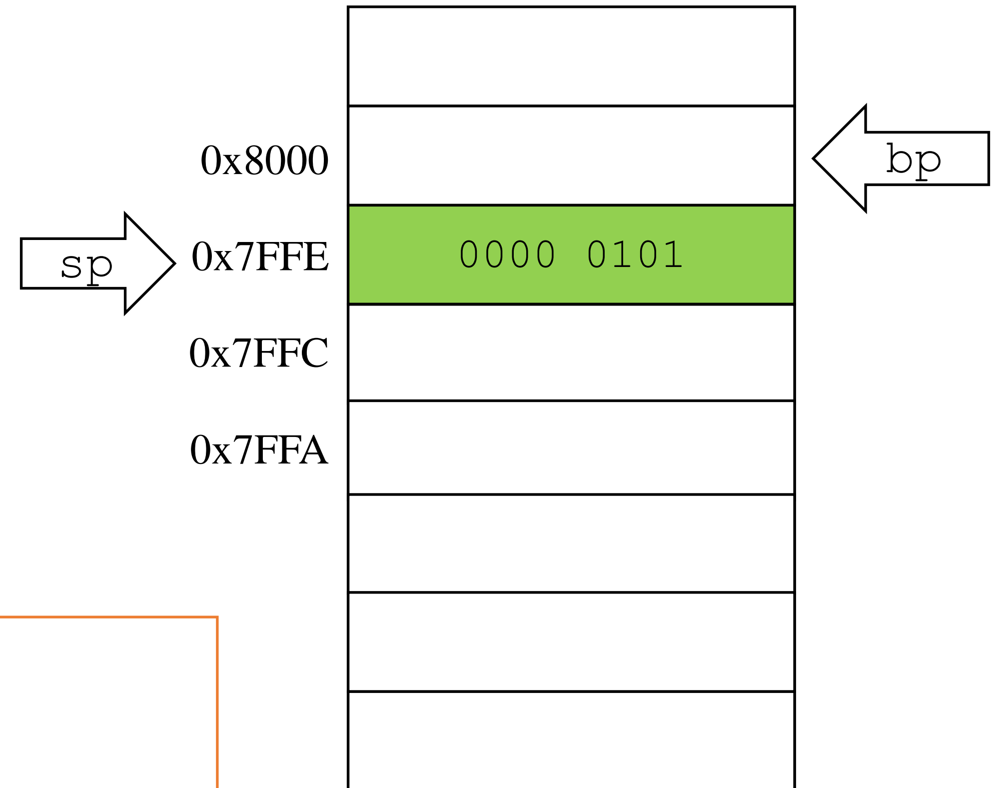
Stack is simple solution to a CPU inconvenience:

The limited number of registers.

Two simple instructions: `push` and `pop`.

Two special registers: `bp` and `sp`.

Stack grows downwards.




```
; push

push 'A'
```

Stack

Is stack something special and complicated thing?

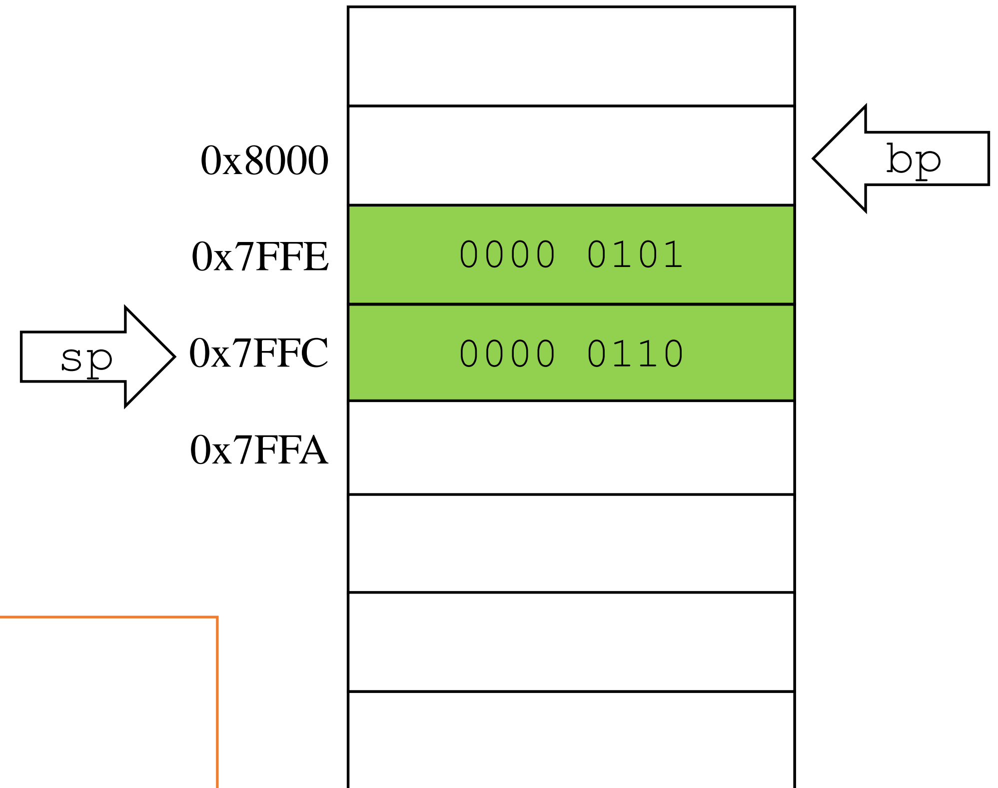
Stack is simple solution to a CPU inconvenience:

The limited number of registers.

Two simple instructions: `push` and `pop`.

Two special registers: `bp` and `sp`.

Stack grows downwards.




```
; push

push 'A'
push 'B'
```

Stack

Is stack something special and complicated thing?

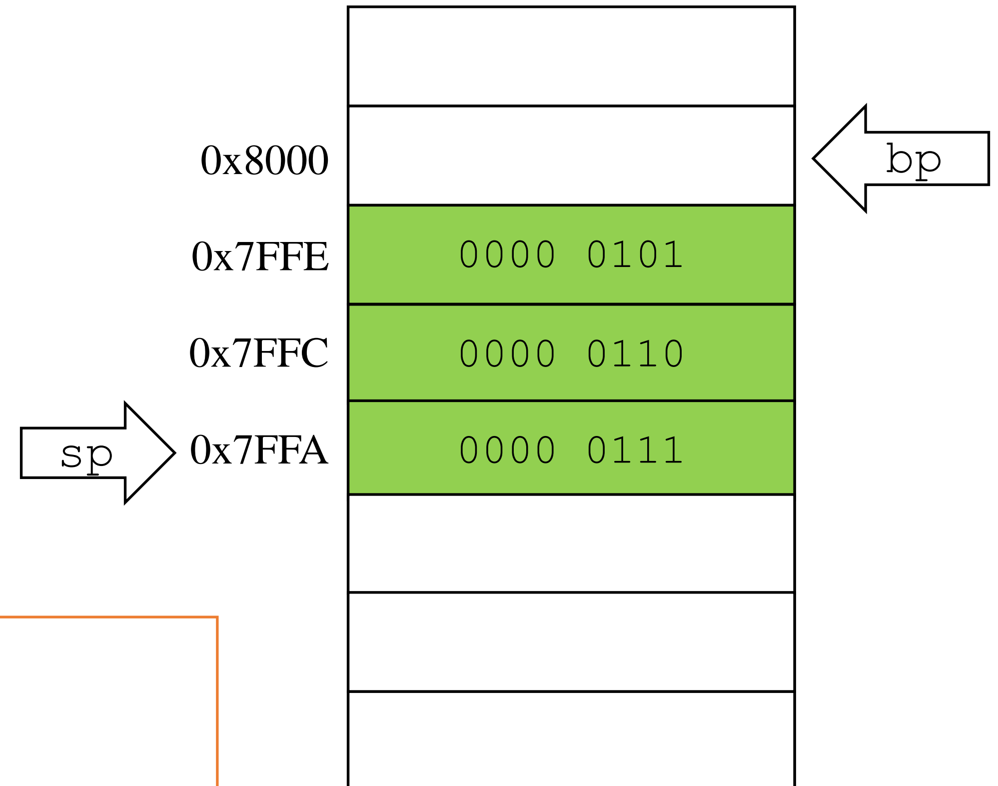
Stack is simple solution to a CPU inconvenience:

The limited number of registers.

Two simple instructions: `push` and `pop`.

Two special registers: `bp` and `sp`.

Stack grows downwards.




```
; push

push 'A'
push 'B'
push 'C'
```

Stack

```
; Stack

[org 0x7c00]

mov ah, 0x0e

mov bp, 0x8000
mov sp, bp


push 'A'

push 'B'

push 'C'

pop bx
mov al, bl
int 0x10
```

output 1?

```
mov al, [0x7ffc]
int 0x10

jmp $

times 510 - ($ - $$) db 0

dw 0xaa55
```

output 2?

Stack

- `pusha`
- `popa`

push all registers into stack and pop all registers back in their corresponding registers.

Compare and Jump

- `cmp`

compare a register to another register or immediate value.

- `je   target      ; jump if equal                  (x == y)`
- `jne  target      ; jump if not equal              (x != y)`
- `jl   target      ; jump if less than              (x < y)`
- `jle  target      ; jump if less than or equal     (x <= y)`
- `jg   target      ; jump if greater                (x > y)`
- `jge  target      ; jump if greater or equal       (x >= y)`

Call functions

How can we go to an address and then return to the previous address of execution?

Using labels?

`call` and `ret` instructions.

Include

```
%include “name_of_file.asm”
```

Example: Printer function

```
; printer.asm

print_string:

    mov ah, 0x0e
    start1:
        mov al, [bx]
        cmp al, 0x0
        je end1
        int 0x10
        inc bx
        jmp start1

    end1:
        ret
```

```
; main.asm
[org 0x7c00]

; init stack
mov bp, 0x800
mov sp, bp

mov bx, boot_msg
call print_string

jmp $

%include "printer.asm"

boot_msg:
        db "Booting loader", 0


times   510 - ($-$$) db 0

dw 0xaa55
```

Questions?

?