# x86 HW1

2019. 05. 10

Embedded System Lab
Sungkyunkwan Univ.

# Contents

- 1st Homework
  - NASM Instruction
  - NASM Syntax
  - Boot Strap
  - Print on Screen
  - x86 HW1

# NASM Instruction

- Instruction used in example
  - MOV Register(EAX), Data-value
    - Store the data in register(EAX).
  - ADD/SUB Register(EAX), Data-value
    - Add data value to register(EAX) / Subtract data-value from register
    - Store the result to Register(EAX)
  - XOR Register(EAX), Data-value
    - MOV EAX, 0xF0F0
    - XOR EAX, 0xFFFF → EAX : 0x0F0F
  - CMP Register(EAX), Data-value
    - Compare register value with data-value
  - JMP Label's name
    - Move to Label's location from the present address
    - Ex) Loop: ADD EAX, 0xF0F0
      JMP Loop

# NASM Instruction

- Instruction used in example(con't)
  - JE Label's name
    - Compare data value and register value
    - JMP when the CMP's result is same
    - Ex) LOOP: MOV EAX, 0xF0
      CMP EAX, 0xFF
      JE LOOP
  - JNE Label's name
    - JMP when the CMP's result is not same
    - Ex) LOOP: MOV EAX, 0xF0
      CMP EAX, 0xFF
      JNE LOOP

# NASM Instruction

- Instruction used in example(con't)
  - CALL/RET
    - CALL Label's name
      - Jump to Label's name
    - RET
      - Return to IP(Instruction Point)

    - Ex)

```
        XOR EAX, EAX
        CALL LOOP
        ADD EAX, 0x01
LOOP:   MOV EAX, 0xF0
        RET
```

→ EAX : 0xF1

# NASM Instruction

## Instruction used in example(con't)

| 명령어 | 분기 조건 | 설명 |
|---|---|---|
| JA/JNBE | C = 0 and Z = 0 | Jump Above / Jump if Not Below or Equal to |
| JAE/JNB | C = 0 | Jump Above or Equal to / Jump if Not Below |
| JB/JNAE | C = 1 | Jump Below / Jump if Not Above or Equal to |
| JBE/JNA | C = 1 or Z = 1 | Jump Below or Equal to / Jump if Not Above |
| JC | C = 1 | Jump if Carry |
| JE/JZ | Z = 1 | Jump Equal to / Jump Zero |
| JG/JNLE | Z = 0 and S = O | Jump Greater than / Jump Not Less or Equal |
| JGE/JNL | S = O | Jump Greater than or Equal / Jump Not Less |
| JL/JNGE | S ≠ O | Jump Less than / Jump Not Greater or Equal |
| JLE/JNG | Z = 1 or S ≠ O | Jump Less than or Equal / Jump Not Greater |
| JNC | C = 0 | Jump if Not Carry |
| JNE/JNZ | Z = 0 | Jump Not Equal to / Jump Not Zero |
| JNO | O = 0 | Jump No Overflow |
| JNS | S = 0 | Jump No Sign |
| JNP/JPO | P = 0 | Jump No Parity / Jump Parity Odd |
| JO | O = 1 | Jump on Overflow |
| JP/JPE | P = 1 | Jump on Parity / Jump Parity Even |
| JS | S = 1 | Jump on Sign |
| JCXZ | CX=0 | Jump if CX=0 |

# NASM Syntax

## Move Instructions

### Allowed Case

- mov ecx, [ebx+edi] ; double word (by EBX+EDI) in data segment
- mov eax, [esi-4] ; double word (by ESI-4) in data segment
- mov eax, [bar] ; double word contents of bar in data segment
- mov eax, [es:edi] ; double word (by es:edi) in extra segment
  - [es:edi] → base address and offset address
- mov byte[es:edi], al ; store al value to memory address [es:edi]
- mov eax, bar ; double word address of bar in data segment

### Not allowed Case

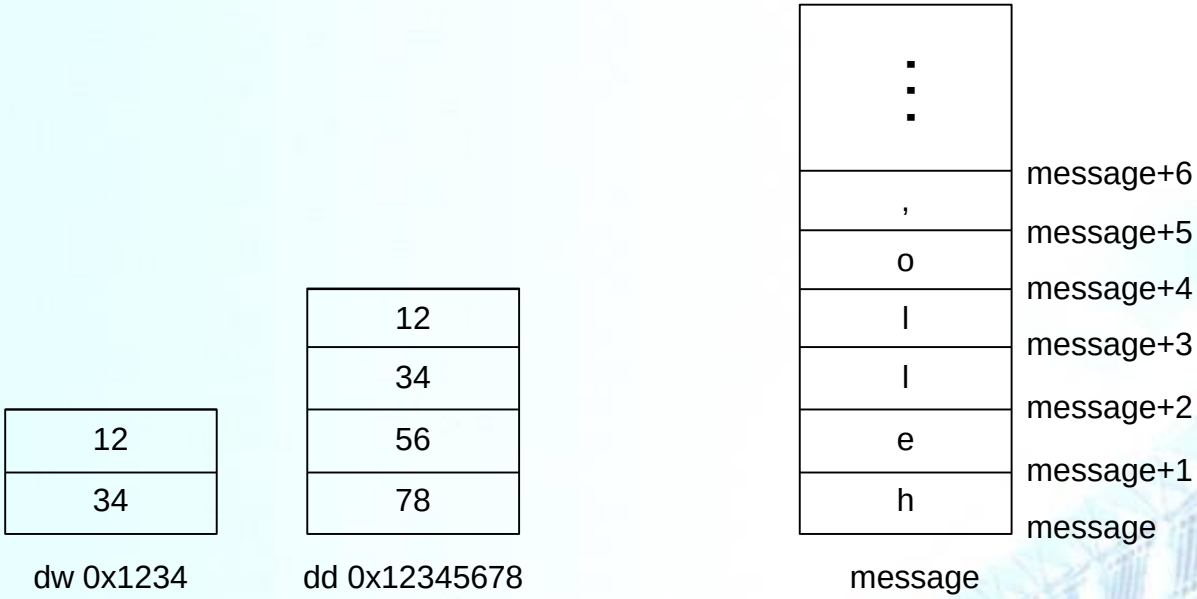
- mov [bx], [si] ; not allowed, mem-to-mem
- mov es, ds ; not allowed, (seg reg)-to-(seg reg)
- mov bl, bx ; not allowed, different size

# NASM Syntax

- Mov Instructions
  - NASM does not support the hybrid syntaxes such as
    - mov eax, table[ebx] ; error
    - mov eax, [table+ebx] ; ok
    - mov eax, [es:edi] ; ok
  - NASM does NOT remember variable types
    - data dw 0 ; data type defined as double word
    - mov [data], 2 ; doesn't work
    - mov word [data], 2 ; ok
  - There are many cases.
  - You can refer lecture note "Summary_General_Purpose_Ins"

# NASM Syntax

- **Storage directives (little endian)**
  - **db, dw, dd are used for initialized data in data segment**
    - **db 0x55** ; the byte 0x55
    - **dd 0x12345678** ; 0x78 0x56 0x34 0x12
    - **message db ‘hello, world’** ; data type defined as byte
      ; put ‘hello, world’ (ASCII) in message
  - **EQU defines a symbol to a constant**
    - **message equ 0x01**

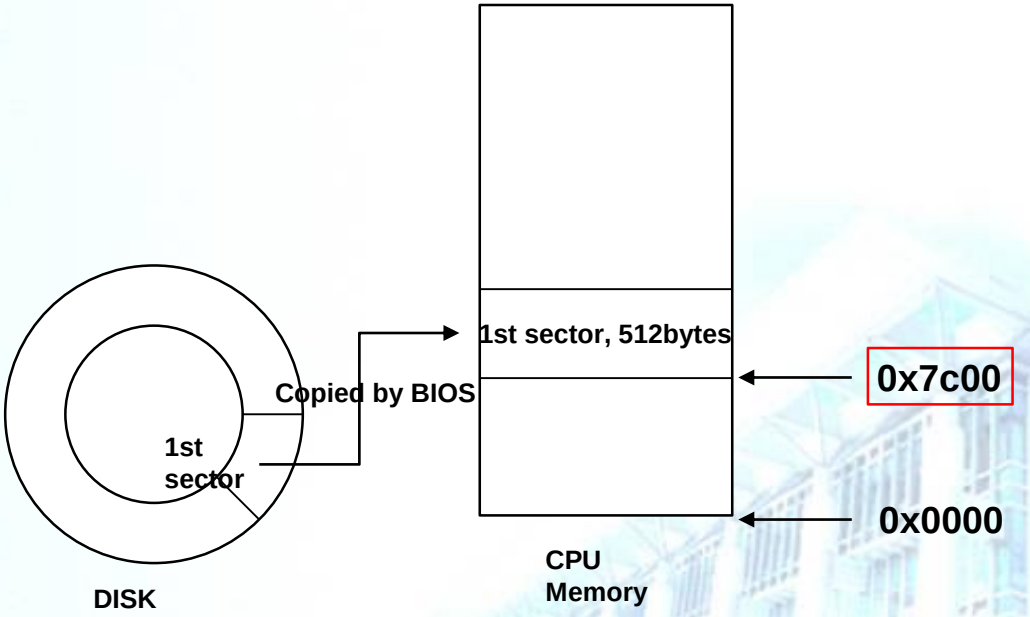


# Boot Strap

**Bios** → **Bootloader** → **OS** (Program)

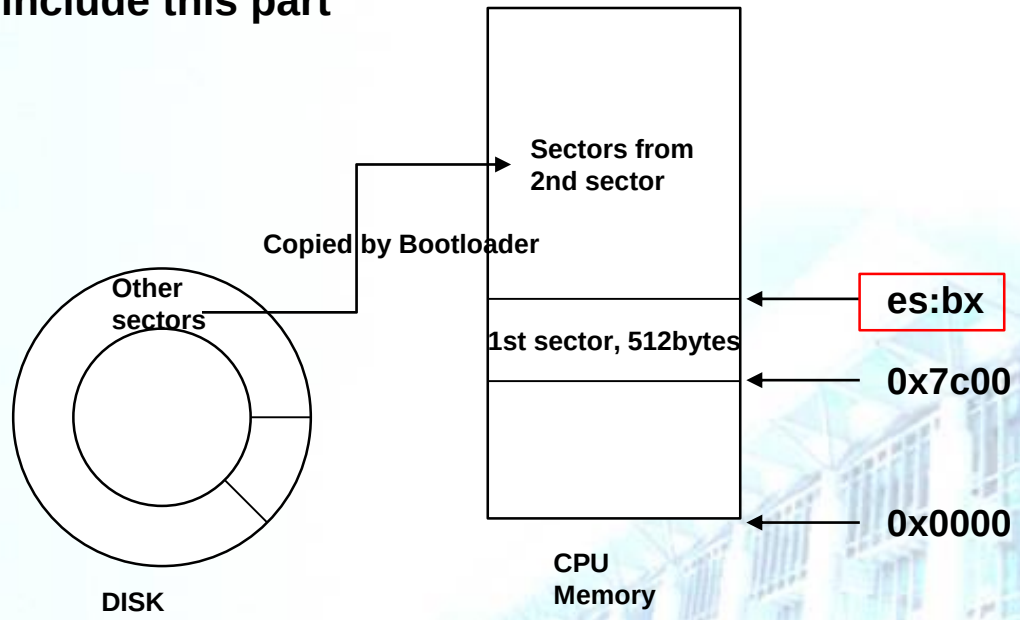
# Boot Strap

- **BIOS**
  - When PC turns on, BIOS controls PC at first
  - Read the first sector(MBR) of disk
  - Size of one sector is 512bytes
  - Load it to 0x7c00(CPU Memory Address)
  - This process is automatically done by BIOS
    - Source code doesn't include this part



# Boot Strap

- **Boot Loader**
  - Generally, the size of OS(Program) is larger than 512bytes
    - Need to load other sectors
  - Run the code in first Sector [org 0x7c00]
  - Bootloader code reads and loads next Sectors of DISK
    - Load it to es:bx (base:offset) memory address
  - This process is done by Boot Loader
    - Source code should include this part



# Boot Strap

## Boot Loader

### Load sectors

```
call load_sectors
jmp sector_2

load_sectors:

        push es

        xor ax, ax
        mov es, ax
        mov bx, sector_2
        mov ah,2
        mov al,(sector_end - sector_2)/512 + 1
        mov ch,0
        mov cl,2
        mov dh,0
        mov dl,0
        int 0x13

        pop es
        ret
```

Load sectors to Memory address sector_2 (es:bx)

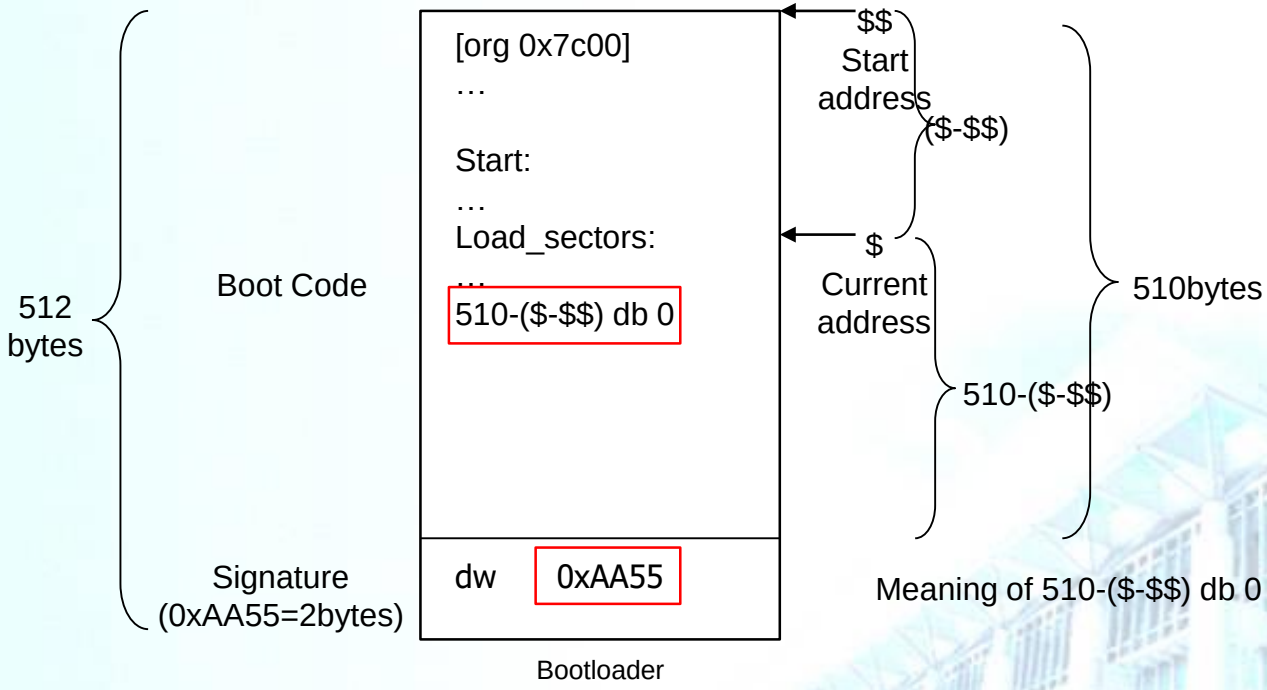
**INT 13h AH=02h: Read Sectors From Drive**

Parameters

| | |
|---|---|
| AH | 02h |
| AL | Sectors To Read Count |
| CH | Track |
| CL | Sector |
| DH | Head |
| DL | Drive |
| ES:BX | Buffer Address Pointer |

# Boot Strap

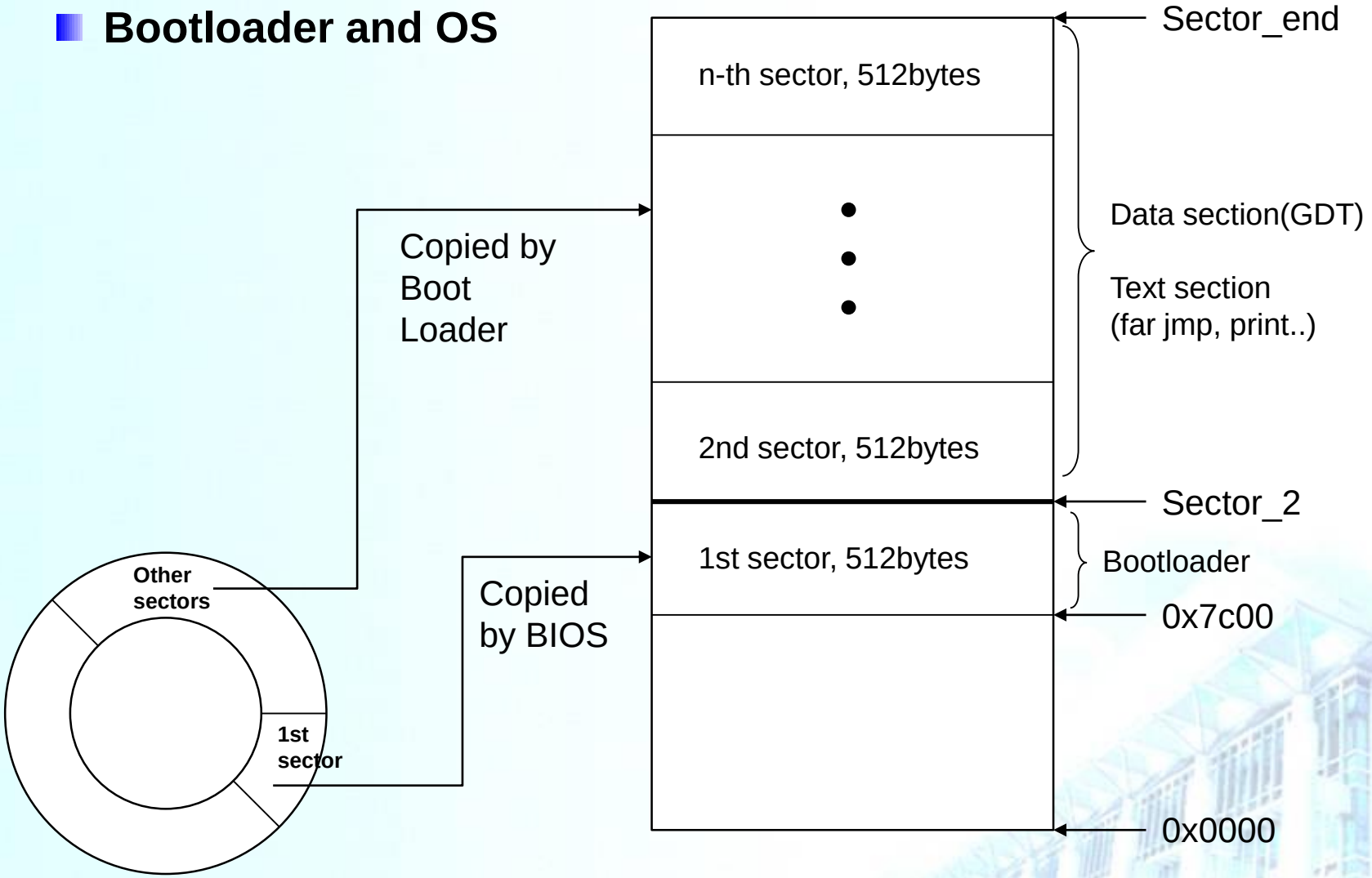
- **Boot Loader**
  - **Size of Bootloader is also 512bytes**
  - **It can include boot code**
  - **It must have signature bytes end of section**
    - **→ Signature bytes distinguish bootloader from other sector**



# Boot Strap

## Memory Structure of Source Code

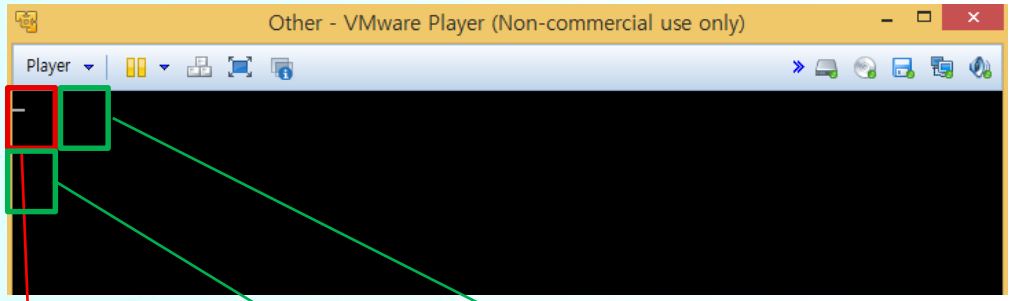
### Bootloader and OS



# Print on Screen

## Memory Address of VMware Screen

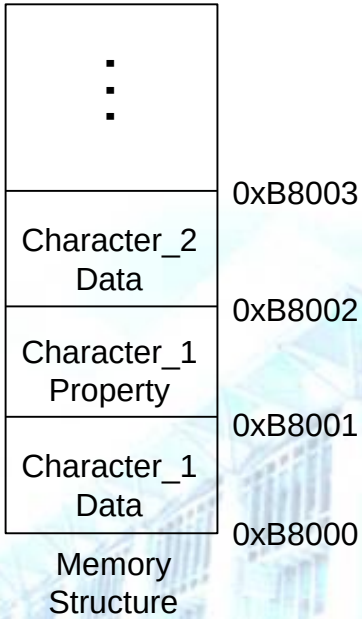
- Use two bytes for print a character
  - One byte for character it self
  - Other byte for Text Property
    - Upper 4bit : Background Color
    - Lower 4bit : Character Color
  - Printing position on screen
    - Square of each row(80)*2*(row number)+2*(column number)




Address of Video Memory is 0xB8002
Position of character : 80*2*0+2*1

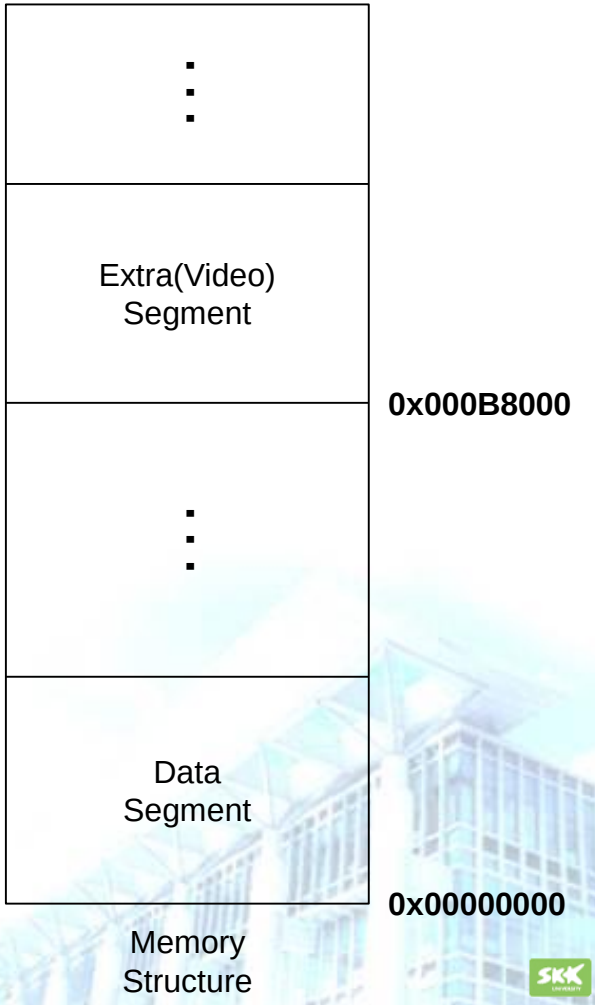
Position of character : 80*2*1+2*0

Start Address of Video Memory in VMware
→ 0xB8000



# Print on Screen

- **Print characters in Real Mode**
  - **Store the start address of Video Memory directly on ES Register**
    - **Access memory address by using a segment register and offsets**
    - **Memory address**
      **= Segment Register*10H+Offset Address**
      - **Start address of memory in VMware is 0xB8000**
      - **Print characters on screen using Start address and offsets address**



# Print on Screen

- **Memory Structure**
  - **Print a string on Screen**
    - Data is stored in Data Segment
      - Use Data Segment Register and BX / DI / SI / 16-bit number
    - Data type defined as byte
      - Store your ID in ‘ID’ variable
      - Store your NAME in ‘NAMEE’ variable
    - Store a character(byte) in string in the Video Segment
  - **Print a value(word size) in memory on Screen**
    - Values are expressed in hexadecimal value
    - Convert the value to ASCII code
    - Stack Pointer is in the Stack Segment
      - Use Stack Segment Register and SP / BP
    - Not allowed using mem-to-mem or (seg/reg)-to(seg/reg)

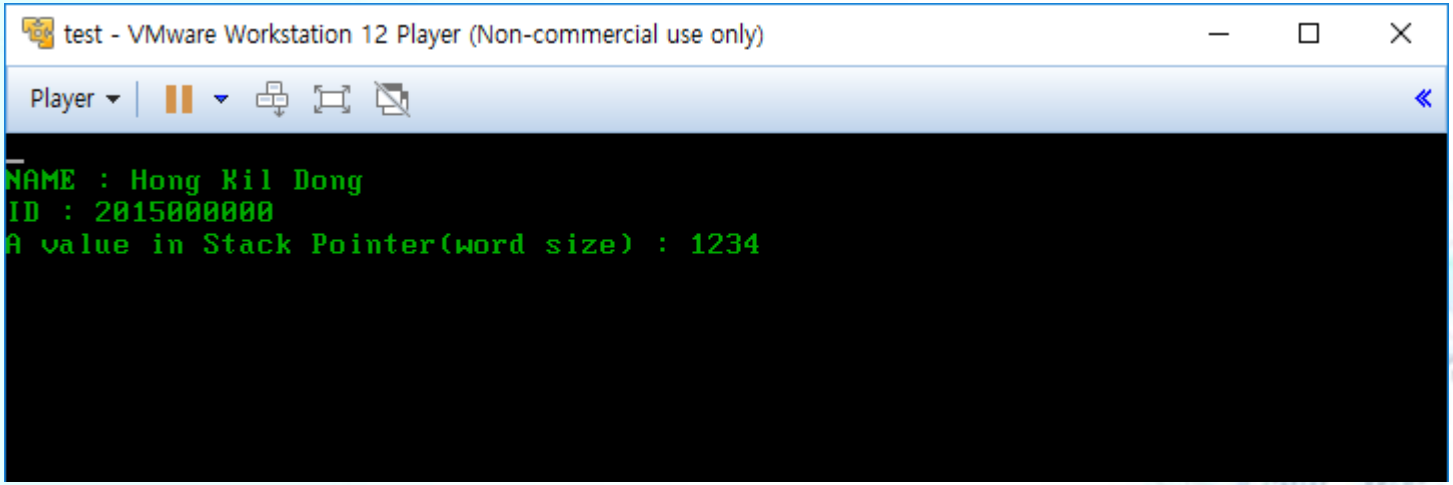
# x86 HW1

## 1st Homework

### Print characters in Real Mode

- Print your NAME numbers on first line
- Print your ID on second line
- Print the value in the stack pointer on third line

- No restriction about positions and formats
- No restriction about color of characters



# x86 HW1

- Time and Place
  - May 17th(Fri) 19:00
  - Semi-conductor building 2 floor computer room
    - 400212, 400202

- How to submit
  - .asm and .bin files
  - I-Campus, until May 17th 18:59
    - format
      - 2018000000_HW1.asm
      - 2018000000_HW1.bin