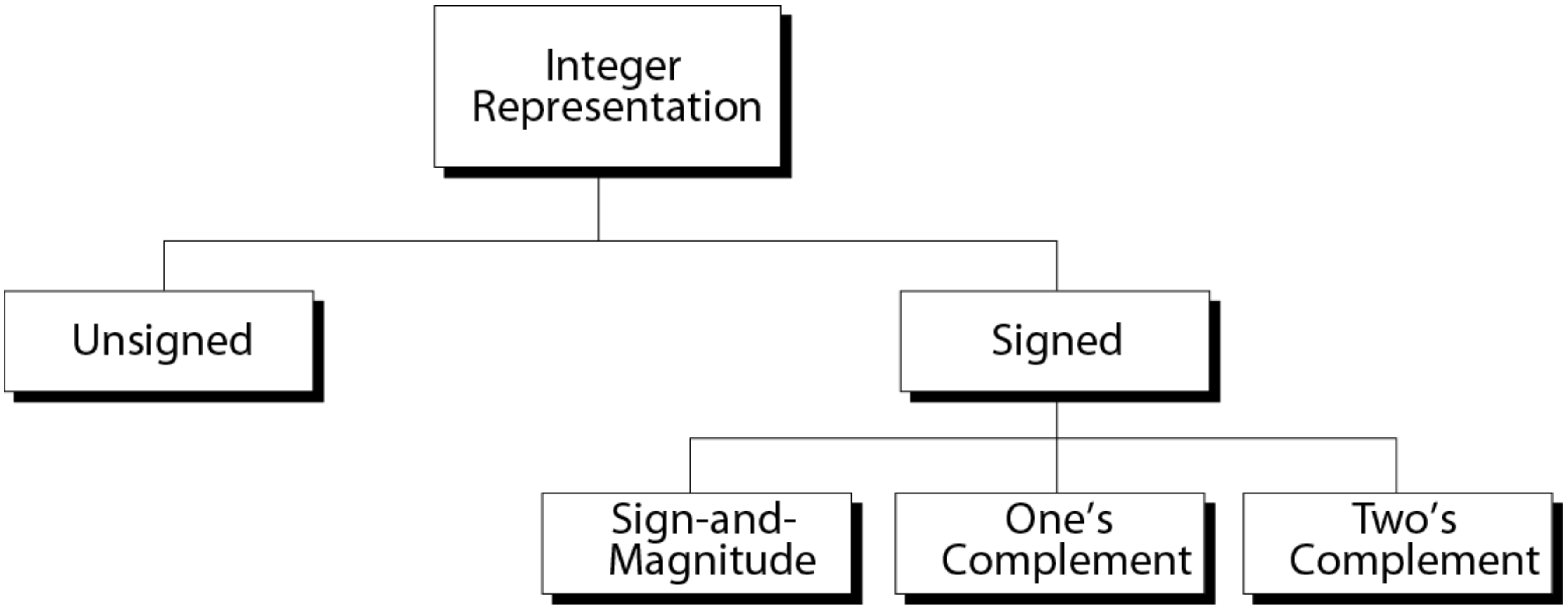
# CYBV 471 Assembly Programming for Security Professionals
# Week 2

## Integer Presentation, Boolean Operations, x86 Processor Architecture

# Agenda

- **Integer Number Representation**
  - Unsigned Integer presentation
  - Signed Integer presentation
  - One’s complement
  - Two’s complement
  - Binary Addition
  - Binary Subtraction
- **Boolean Operations**
- **Integer Storage Size**
- **ASCII Presentation**
- **Unicode Standard**
- **The x86 Architecture**
  - CPU Components (Clock, ALU, Registers)
  - Bus (Address Bus, Control Bus, data bus_
  - I/O Devices
  - General Purpose Registers

# Integer Number Representation



# Unsigned Integers

- Unsigned Integers: Positive or zero
- n-bits can have a value between 0 and ($2^n$ – 1), ($2^n$ different values)
- For 8 bits, Max. positive value is ($2^8$ – 1) = 255
- For 16 bits, Max. positive value is ($2^{16}$ – 1) = 65535

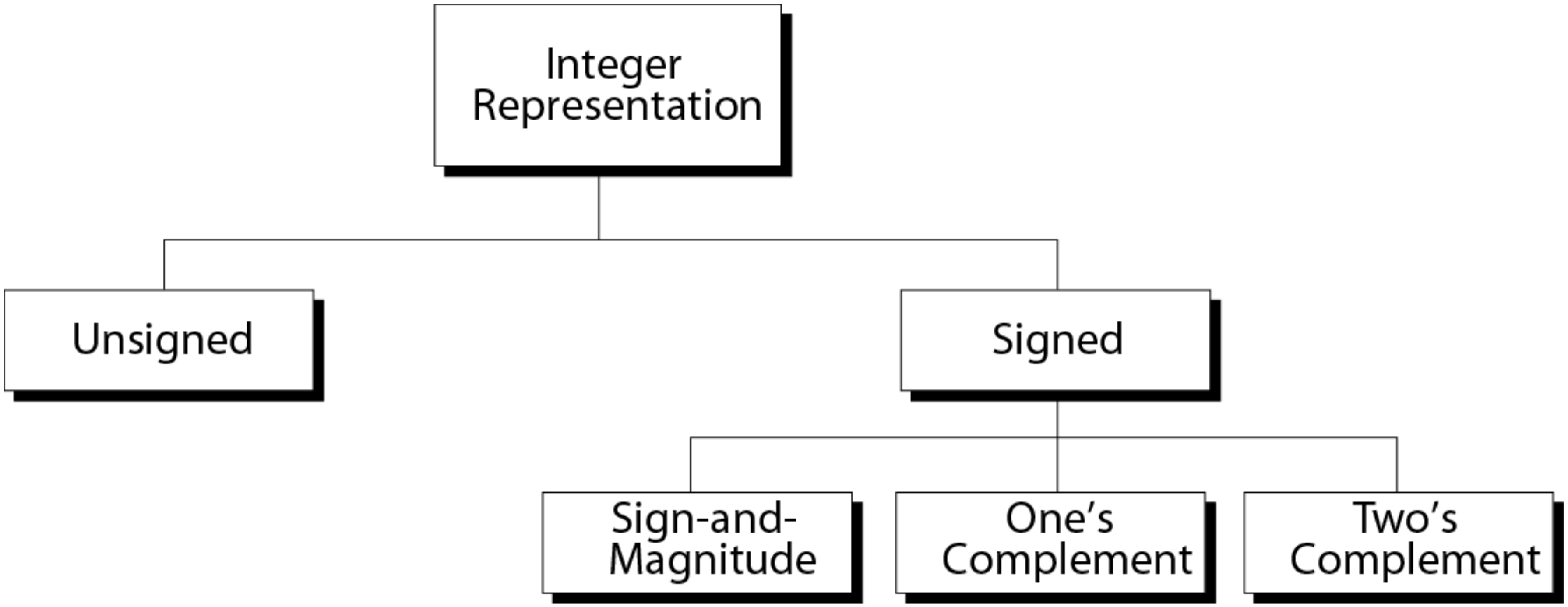
- Example-1: How a number “7” is stored in 8-bit memory location?
- Solution: Number “7” in binary is 111
- To store it in 8-bit memory location, add five zeroes to the left of 111
- Therefore, “7” will be stored in memory as “00000111”

# Unsigned Integers

- Example-2: Can you store "258" in 8-bit memory location?
- Solution: Number "258" in binary is ***100000010 (9 bits)***
- ***You can't store 258 in 8-bit memory location.***
- ***We will have overflow condition***
- To store 258 in 8-bit memory, you need two 8-bit locations (16 bits)
- Therefore, "258" will be stored in memory as "***0000000100000010***"

# Signed Integers

- Signed Integers:
  - Positive, zero, or negative values.
  - Negative values are stored as 2's complement (MSB is sign value)



# Sign-and-magnitude format

**The leftmost bit defines the sign of the number.**

- **If it is 0, the number is positive.**
- **If it is 1, the number is negative.**

- Example: 127 is represented as 0 1111111
- -127 is represented as 1 1111111
- Notice that, for 8 bits, the range of stored numbers in 8-bit will be from (-127) to (127) since we reserve one bit for the sign.
- For 16-bit, the range is –32767 to 32767
- PS: The sign-and-magnitude representation is not used to store signed numbers by computers.
- Operations such adding and subtracting are not straightforward for this representation.

# One’s complement

**The leftmost bit defines the sign of the number.**

**If it is 0, the number is positive.** No action needed

**If it is 1, the number is negative. Take action**

If a number is negative, convert the bits

- If the value of a bit is 0, change it to 1
- If the value of a bit is 1, change it to 0

# One's complement

Example: Store –258 in a 16-bit memory location using one's complement representation

First change the number (258) to binary 100000010.

Add seven 0s to make a total of N (16) bits, 0000000100000010.

2- Convert all bits

3- The result is: ***1111111011111101***

4- Note that the leftmost bit defines the sign of the number

# One’s complement

Example: Store 258 in a 16-bit memory location using one’s complement representation

First change the number (258) to binary 100000010.

Add seven 0s to make a total of N (16) bits, 0000000100000010.

2- Since it is positive number, no action

3- The result is: 0000000100000010

4- Note that the leftmost bit defines the sign of the number

One’s complement representation is not used to store numbers in computers today.

# Two's complement

- **Two's complement** is what computer use to represent signed integers.
- In two's complement representation, the **leftmost bit** defines the sign of the number.

If it is **0**, the number is **positive**. No Action

The value is straight forward

If it is **1**, the number is **negative.**

**What is the value?**

Apply 2's complement steps

# Two’s complement

- To calculate the 2’s complement of a negative number, follow the following steps:

 1- Write the magnitude of the number in binary (ignore the sign)

 2- Add 0s to the left of the binary number to make it “n” bits

 2- Calculate the 1’s complement (convert 1 to 0, and convert 0 to 1)

 3- Add 1 to the new binary one

  Use the following adding rules

  $0 + 0 = 0$     $0 + 1 = 1$     $1 + 0 = 1$

  $1 + 1 = 0$, carry 1 (ignore it) ($1 + 1 =2$. And 2 in binary is 10)

# Two's complement

Example: What is the 4-bit binary representation for "-5"

1- Write the magnitude of the number in binary (ignore the sign)

5 in binary is 0101

2- Calculate the 1's complement: 1010

3- Add 1 to the new binary one

```
  1010
+ 0001
-------------
 1011
```

"-5" in 2's complement is 1011. Notice the most significant bit is 1

# Two's complement

Example: Store +7 in an 8-bit memory location using two's complement representation.

First change the number to binary 111.

Add five  0s to make a total of N (8) bits, 00000111.

The sign is positive, so no more action is needed.

The result is: 00000111

# Two’s complement

Examples: Convert (-13) to two’s complement representation using 8-bit representation

| | |
|---|---|
| 00001101 | base integer |
| 11110010 | 1's complement |
| +1 | |
| 11110011 | 2's complement |

For -227, 12-bit representation

| | |
|---|---|
| 000011100011 | base integer |
| 111100011100 | 1's complement |
| +1 | |
| 111100011101 | 2's complement |

Example: Interpret 11110110 in decimal if the number was stored as a two's complement integer.

Solution: The leftmost bit is 1. Therefore, the number is negative.

To get the value of the number, implement the 2's complement for the all bits.

1- Implement one's complement -> 00001001

2- Add 1, the result is 00001010

3- The decimal value is 10

4- Since the number is negative, the actual number is -10

# Two's Complement Operation

Perform the following subtraction operation in *the Binary Number System*:

**10 – 12**= $X_2$? (represent the result in binary format)

Solution: 10 – 12 = 10 + (-12)

$10_{10} = 01010_2$

Since (-12) is negative, we use two's complement to represent it.

$12_{10} = 01100_2$. One's complement = 10011

Two's complement = 10011 + 1 = 10100.

Perform the addition: 10 – 12 = 10 + (-12)

| | | | | | | Decimal |
|---|---|---|---|---|---|---|
| 10 | 0 | 1 | 0 | 1 | 0 | 10 |
| + (-12) | 1 | 0 | 1 | 0 | 0 | -12 |
| = 1110 | 1 | 1 | 1 | 1 | 0 | -2 |

Result = 1110. Notice that the sign bit is "1". It is negative number

To get the magnitude value, perform two's operation again.

1110 -> 0001 > (+1) -> 0010 (= 2). The final answer is (-2)

# Binary Addition

- Starting with the LSB, add each pair of digits, include the carry if present.

| A | B | Sum | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

carry: 1

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | (4) |
| + | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | (7) |
| | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | (11) |
| bit position: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

# Binary Subtraction

Perform the following subtraction operation in *the Binary Number System.*
**10 – 12**= $X_2$? (represent the result in binary format)
Solution: 10 – 12 = 10 + (-12)
$10_{10} = 01010_2$
Since (-12) is negative, we use two's complement to represent it.
$12_{10} = 01100_2$.   One's complement = 10011
Two's complement = 10011 + 1 = 10100.
Perform the addition: 10 – 12 = 10 + (-12)

| | | | | | | Decimal |
|---|---|---|---|---|---|---|
| 10 | 0 | 1 | 0 | 1 | 0 | 10 |
| + (-12) | 1 | 0 | 1 | 0 | 0 | -12 |
| = 1110 | 1 | 1 | 1 | 1 | 0 | -2 |

Result = 1110. Notice that the sign bit is "1". It is negative number

To get the magnitude value, perform two's operation again.

1110 -> 0001 > (+1) -> 0010  (= 2). The final answer is (-2)

# Boolean Operations

- Not operation
- OR operation
- AND operation

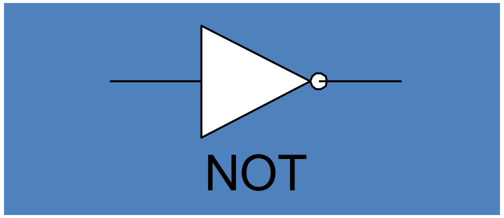
| Expression | Description |
|---|---|
| $\neg X$ | NOT X |
| $X \wedge Y$ | X AND Y |
| $X \vee Y$ | X OR Y |
| $\neg X \vee Y$ | ( NOT X ) OR Y |
| $\neg(X \wedge Y)$ | NOT ( X AND Y ) |
| $X \wedge \neg Y$ | X AND ( NOT Y ) |

# NOT

- Inverts (reverses) a Boolean value
- Truth table for Boolean NOT operator:

| X | ¬X |
|---|---|
| F | T |
| T | F |

Digital gate diagram for NOT:

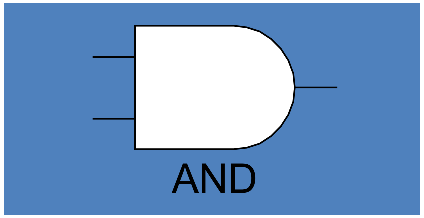


# AND

- Truth table for Boolean AND operator:

| X | Y | X ∧ Y |
|---|---|---|
| F | F | F |
| F | T | F |
| T | F | F |
| T | T | T |

Digital gate diagram for AND:

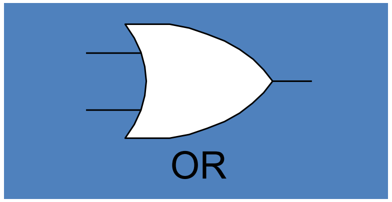


# OR

- Truth table for Boolean OR operator:

| X | Y | X ∨ Y |
|---|---|---|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | T |

Digital gate diagram for OR:



# Operator Precedence

- Examples showing the order of operations:

| Expression | Order of Operations |
| --- | --- |
| $\neg X \vee Y$ | NOT, then OR |
| $\neg(X \vee Y)$ | OR, then NOT |
| $X \vee (Y \wedge Z)$ | AND, then OR |

# Integer Storage Sizes

Standard sizes:

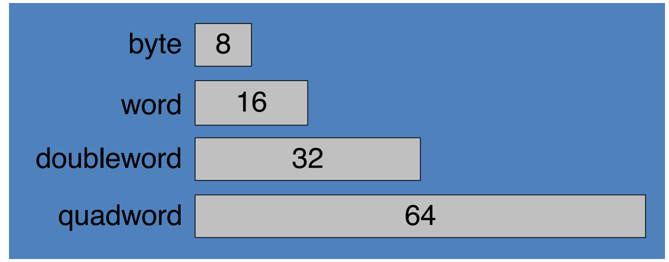



**Table 1-4** Ranges of Unsigned Integers.

| Storage Type | Range (low–high) | Powers of 2 |
|---|---|---|
| Unsigned byte | 0 to 255 | 0 to $(2^8 - 1)$ |
| Unsigned word | 0 to 65,535 | 0 to $(2^{16} - 1)$ |
| Unsigned doubleword | 0 to 4,294,967,295 | 0 to $(2^{32} - 1)$ |
| Unsigned quadword | 0 to 18,446,744,073,709,551,615 | 0 to $(2^{64} - 1)$ |

What is the largest unsigned integer that may be stored in 20 bits?

# ASCII Presentation

- It is an acronym for the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.
- The purpose of ASCII was to provide a standard to code various symbols ( visible and invisible symbols)
- In the **ASCII character set**, each binary value between 0 and 127 represents a specific character.
- Most computers extend the ASCII character set to use the full range of 256 characters available in a byte. The upper 128 characters handle special things like accented characters from common foreign languages.
- For example, “Load” in ASCII is:

| | | | |
|---|---|---|---|
| L | o | a | d |
| 76 | 111 | 97 | 100 |
| 01001100 | 01101111 | 01100001 | 01100100 |

# ASCII TABLE

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | [NULL] |
| 1 | 1 | 1 | 1 | [START OF HEADING] |
| 2 | 2 | 10 | 2 | [START OF TEXT] |
| 3 | 3 | 11 | 3 | [END OF TEXT] |
| 4 | 4 | 100 | 4 | [END OF TRANSMISSION] |
| 5 | 5 | 101 | 5 | [ENQUIRY] |
| 6 | 6 | 110 | 6 | [ACKNOWLEDGE] |
| 7 | 7 | 111 | 7 | [BELL] |
| 8 | 8 | 1000 | 10 | [BACKSPACE] |
| 9 | 9 | 1001 | 11 | [HORIZONTAL TAB] |
| 10 | A | 1010 | 12 | [LINE FEED] |
| 11 | B | 1011 | 13 | [VERTICAL TAB] |
| 12 | C | 1100 | 14 | [FORM FEED] |
| 13 | D | 1101 | 15 | [CARRIAGE RETURN] |
| 14 | E | 1110 | 16 | [SHIFT OUT] |
| 15 | F | 1111 | 17 | [SHIFT IN] |
| 16 | 10 | 10000 | 20 | [DATA LINK ESCAPE] |
| 17 | 11 | 10001 | 21 | [DEVICE CONTROL 1] |
| 18 | 12 | 10010 | 22 | [DEVICE CONTROL 2] |
| 19 | 13 | 10011 | 23 | [DEVICE CONTROL 3] |
| 20 | 14 | 10100 | 24 | [DEVICE CONTROL 4] |
| 21 | 15 | 10101 | 25 | [NEGATIVE ACKNOWLEDGE] |
| 22 | 16 | 10110 | 26 | [SYNCHRONOUS IDLE] |
| 23 | 17 | 10111 | 27 | [ENG OF TRANS. BLOCK] |
| 24 | 18 | 11000 | 30 | [CANCEL] |
| 25 | 19 | 11001 | 31 | [END OF MEDIUM] |
| 26 | 1A | 11010 | 32 | [SUBSTITUTE] |
| 27 | 1B | 11011 | 33 | [ESCAPE] |
| 28 | 1C | 11100 | 34 | [FILE SEPARATOR] |
| 29 | 1D | 11101 | 35 | [GROUP SEPARATOR] |
| 30 | 1E | 11110 | 36 | [RECORD SEPARATOR] |
| 31 | 1F | 11111 | 37 | [UNIT SEPARATOR] |
| 32 | 20 | 100000 | 40 | [SPACE] |
| 33 | 21 | 100001 | 41 | ! |
| 34 | 22 | 100010 | 42 | " |
| 35 | 23 | 100011 | 43 | # |
| 36 | 24 | 100100 | 44 | $ |
| 37 | 25 | 100101 | 45 | % |
| 38 | 26 | 100110 | 46 | & |
| 39 | 27 | 100111 | 47 | ' |
| 40 | 28 | 101000 | 50 | ( |
| 41 | 29 | 101001 | 51 | ) |
| 42 | 2A | 101010 | 52 | * |
| 43 | 2B | 101011 | 53 | + |
| 44 | 2C | 101100 | 54 | , |
| 45 | 2D | 101101 | 55 | - |
| 46 | 2E | 101110 | 56 | . |
| 47 | 2F | 101111 | 57 | / |

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 48 | 30 | 110000 | 60 | 0 |
| 49 | 31 | 110001 | 61 | 1 |
| 50 | 32 | 110010 | 62 | 2 |
| 51 | 33 | 110011 | 63 | 3 |
| 52 | 34 | 110100 | 64 | 4 |
| 53 | 35 | 110101 | 65 | 5 |
| 54 | 36 | 110110 | 66 | 6 |
| 55 | 37 | 110111 | 67 | 7 |
| 56 | 38 | 111000 | 70 | 8 |
| 57 | 39 | 111001 | 71 | 9 |
| 58 | 3A | 111010 | 72 | : |
| 59 | 3B | 111011 | 73 | ; |
| 60 | 3C | 111100 | 74 | < |
| 61 | 3D | 111101 | 75 | = |
| 62 | 3E | 111110 | 76 | > |
| 63 | 3F | 111111 | 77 | ? |
| 64 | 40 | 1000000 | 100 | @ |
| 65 | 41 | 1000001 | 101 | A |
| 66 | 42 | 1000010 | 102 | B |
| 67 | 43 | 1000011 | 103 | C |
| 68 | 44 | 1000100 | 104 | D |
| 69 | 45 | 1000101 | 105 | E |
| 70 | 46 | 1000110 | 106 | F |
| 71 | 47 | 1000111 | 107 | G |
| 72 | 48 | 1001000 | 110 | H |
| 73 | 49 | 1001001 | 111 | I |
| 74 | 4A | 1001010 | 112 | J |
| 75 | 4B | 1001011 | 113 | K |
| 76 | 4C | 1001100 | 114 | L |
| 77 | 4D | 1001101 | 115 | M |
| 78 | 4E | 1001110 | 116 | N |
| 79 | 4F | 1001111 | 117 | O |
| 80 | 50 | 1010000 | 120 | P |
| 81 | 51 | 1010001 | 121 | Q |
| 82 | 52 | 1010010 | 122 | R |
| 83 | 53 | 1010011 | 123 | S |
| 84 | 54 | 1010100 | 124 | T |
| 85 | 55 | 1010101 | 125 | U |
| 86 | 56 | 1010110 | 126 | V |
| 87 | 57 | 1010111 | 127 | W |
| 88 | 58 | 1011000 | 130 | X |
| 89 | 59 | 1011001 | 131 | Y |
| 90 | 5A | 1011010 | 132 | Z |
| 91 | 5B | 1011011 | 133 | [ |
| 92 | 5C | 1011100 | 134 | \ |
| 93 | 5D | 1011101 | 135 | ] |
| 94 | 5E | 1011110 | 136 | ^ |
| 95 | 5F | 1011111 | 137 | _ |

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 96 | 60 | 1100000 | 140 | ` |
| 97 | 61 | 1100001 | 141 | a |
| 98 | 62 | 1100010 | 142 | b |
| 99 | 63 | 1100011 | 143 | c |
| 100 | 64 | 1100100 | 144 | d |
| 101 | 65 | 1100101 | 145 | e |
| 102 | 66 | 1100110 | 146 | f |
| 103 | 67 | 1100111 | 147 | g |
| 104 | 68 | 1101000 | 150 | h |
| 105 | 69 | 1101001 | 151 | i |
| 106 | 6A | 1101010 | 152 | j |
| 107 | 6B | 1101011 | 153 | k |
| 108 | 6C | 1101100 | 154 | l |
| 109 | 6D | 1101101 | 155 | m |
| 110 | 6E | 1101110 | 156 | n |
| 111 | 6F | 1101111 | 157 | o |
| 112 | 70 | 1110000 | 160 | p |
| 113 | 71 | 1110001 | 161 | q |
| 114 | 72 | 1110010 | 162 | r |
| 115 | 73 | 1110011 | 163 | s |
| 116 | 74 | 1110100 | 164 | t |
| 117 | 75 | 1110101 | 165 | u |
| 118 | 76 | 1110110 | 166 | v |
| 119 | 77 | 1110111 | 167 | w |
| 120 | 78 | 1111000 | 170 | x |
| 121 | 79 | 1111001 | 171 | y |
| 122 | 7A | 1111010 | 172 | z |
| 123 | 7B | 1111011 | 173 | { |
| 124 | 7C | 1111100 | 174 | \| |
| 125 | 7D | 1111101 | 175 | } |
| 126 | 7E | 1111110 | 176 | ~ |
| 127 | 7F | 1111111 | 177 | [DEL] |

# Unicode Standard

- **UTF-8** is used in HTML, and has the same byte values as ASCII.
- **UTF-16** is used in environments that balance efficient access to characters with economical applications
- Recent versions of Microsoft Windows, for example, use UTF-16 encoding.
- Each character is encoded in 16 bits.

# The x86 Architecture

Figure 1 shows the basic design of a microcomputer. It has the following main components

- **CPU** (Central Processing Unit) executes code
- The memory storage (e.g. RAM) stores all data and code
- Address, control, and Data buses
- **I/O** system interfaces with hard disk, keyboard, monitor, etc.

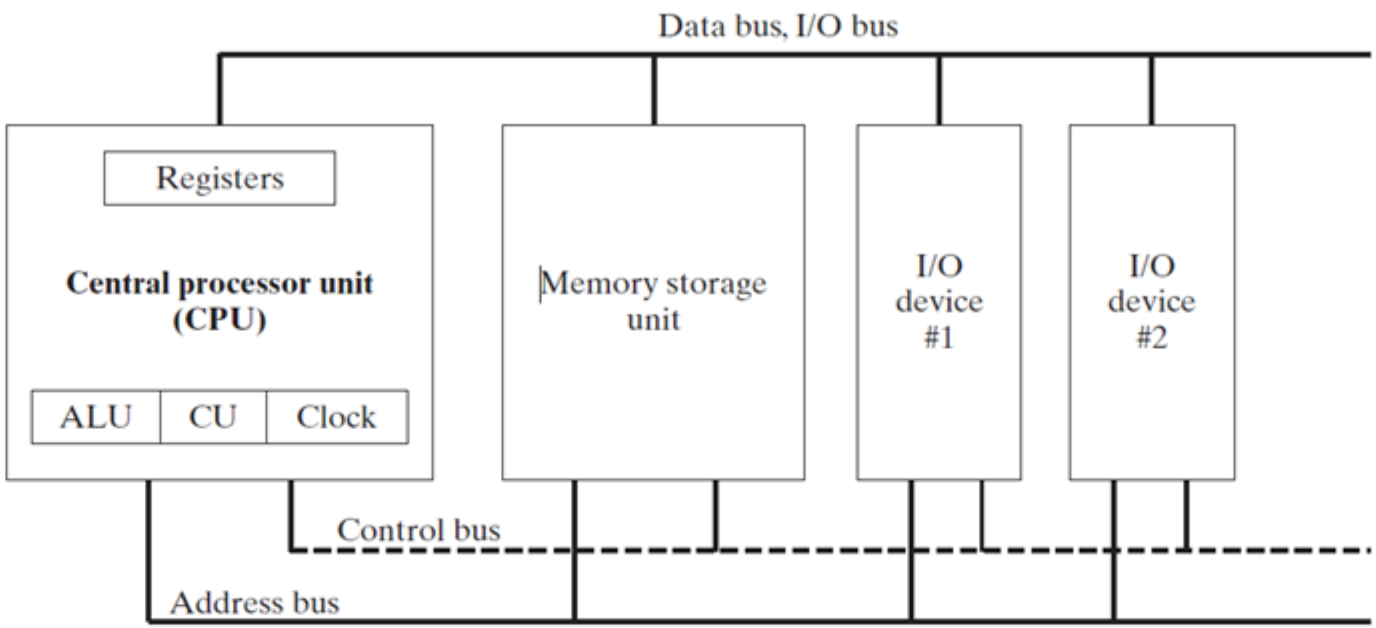
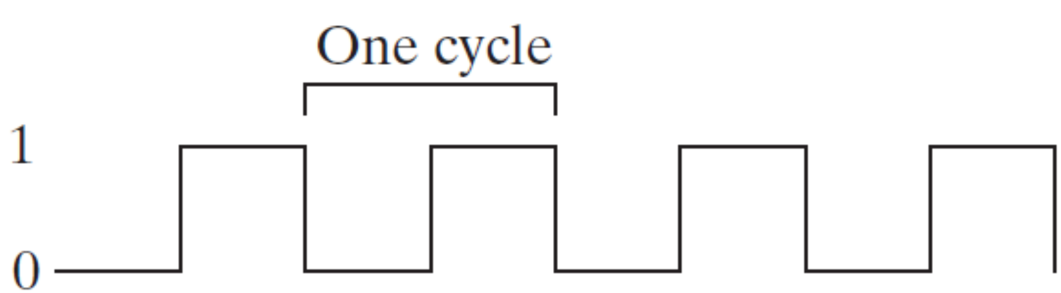



Figure 2 Block diagram of a microcomputer

# CPU Components

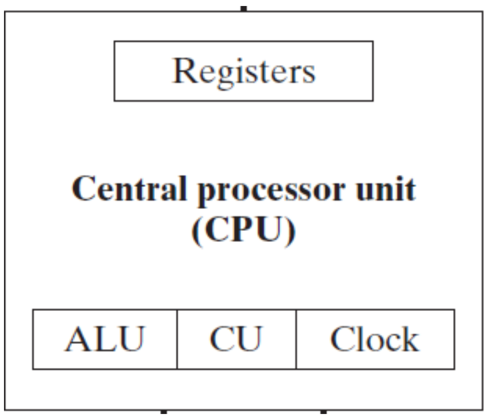
## Clock

- The *clock* synchronizes the internal operations of the CPU with other system components.
- Computer CPUs are often described in terms of their clock speeds.
- A speed of *1 GHz*, for example, means the clock ticks, or oscillates, 1 billion times per second, produces a clock cycle with a duration of one billionth of a second (1 nanosecond ($10^{-9}$ sec.)
- A machine instruction requires at least one clock cycle to execute, and a few require in excess of 50 clocks.
- Instructions requiring memory access often have empty clock cycles called *wait states* because of the differences in the speeds of the CPU, the system bus, and memory circuits.



# CPU Components

- **ALU (Arithmetic Logic Unit)**
  - Performs all arithmetic computations including addition, subtraction, multiplication, division, and all logical operations such as AND, OR, and NOT
  - Executes an instruction (fetched from RAM using a **register)** and places results in registers or RAM
- **Control Unit** (CU)
  - Coordinates the sequencing of steps involved in executing machine instructions
- **Registers**
  - Temporary data storage within the CPU
  - Faster than RAM
  - Controlled by control unit to accept, hold and transfer instructions or data and perform arithmetic or logical comparisons at a high rate of speed



# Bus

- A *bus* is a group of parallel wires that transfer data from one part of the computer to another.
- A computer system usually contains four bus types: data, I/O, control, and address.

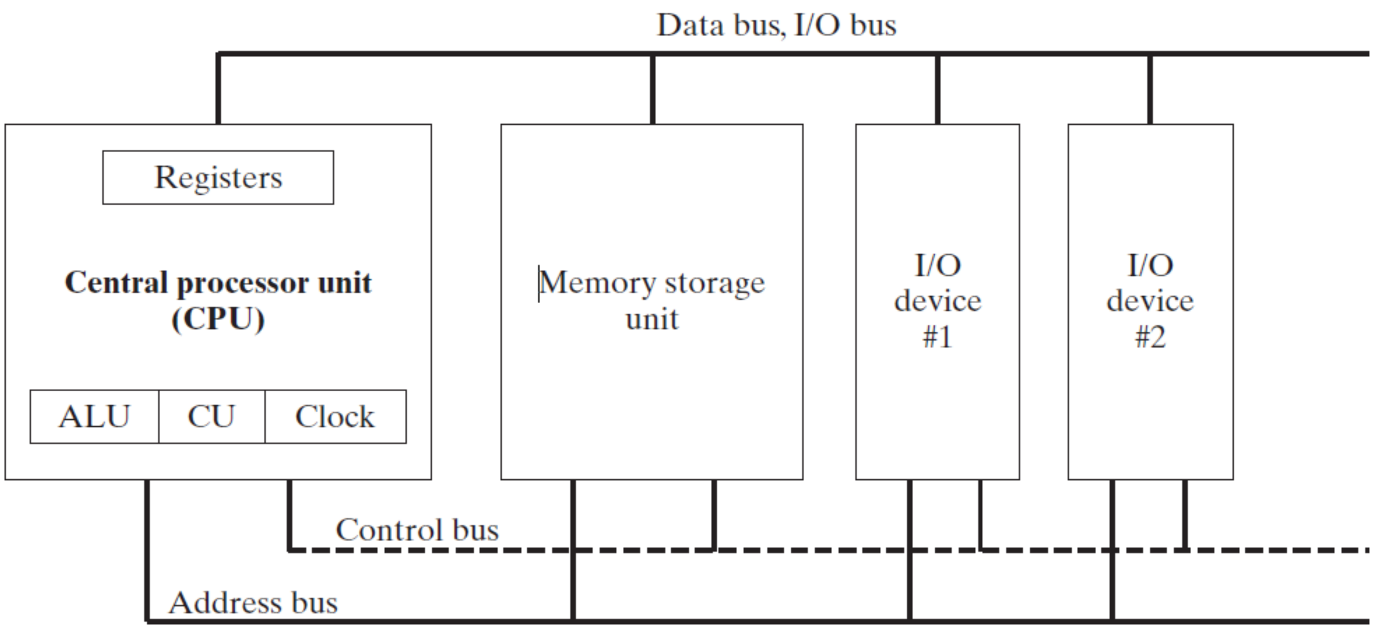



FIGURE 2–1 Block diagram of a microcomputer

# Data & CONTROL &Address Bus

- The *data bus* transfers instructions and data between the CPU and memory.
- The *address bus* holds the addresses of instructions and data when the currently executing instruction transfers data between the CPU and memory.
- The *control bus* uses binary signals to synchronize actions of all devices attached to the system bus.

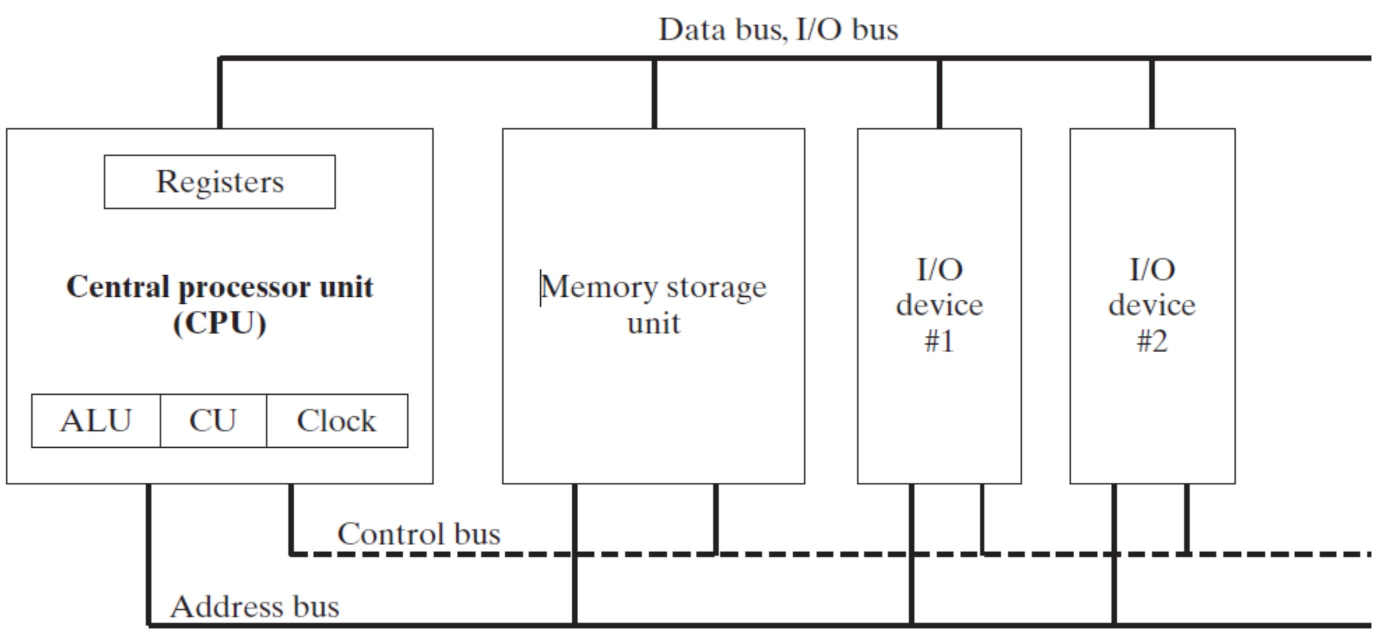



FIGURE 2–1 Block diagram of a microcomputer

# Registers

- A register is a small amount of data storage available to the CPU, whose contents can be accessed more quickly than the RAM
- Most common x86 registers, which fall into four categories:

1- General registers

- Used by the CPU during execution
- Examples: EAX, EBX, ECX, EDX, EBP, ESP, ESI

2- Segment registers

- Used to track sections of memory
- Examples: CS, SS, DS, ES, FS, GS

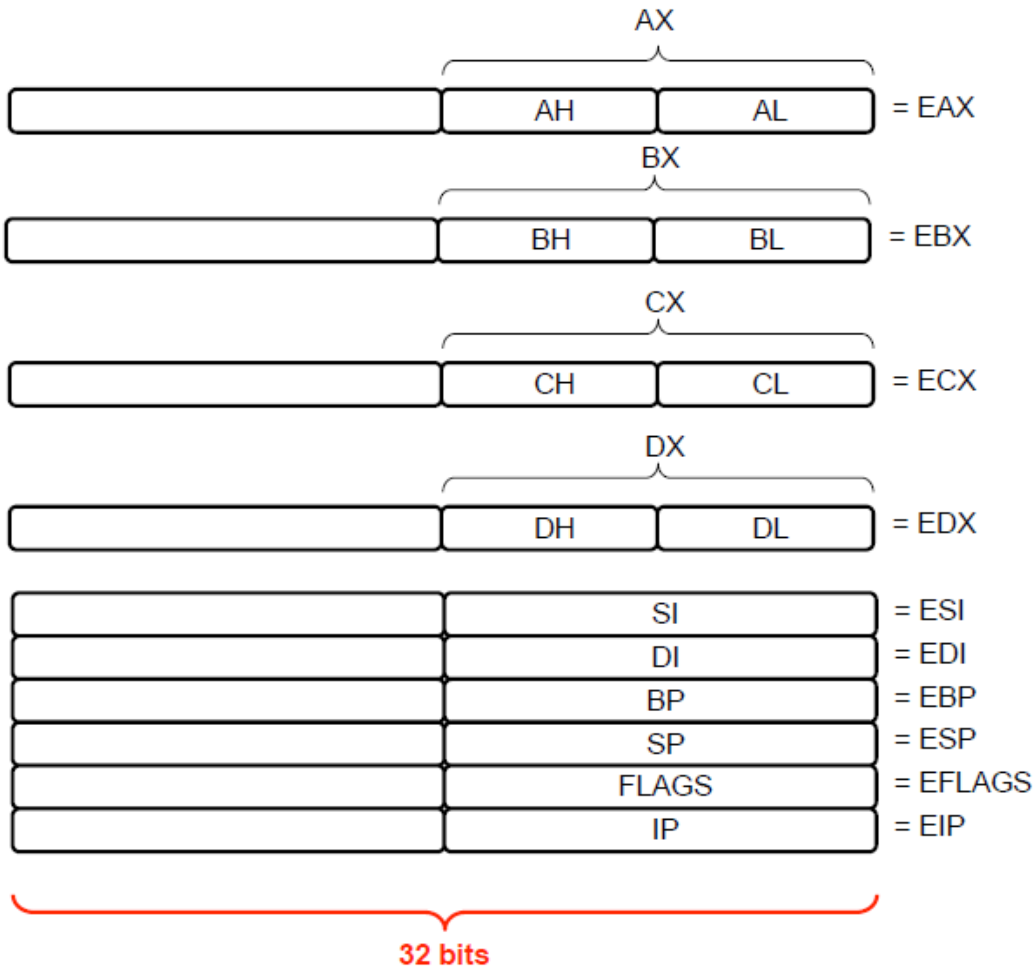
3- Status flags

- Used to make decisions
- Example: EFLAGS

4- Instruction pointer

- Address of next instruction to execute
- Example: EIP

# 80386 Registers

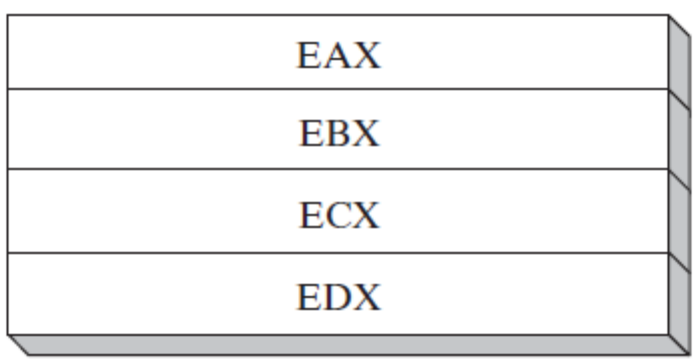


# General and Specialized Registers Tasks

- General-Purpose Registers
  - EAX – accumulator (e.g. store or load a variable’s value)
  - EAX is automatically used by multiplication and division instructions.
  - ECX – loop counter
  - ESP – Extend stack pointer: Points to the top of the function’s stack (memory structure)
  - EBP -Extended frame pointer (stack)
  - ESI- Extended Source Index Register
  - EDI- Extended Destination Index Register
  - EBP – extended frame pointer (stack)
- EIP – instruction pointer
- EFLAGS
  - status and control flags
  - each flag is a single binary bit
- Segment Registers
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments

# General Registers

- EAX  Extend Accumulator Register (32 bits)
  (EAX, AX, AH, AL)
- EBX  Extended Base Register (332 bits)
  (EBX, BX, BH, BL)
- ECX: Extend Counter Register (32 bits)
  (ECX, CX, CH,CL)
- EDX: Extend Data Register (32 bits)
  (EDX, DX, DH, DL)
- Can be used for several tasks



# Accessing Parts of Registers

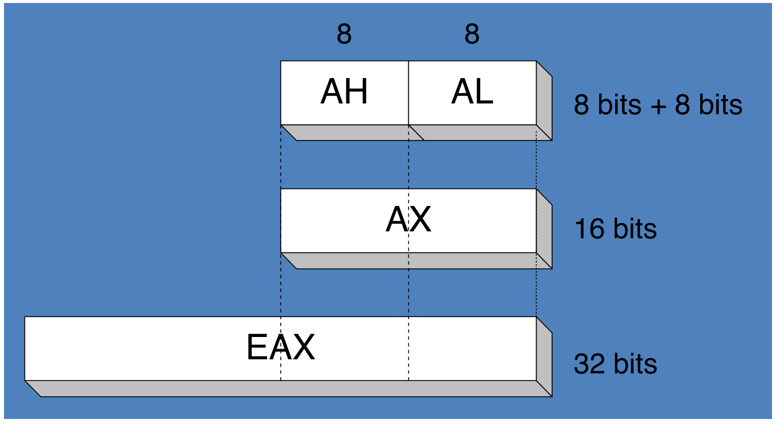
EAX Extend Accumulator Register (32 bits) (EAX, AX, AH, AL)

- Apply same concept to EAX, EBX, ECX, and EDX
- EAX, EBX, ECX, and EDX provide four 32-bits data registers, they can be used as:

  Four 32-bits registers (EAX, EBX, ECX, EDX)

  Four 16-bits registers (AX, BX, CX, DX)

  Eight 8-bits registers (AL, AH, BL, BH, CL, CH, DL, DH)

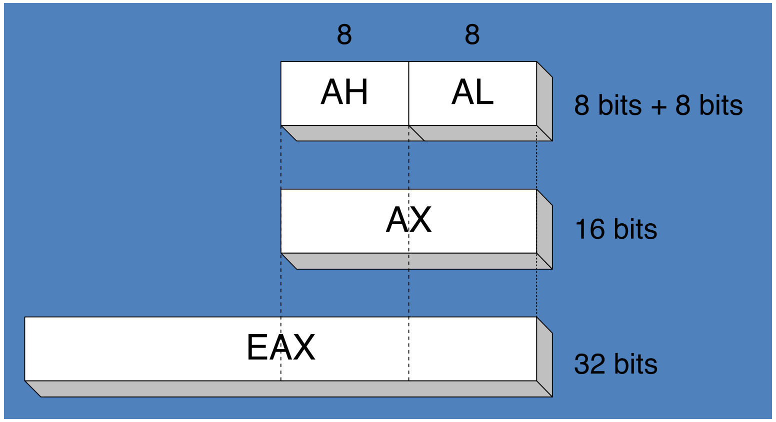



| 32-bit | 16-bit | 8-bit (high) | 8-bit (low) |
|---|---|---|---|
| EAX | AX | AH | AL |
| EBX | BX | BH | BL |
| ECX | CX | CH | CL |
| EDX | DX | DH | DL |

# Accessing Parts of Registers

MOV EAX, 0x12345678 ;
EAX = 0x12345678
AX = 0x5678,
AH = 0x56,
AL = 0x78

MOV AL,0x01 ;
AL = 0x01
AH = 0x56,
AX = 0x5601,
EAX = 0x12345601,



# Putting It All Together

**You should know:**

- **Integer Number Representation**
  - Unsigned Integer presentation
  - Signed Integer presentation
  - One’s complement
  - Two’s complement
  - Binary Addition
  - Binary Subtraction
- **Binary Addition**
- **Binary Subtraction**
- **Boolean Operations**
  - Not operation
  - OR operation
  - AND operation
- **ASCII Presentation, Unicode Standard**
- **The x86 Architecture**
  - CPU Components (Clock, ALU, Registers)
  - Bus (Address Bus, Control Bus, data bus_
  - I/O Devices
  - General Purpose Registers

# Questions?

# Week 2 Assignments

- **Learning Materials**
  1- Read Week 2 Presentation
  2- Ch.1, 2: Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux,
  3- Reading Ch1: PCASM textbook

- **Assignment**
  1- Complete “Lab 2” by coming Sunday 11:59 PM.
  Fill up the “Lab 2 Table Answer” before submitting your answer
  2- Complete “Lab 3” by coming Sunday 11:59 PM.
  Fill up the “Lab 3 Table Answer” before submitting your answer