Assembly Language

Compiled By: Abdul Haseeb

Recap

- Generation to Generation Technologies and Input Methods
- ENIAC, UNIVAC,IBM 360 Family and some other Models
- Common Characteristics of a computer Family
- Jhon Von Neuman and IAS Computer

What is a Number System?

- A writing system for expressing Numbers
- There are variety of Number systems like:
  - Binary
  - Hexadecimal
  - Decimal
  - Octal

Number System

- Base of Binary?
- Base of Decimal?
- Base of HEX?

Number System

- Convert Binary to Decimal and vice versa
- Convert Binary to HEX and vice versa
- Convert Decimal to HEX and vice versa

Binary to Decimal

Example 2

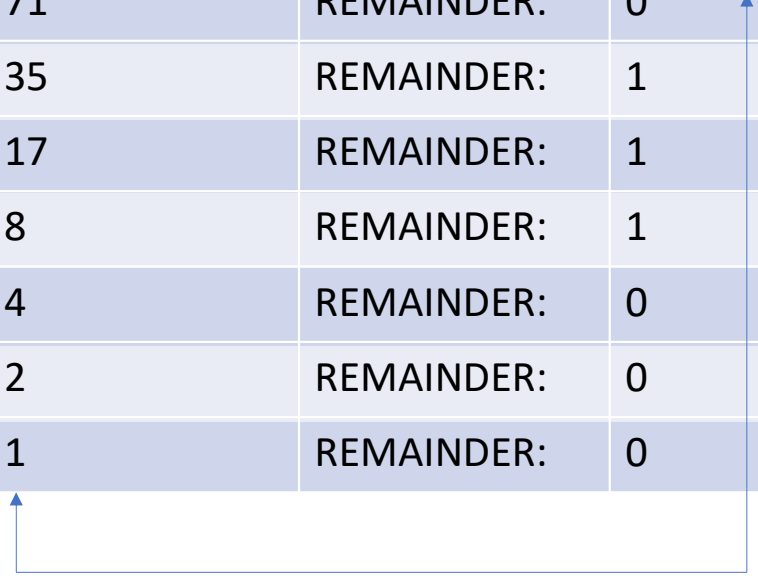
Convert the following binary number, 011110001011, into a denary number.

2048	1024	512	256	128	64	32	16	8	4	2	1
0	1	1	1	1	0	0	0	1	0	1	1

The equivalent denary number is 1024 + 512 + 256 + 128 + 8 + 2 + 1 = 1931

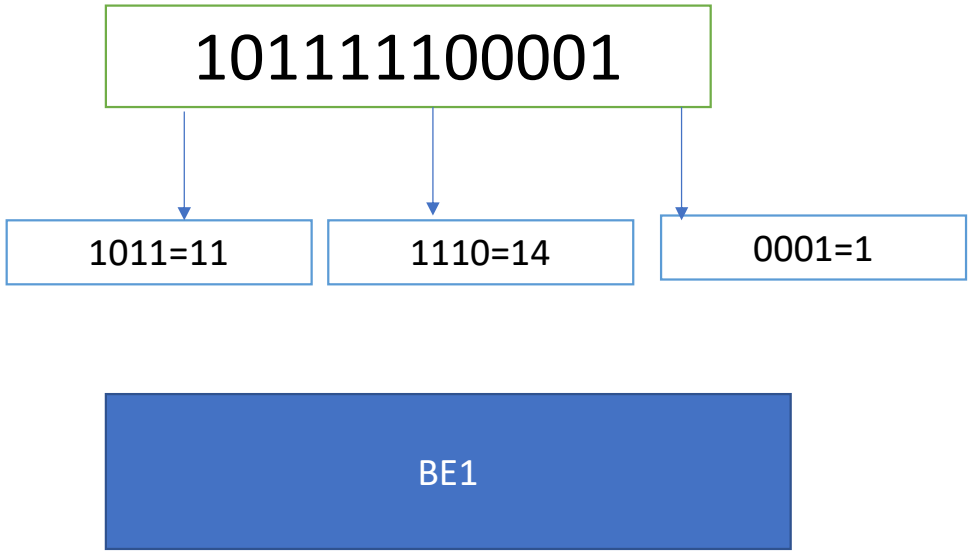
Decimal to Binary

2	142		
2	71	REMAINDER:	0
2	35	REMAINDER:	1
2	17	REMAINDER:	1
2	8	REMAINDER:	1
2	4	REMAINDER:	0
2	2	REMAINDER:	0
2	1	REMAINDER:	0



Result:
10001110

CONVERTING FROM BINARY TO HEXADECIMAL



HEXADECIMAL TO BINARY

Example 4

B	F	0	8

Again just use Table 1.1:

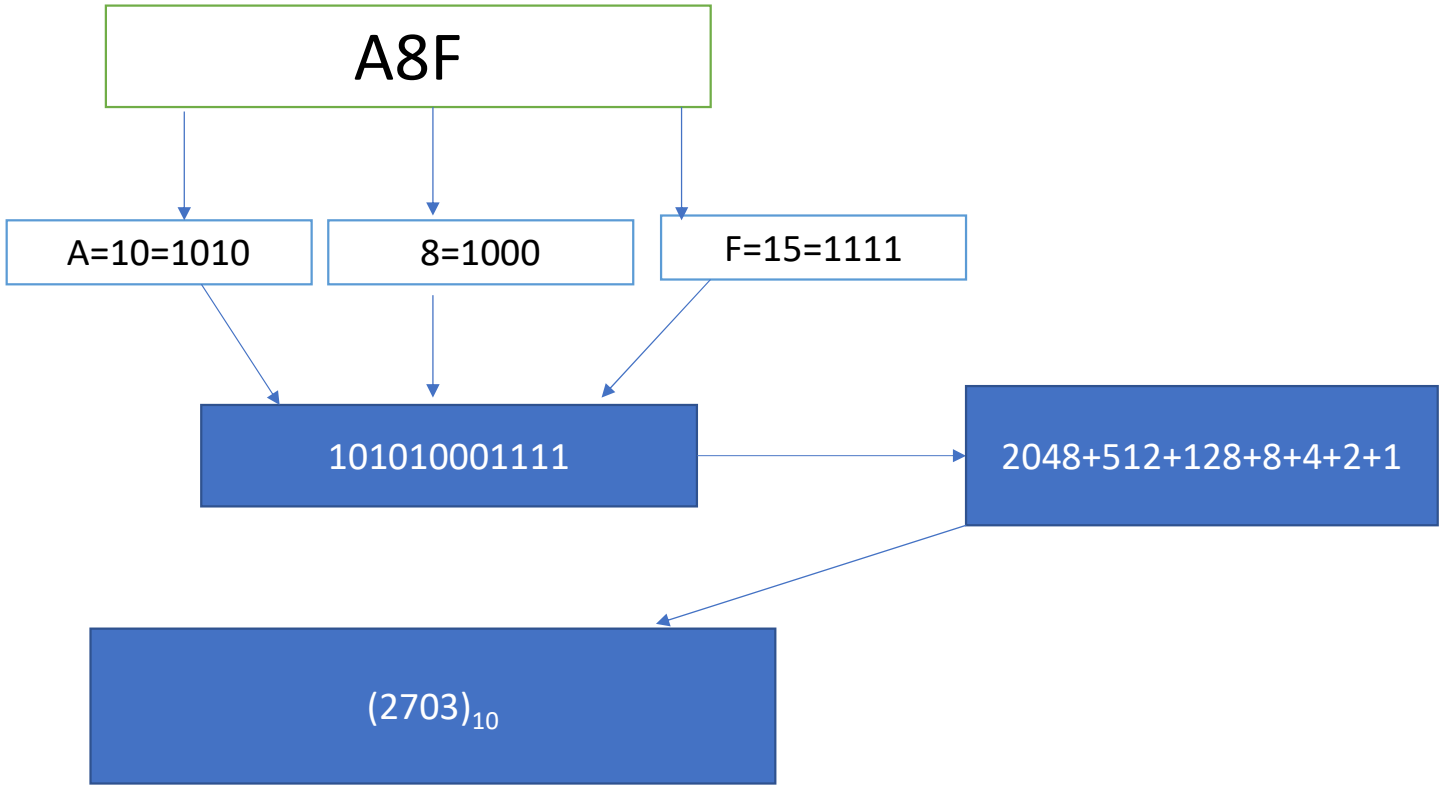
1 0 1 1	1 1 1 1	0 0 0 0	1 0 0 0

Then put all the digits together:

1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0

CONVERTING FROM HEXADECIMAL TO DENARY

HEXA TO BINARY



CONVERTING FROM DENARY TO HEXA

16	8463	
16	528	→ 15
16	33	→ 0
16	2	→ 1

ANSWER:
210F

Why Computers use Binary Number System?

- Millions of Tiny switches in computer, that are either in ON state or OFF State
- When its ON, it is represented by 1
- When its OFF, it is represented by 0

Representation of Numbers in Computer

- One’s Complement
- Two’s Complement

8086/8088

- Both are 16 bit microprocessors
- IBM choose 8086 over 8088:
  - Better performance
  - Less expensive

What is Assembly Language

- **Assembly Language** is a Low Level Programing Language Design for a Specific type Processor .
- It is one step higher than machine language.
- In Assembly Language symbols are used instead of binary code.
- These symbols are called **mnemonics**
- For Example ADD instruction is used to Addition two numbers

What is Assembly Language

- In computer there is assembler that helps in converting the assembly code into machine code Executable
- Assembly language is designed to understand the instruction and provide to machine language for further processing
- It depends on the architecture of the system whether it is the operating system or computer architecture

Why We Study Assembly Language

- Better under Standing about the interaction of Hardware and software
- Optimize the processing Time
- Assembly language helps in providing full control of what tasks a computer is performing.
- By learning assembly language, the programmer is **able to write** the code to access registers and able to retrieve the memory address of pointers and values
- Assembly language learning helps in understanding the **processor** and **memory** functions.
- Assembly language helps in understanding the work of processor and memory.

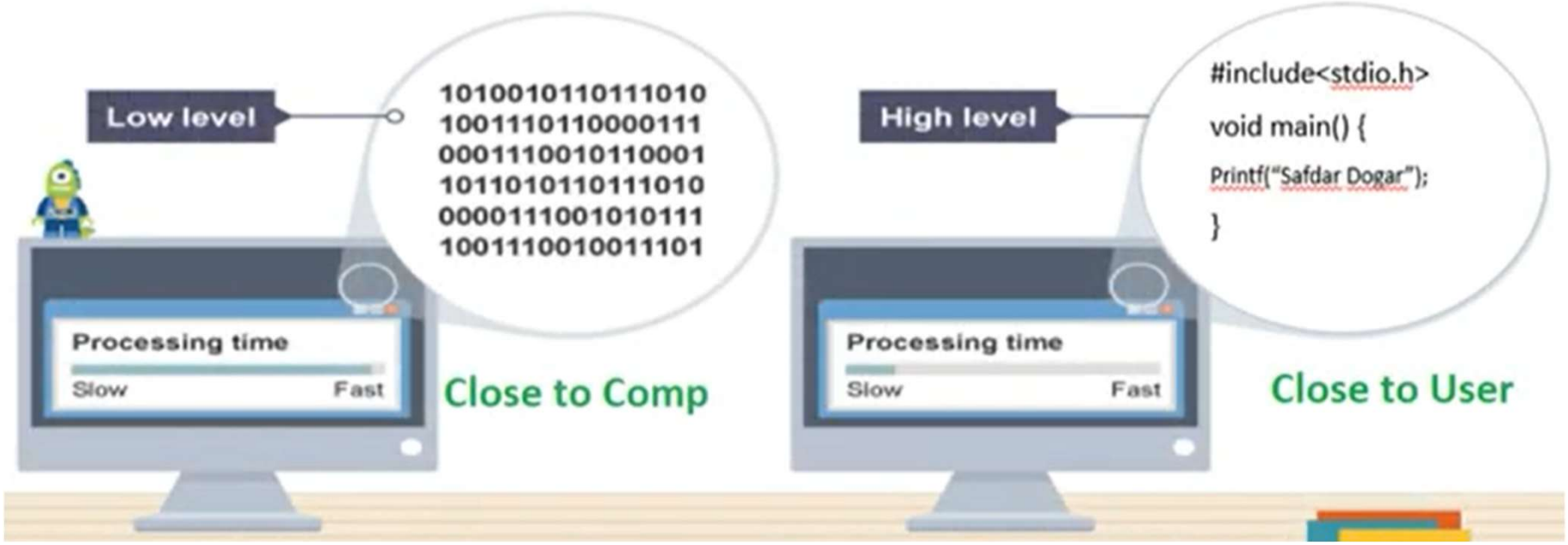
Programming Language

- A Language to write Computer Programs
- Two Types:
  - Low Level:
    - Machine and Assembly
    - Faster
  - High Level:
    - C, Java, Python, R etc

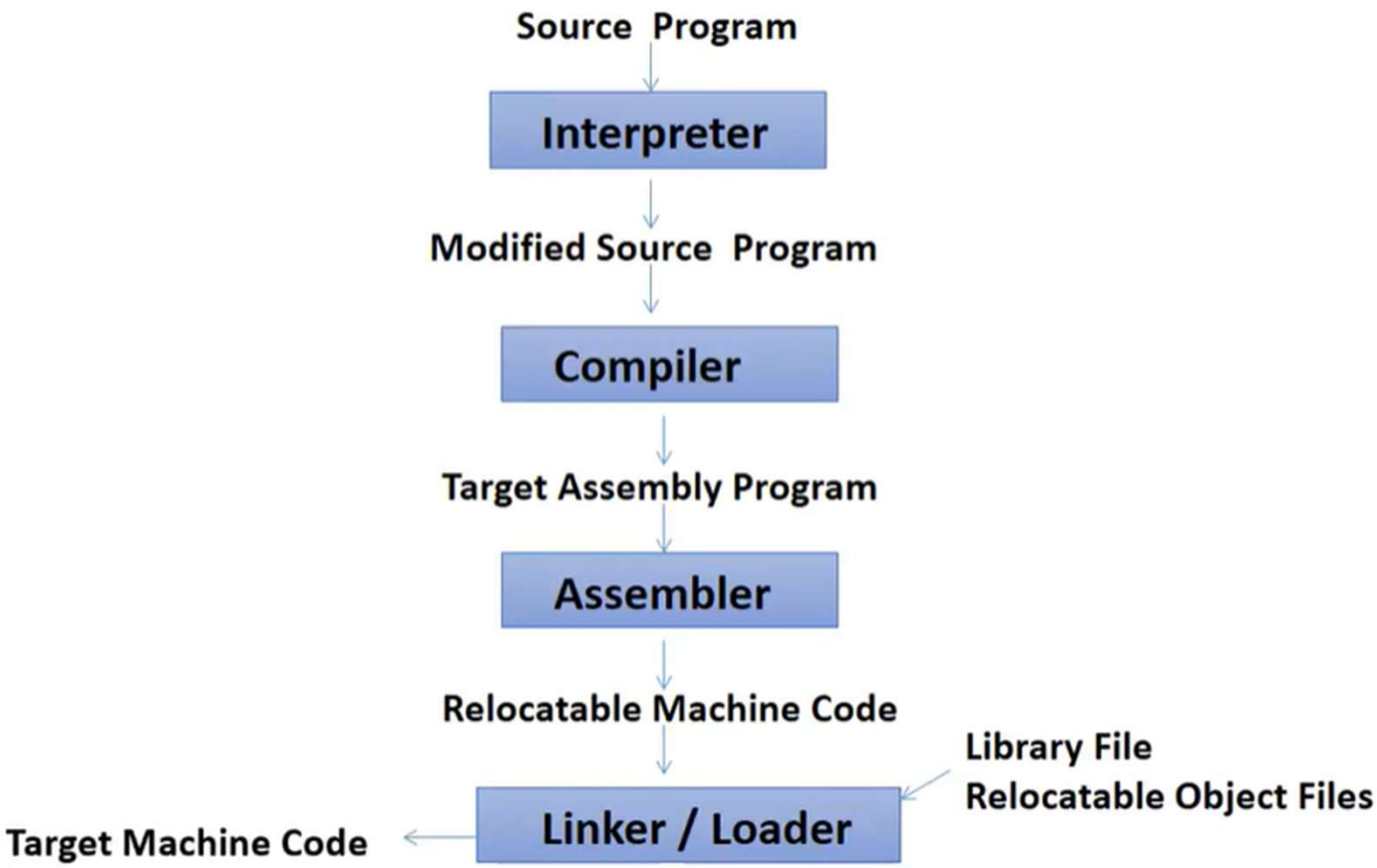
Machine Language

- **A language in which instruction are in binary code is called machine language.**
- **Computer understand only binary form**
- **Machine Language program is directly understood by the computer**

Diff B/T Low Level High Level Language



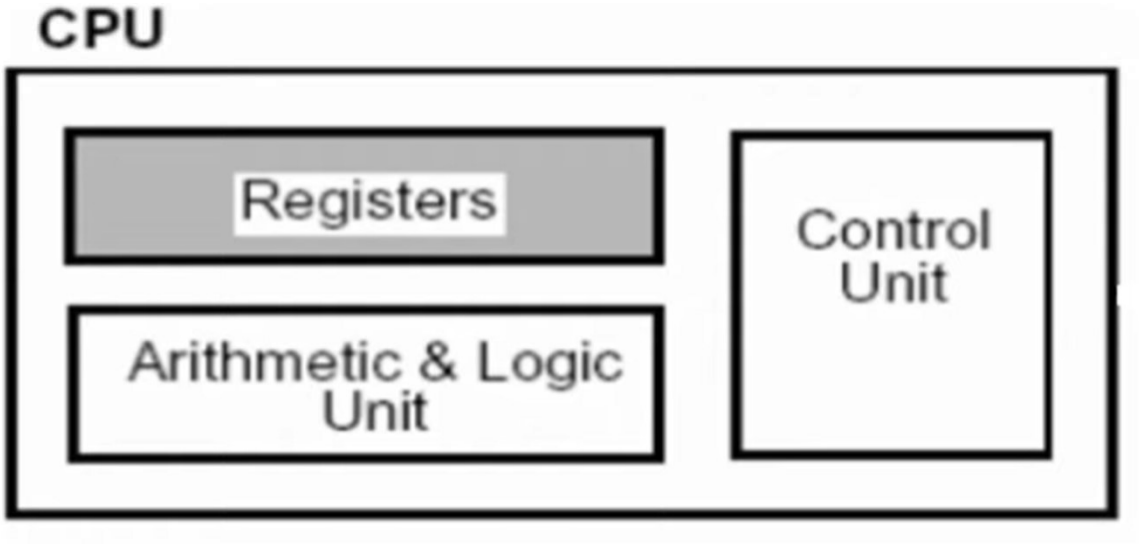
Language Processing System



What are CPU Registers

- The small storage areas inside the CPU are called Registers.
- These are high speed memories
- These are used to store data and instruction during program execution.
- CPU consists of many types of registers
- Each register is used for a specific purpose
- The size of register is 2 , 4 or 8 bytes or more
- The large size of register increase the performance of CPU

Registers Location



Type of Registers

- **Special Purpose Registers**
- **General Purpose Registers**
- **Address or Segment Registers**

Special Purpose Registers

- **Program Counter (PC)**
- **Instruction Register (IR)**
- **Memory Address Register (MAR)**
- **Memory Buffer Register (MBR)**
- **Stack Pointer Register (SPR)**

General Purpose Registers

- **Accumulator Register (AX)**
- **Base Register (BX)**
- **Counter Register (CX)**
- **Data Register (DX)**

Address Purpose Registers

- **Code Segment (CS)**
- **Data Segment (DS)**
- **Extra Segment (ES)**
- **Stack Segment (SS)**

Program Counter (PC)

- Program counter is used to store the address of the next instruction to be fetched for execution
- When the instruction is fetched the value of program counter is incremented
- It now refers to the next instruction

Instruction Register (IR)

- **Instruction register is used to store the fetched instruction**
- **The instruction is also decode in this register**

Memory Address Register (MAR)

- **Memory address register is used to store memory address being used by CPU**
- **When CPU wants to read or write data in memory it store the address of that memory location in this register**

Memory Buffer Register (MBR)

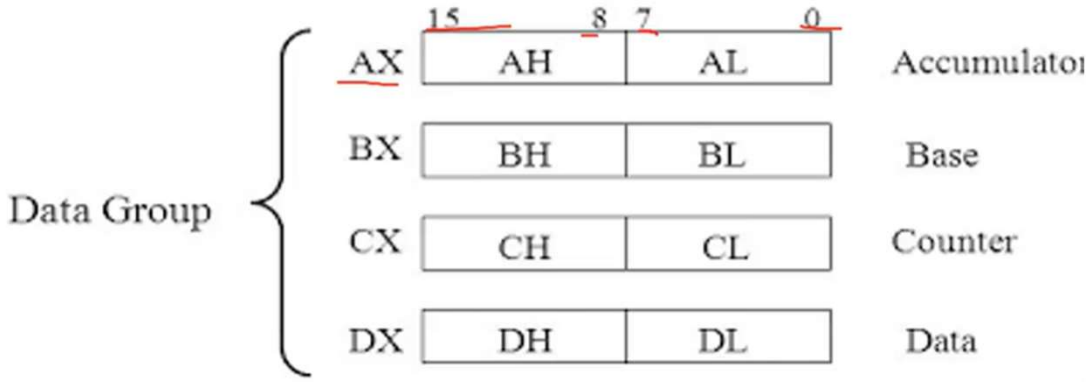
- **Memory buffer register is used to store the coming from the memory or going to the memory**

Stack Pointer Register (SPR)

- **A stack is set of memory locations in which data is stored and retrieved in an order**
- **It points to the topmost element of stack.**
- **16 bit registers**
- **This order is called FIRST IN LAST OUT or LAST IN FIRST OUT**

Accumulator Register (AX)

- **It is Accumulator .**
- **Used for arithmetical and logical instructions.**
- **16 bits and is divided into two 8-bit registers AH and AL to perform 8-bit instructions.**



Base Register (BX)

- It is used for arithmetic and data movement
- It has special addressing capabilities
- It is 16 bits and is divided into two 8-bit registers BH and BL to perform 8-bit instructions.
- used to store the value of the offset.

Counter Register (CX)

- It is used for counting purpose.
- It acts as a counter for repetitions or loops
- 16 bits and is divided into two 8-bit registers CH and CL to perform 8-bit instructions.
- used in looping and rotation.

Data Register (DX)

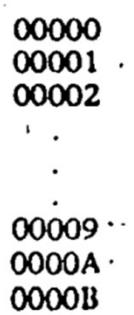
- It is used for division and multiplications
- 16 bits register and is divided into two 8-bit registers DH and DL to perform 8-bit instructions.
- used in multiplication an input/output port addressing.

Segment Registers

- Memory:
  - Collection of bytes
  - Each byte has address which starts from 0
  - 8086 assigns:
    - 20 bit physical address to each location
    - Possible to address one MB of memory

Segment Registers

- Writing addresses in binary is very cumbersome so we express them as five hex digits



- Highest address is FFFFFh

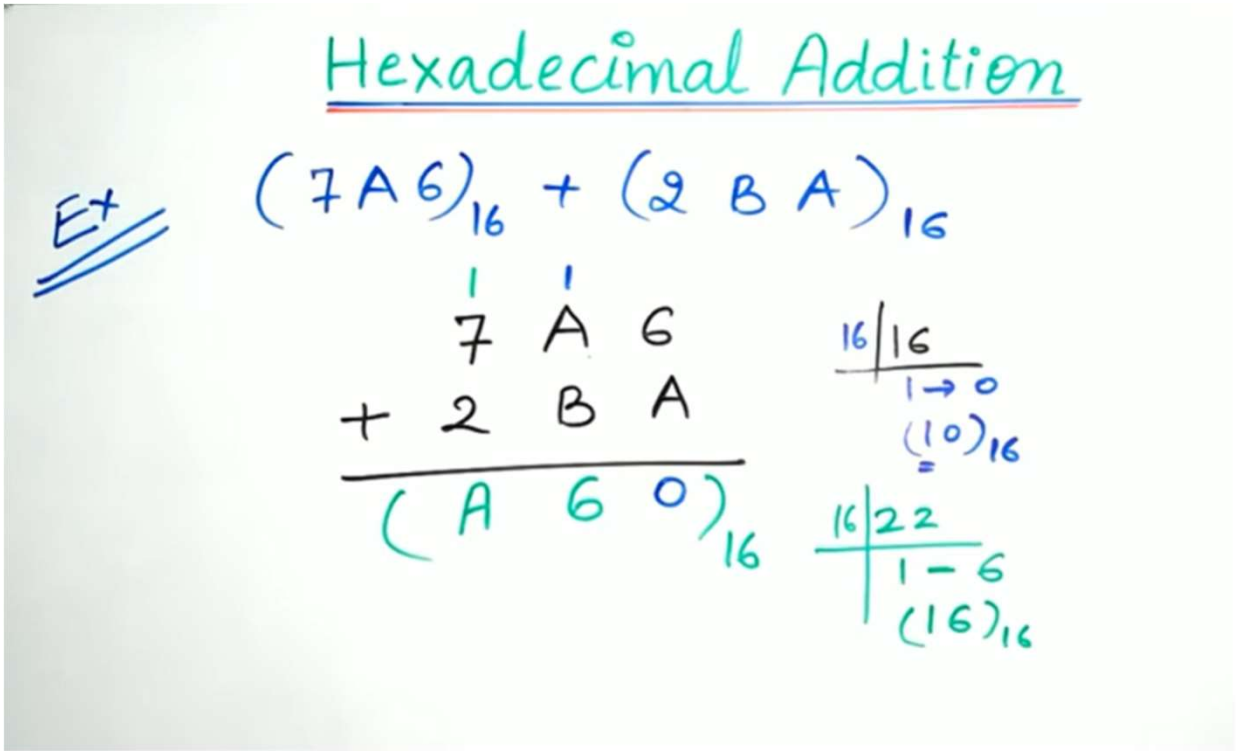
Segment Registers

- Processor uses 16 bit address
- While physical address in 20 bits
- To cover this gap we introduce the concept of memory segments

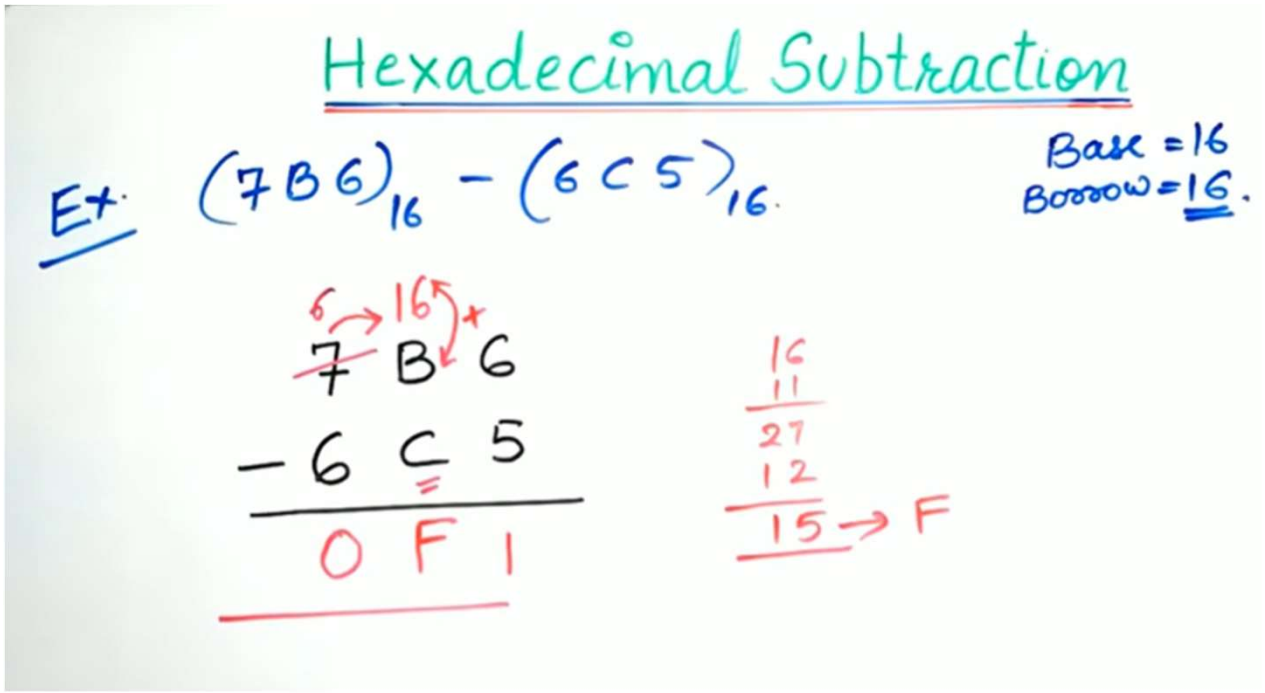
Segment Registers

- Memory Segment:
  - Block of 2^{16} consecutive Memory Location
  - Identified by a segment Number
  - Within a segment a memory location is specified by giving an offset number

Hexa Addition



Hexa Subtraction



Segment:offset Address

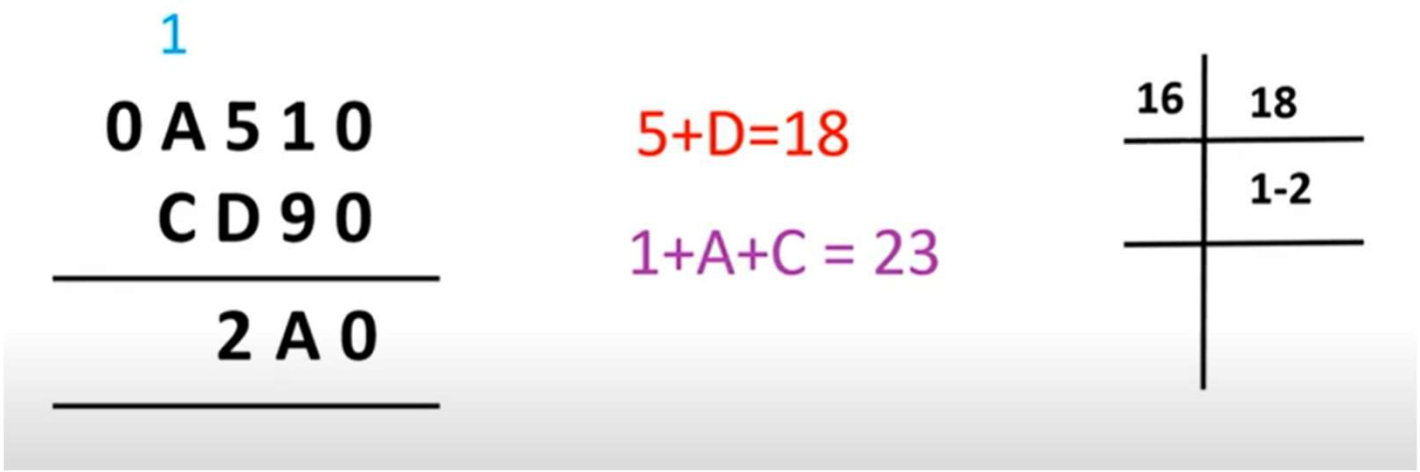
- Also called Logical Address
- We obtain physical Address from it:

$$\begin{aligned}\text{Physical Address} &= \text{Seg} \times 10h + \text{Offset} \\ &= 0A51 \times 10h + CD90h \\ &= 0A510h + CD90h \\ &= 172A0h\end{aligned}$$

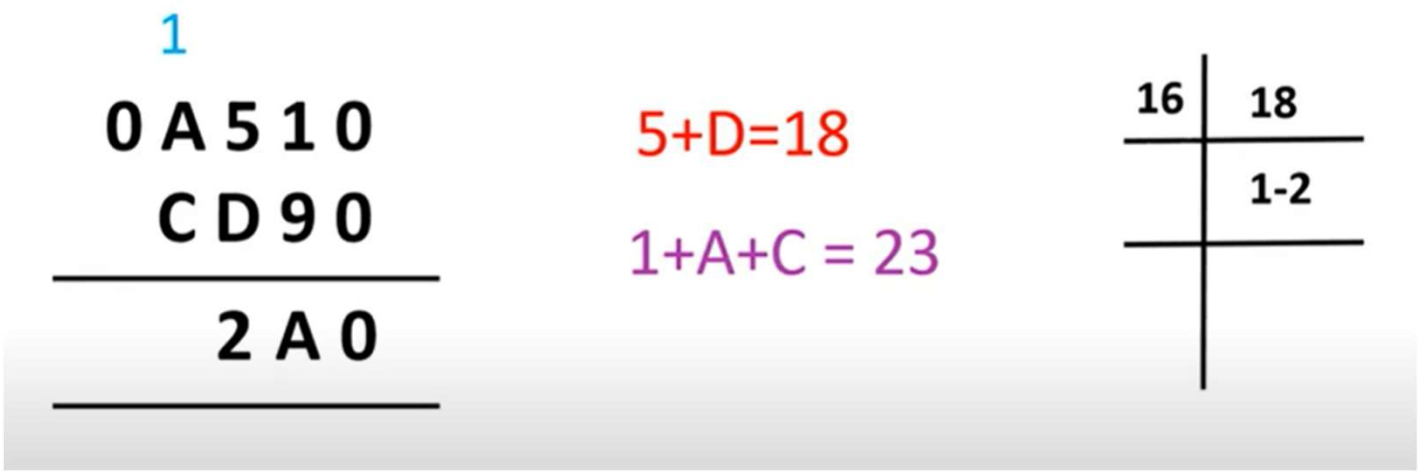
Determine the physical Location of 0A51:CD90h

```
  0A510
  CD90
-------
    A0
-------
```

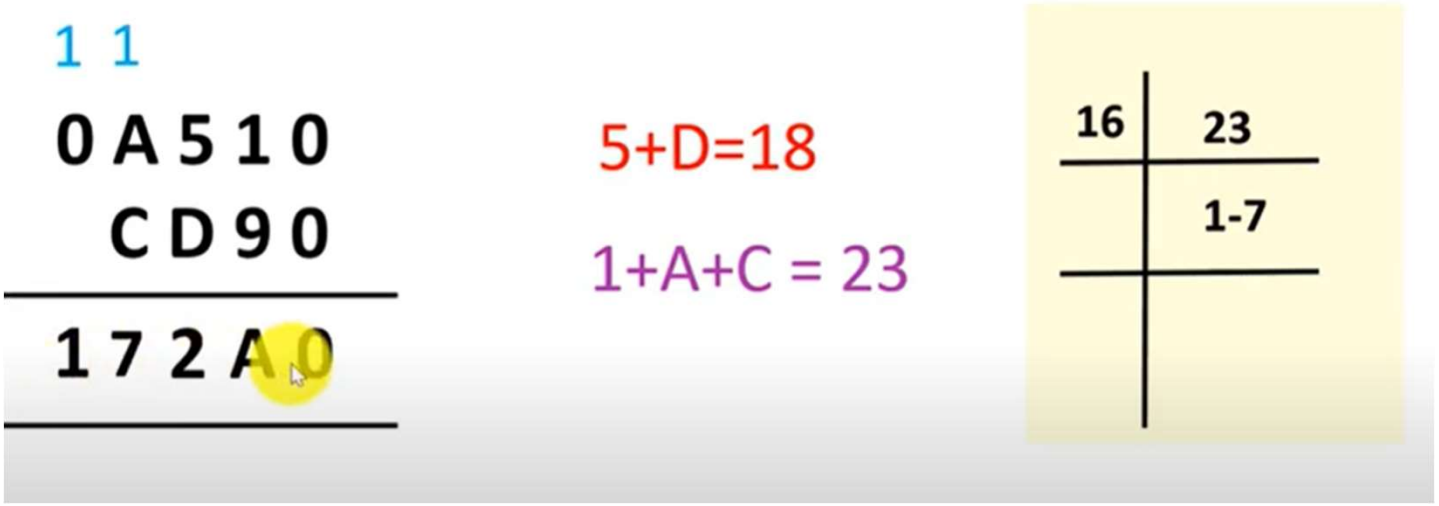
Determine the physical Location of 0A51:CD90h



Determine the physical Location of 0A51:CD90h



Determine the physical Location of 0A51:CD90h



A memory location has physical address of 4A37Bh. Compute the offset Address if the segment number is 40FFh

A memory location has physical address of 4A37Bh. Compute the offset Address if the segment number is 40FFh

- Answer:
  - 938Bh

A memory location has a physical Address of 4A37Bh, Compute Segment Number if the offset Address is 123B

Physical Address = Seg × 10h + Offset

Seg × 10h = Physical Address - offset

Seg × 10h = 4A37B – 123B

Seg × 10h = 49140

Seg = 49140 / 10h

= 4914h

```
  4 A 3 7 B
    1 2 3 B
-----------
  4 9 1 4 0
```

Segment Registers

- **Code Segment:**
  - Contains the address of the code
- **Data Segment:**
  - Contains the address of data
- **Stack Segment:**
  - Contains the stack segment number
- **Extra Segment:**
  - If program needs to contain more data, it can use DS

Pointer Registers

- They point to the offset address
  - **Stack Pointer:**
    - Used in conjunction with SS, to access stack segment
    - Points to TOS
  - **Base Pointer**:
    - Used to access the data on the stack
    - Points to Stack Base
    - Also used to access data from other segments unlike SP
  - **Source Index:**
    - Points to memory location in the data segment addressed by DS
  - **Destination Index:**
    - Same function as SI
    - String operations:
      - Use DI to access memory locations addressed by ES

Instruction pointer

- Registers covered so far access data only
- To access instructions:
  - CS and IP are used
  - CS holds segment number of the next instruction and IP contains the offset
  - IP is updated each time so it points to the next instruction

Flag registers

- Indicate the status of microprocessor
- Status FLAG
- Control FLAG
- To be covered in next classes

Some Important points

- OS and its functions
- BIOS
- I/O Ports (64KB)
- Startup of computer

Is assembly Language code case sensitive

- No Assembly code is not case sensitive
- But its better to use UPPER CASE Letters
- Assembler which we will be using is MASM(Microsoft Macro Assembler)

Assembly Program

- Has statements
  - One per line
  - Either Instruction
  - OR Assembler Directive
  - There should be a gap between the fields of a statement

General Instruction Format

instruction destination , source

Formal structure of Assembly Statement

name operation operand **;comment**

Name Field

- Used for:
  - Procedure Names and Variable Names
  - 1 to 31 characters
  - Special characters: @,_,$,?,%
  - No difference in upper and lowercase
  - No space allowed
  - Can not begin with a number

Operation field

- Contains opcode:
  - Operation to be performed

Operand field

- Specifies the Data:
  - Zero, one or two operands
    - NOP
    - INC AX
    - ADD WORD1,2

Comments

;this is a comment in assembly

EMU8086

- https://www.malavida.com/en/soft/emu8086/

EMU8086

- Assembly Language Emulator
- Easy to work with interface
  - See the value of flags and registers being changed

Demo

- MOV AX,10
- MOV AX,11
- MOV AX,300

Program: Addition and Subtraction

```
Mov AX, 10
Mov BX, 6
Add AX,BX
Mov BX,34
Sub BX,AX
```

Writing the Demo Program in EMU8086

- Lets have a go!

Task

- Store 20 in Accumulator
- Store 30 in Base Register
- Add value of Accumulator and Base Register and store in Base Register
- Move the value of Base Register to Accumulator
- Move 20 in Base Register
- Subtract Base Register from Accumulator and store back in Accumulator Register

Solution

```
01
02  ; You may customize this and other start-up templates;
03  ; The location of this template is c:\emu8086\inc\0_com_template.txt
04
05  org 100h
06
07  MOV AX,20
08  MOV BX,30
09  ADD BX,AX
10  MOV AX,BX
11  MOV BX,20
12  SUB ax,bx
13
14  ret
```

Task

- Multiply 4*4

Variables

- Play same role as they do in high level
- Each variable has:
  - Data Type
  - Memory Assigned

Byte Variable

- Syntax:
  - name DB initial_value
- Exp:
  - ALPHA DB 4
- Uninitialized byte:
  - BYT DB ?
- Search for range of values…

Word Variable

- Syntax:
  - name DW initial_value
- Exp:
  - WRD DB -2
- Search for range of values…

Array Variable

- Array is sequence of memory bytes
- Defining a three byte array:
  - B_Array DB 10H,20H,30H

Symbol	Address	Contents
B_ARRAY	200h	10h
B_ARRAY+1	201h	20h
B_ARRAY+2	202h	30h

Array Variable

Defining a three words array:

- W_Array DW 1000,3334,3122

Symbol	*Address*	*Contents*
W_ARRAY	0300h	1000d
W_ARRAY+2	0302h	40d
W_ARRAY+4	0304	29887d
W_ARRAY+6	0306h	329d

Character Strings

```
LETTERS     DB      'ABC'
```

is equivalent to

```
LETTERS     DB      41H,42H,43H
```

It is possible to combine characters and numbers in one definition; for example,

```
MSG DB      'HELLO',0AH,0DH,'$'
```

is equivalent to

```
MSG DB      48H,45H,4CH,4CH,4FH,0AH,0DH,24H
```

Named Constants

- To assign a name to a constant: EQU(equates)
- LF EQU 0Ah
- PROMT EQU “Type your name:”
- Instead of MOV DL,0AH
  - MOV DL,LF
- No Memory is allocated for constant names

Demo for XCHG

- Exchanges the contents
- XCHG AX,BX

Legal Combinations for MOV Instruction

Table 4.2 Legal Combinations of Operands for MOV and XCHG

MOV

	Destination Operand			
Source Operand	*General register*	*Segment register*	*Memory location*	*Constant*
General register	yes	yes	yes	no
Segment register	yes	no	yes	no
Memory location	yes	yes	no	no
Constant	yes	no	yes	no

XCHG

	Destination Operand	
Source Operand	*General register*	*Memory location*
General register	yes	yes
Memory location	yes	no

INC, DEC

- INC:
  - Increments by one
- DEC:
  - Decrements by one

NEG

- Performs twos complement

Memory Model

- Determines:
  - Size of code and data in a program
- .MODEL directive:
  - Defines a memory model
- Syntax:
  - .Model memory_model
- Model directive comes before defining any memory segment

Memory Model

Table 4.4 Memory Models

Model	*Description*
SMALL	code in one segment data in one segment
MEDIUM	code in more than one segment data in one segment
COMPACT	code in one segment data in more than one segment
LARGE	code in more than one segment data in more than one segment no array larger than 64k bytes
HUGE	code in more than one segment data in more than one segment arrays may be larger than 64k bytes

Data Segment

- Contains all the variable definitions
- Constants also made here

```
.DATA
WORD1                DW 2
WORD2                DW 5
MSG                  DB 'THIS IS A MESSAGE'
MASK                 EQU 10010010B
```

Stack Segment

- Set aside a memory block to be used as stack
- 100 bytes are enough for most of applications, if size is omitted 1 KB is set aside

```
.STACK          size

.STACK          100H
```

Code Segment

- Contains program instructions
- name is a small name:
  - Will generate an error on SMALL model if we use it
- Inside code segment instructions are organized as procedures

```
.CODE  name

name   PROC
;body  of  the  procedure
name   ENDP
```

Code Segment

```
.CODE
MAIN PROC
;main procedure instructions
MAIN ENDP
;other procedures go here
```

Putting it all to gather

```
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN   PROC
;instructions go here
MAIN   ENDP
;other procedures go here
END    MAIN
```

INT Instruction

- Generates an interrupt with a particular interrupt number
  - Number specifies a particular function
    - INT 16h performs keyboard input

INT 21h

- It has different functions, identified by a function number
  - Place function number in AH

Function number	*Routine*
1	single-key input
2	single-character output
9	character string output

Function 1

Function 1:
Single-Key Input

Input: AH = 1
Output: AL = ASCII code if character key is pressed
= 0 if non-character key is pressed

To invoke the routine, execute these instructions:

```
MOV AH,1          ;input key function
INT 21h           ;ASCII code in AL
```

Function 2

Function 2:
Display a character or execute a control function

Input: AH = 2
DL = ASCII code of the display character or control character

Output: AL = ASCII code of the display character or control character

```
MOV AH,2                ;display character function
MOV DL,'?'              ;character is '?'
INT 21h                 ;display character
```

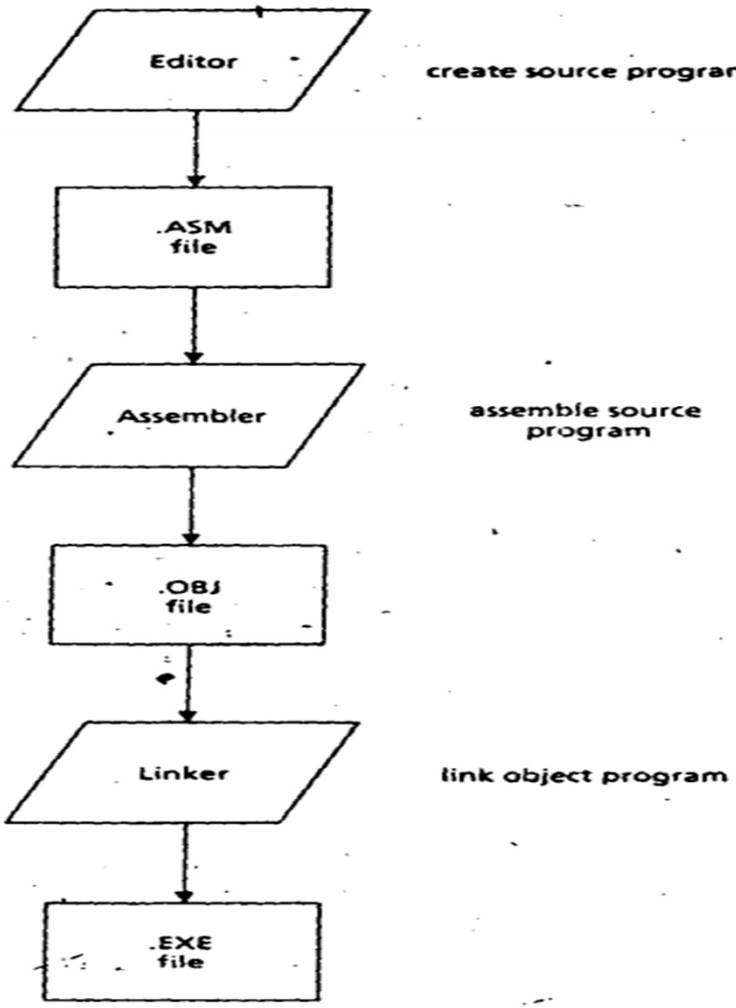
Display first letter of your name using function 2

Function 2

- Also perform control functions by moving different numbers in DL

ASCII code (Hex)	*Symbol*	*Function*
7	BEL	beep (sounds a tone)
8	BS	backspace
9	HT	tab
A	LF	line feed (new line)
D	CR	carriage return (start of current line)

On execution, AL gets the ASCII code of the control character.


Editor
create source program
.ASM
file
Assembler
assemble source
program
.OBJ
file
Linker
link object program
.EXE
file

- Study about cross reference and source listing file