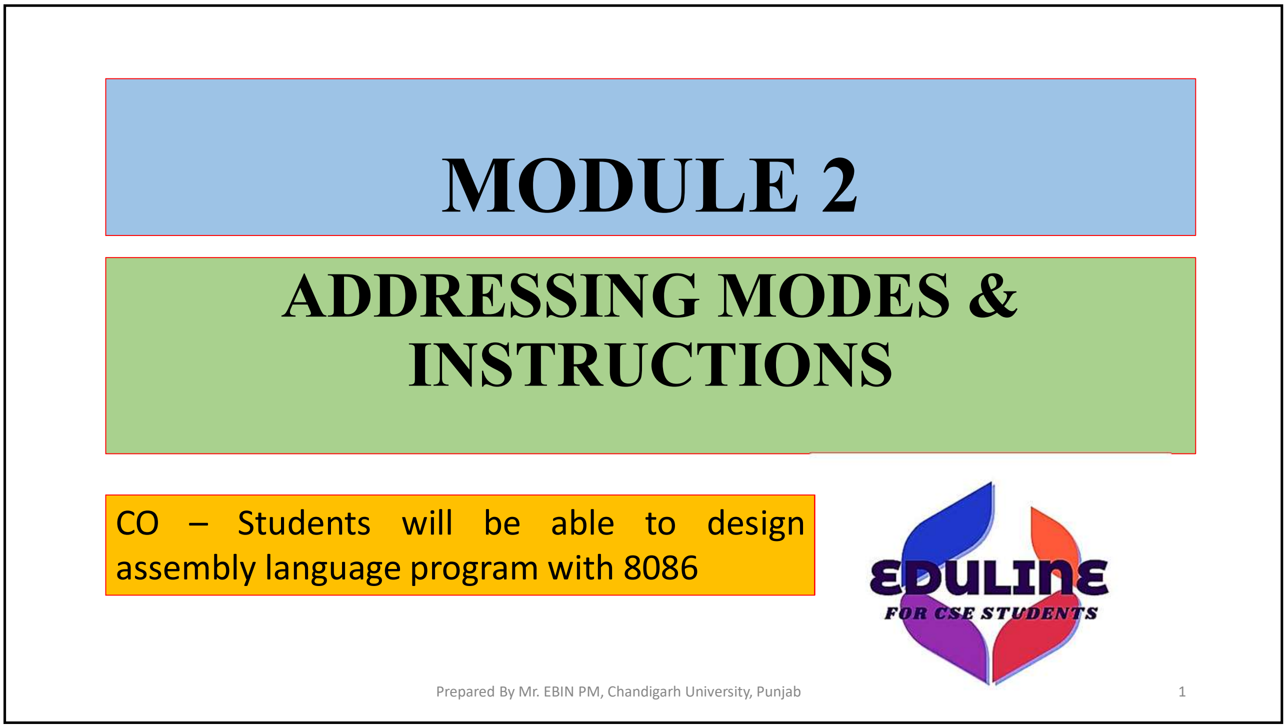
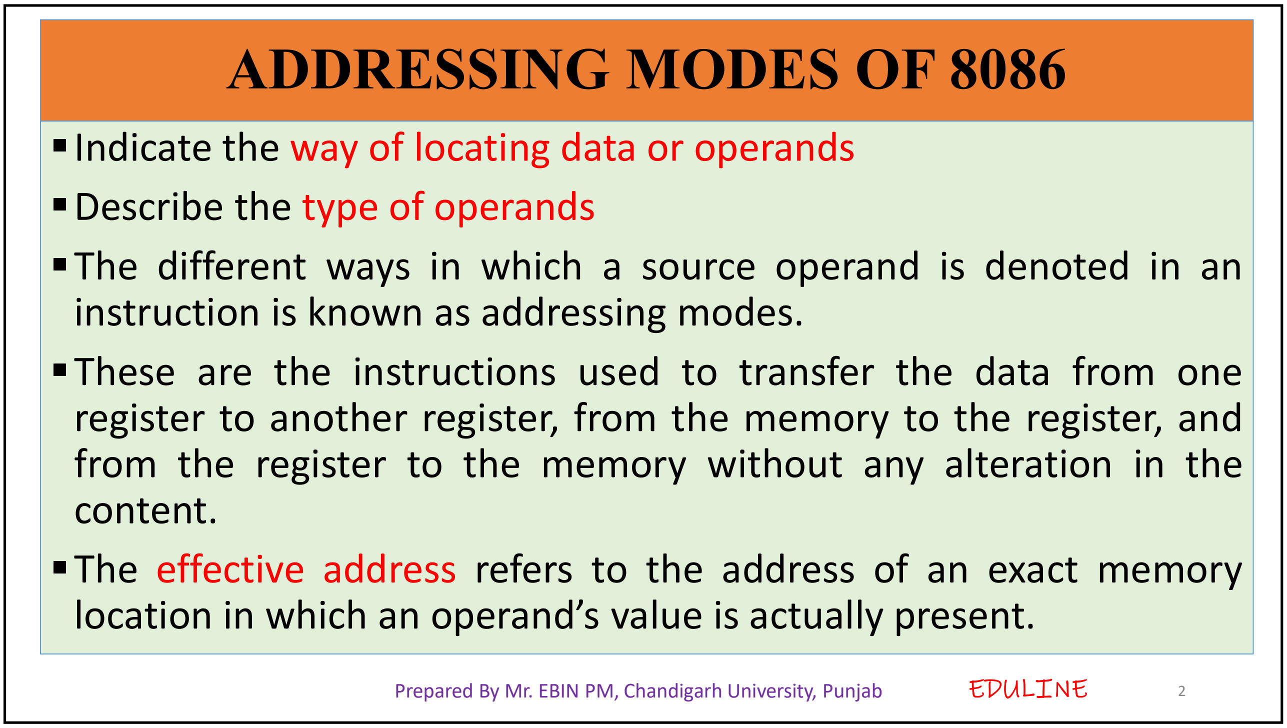


MODULE 2

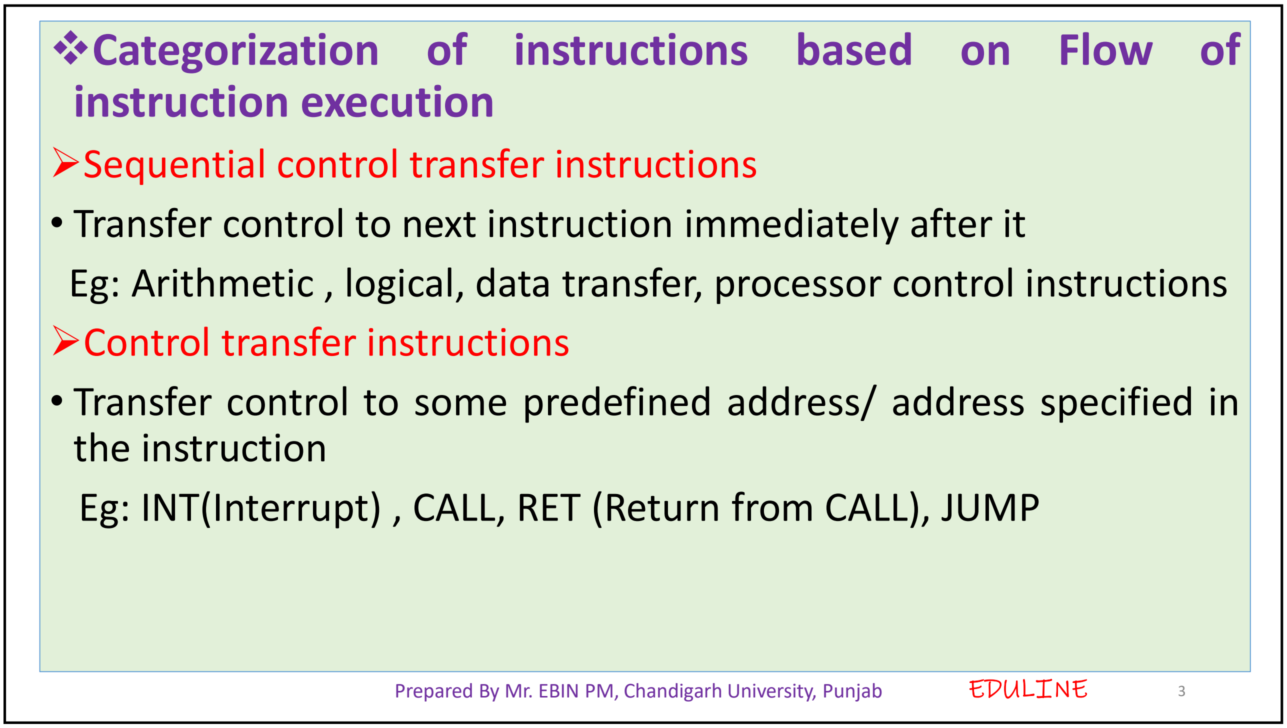
ADDRESSING MODES & INSTRUCTIONS

CO – Students will be able to design assembly language program with 8086



ADDRESSING MODES OF 8086

- Indicate the way of locating data or operands
- Describe the type of operands
- The different ways in which a source operand is denoted in an instruction is known as addressing modes.
- These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content.
- The effective address refers to the address of an exact memory location in which an operand's value is actually present.



❖Categorization of instructions based on Flow of instruction execution

➢Sequential control transfer instructions

- Transfer control to next instruction immediately after it

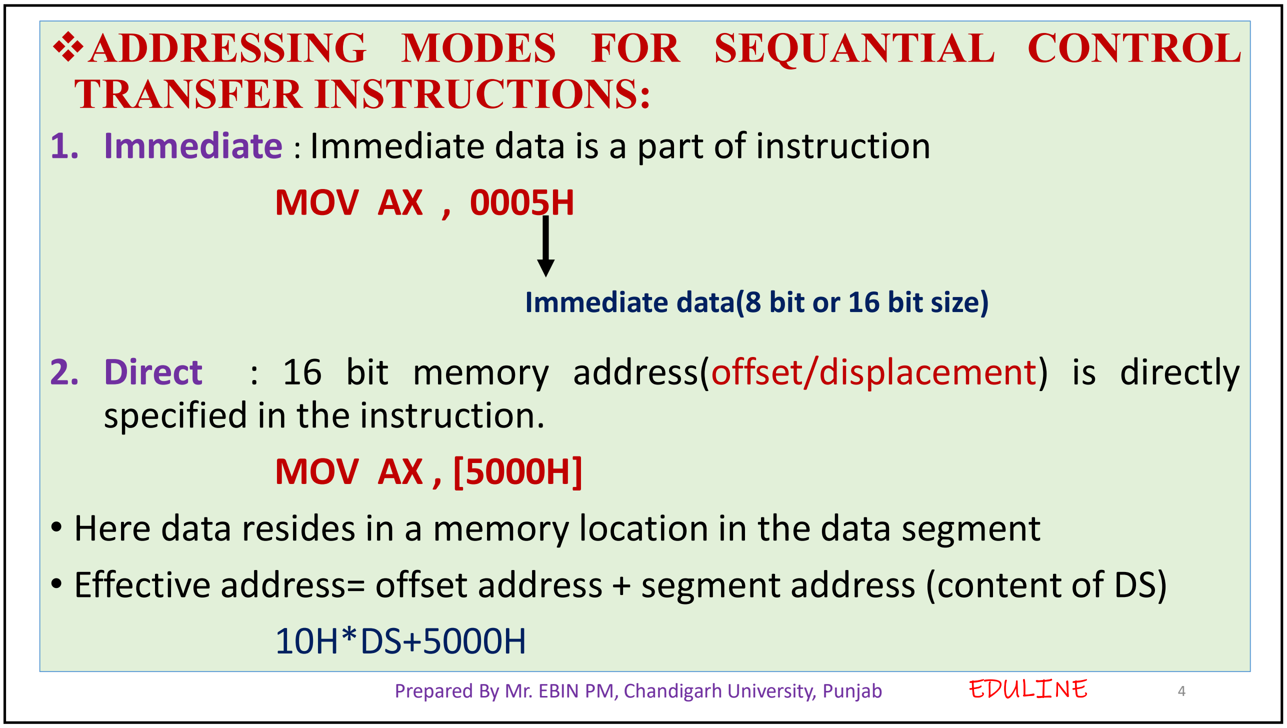
Eg: Arithmetic , logical, data transfer, processor control instructions

➢Control transfer instructions

- Transfer control to some predefined address/ address specified in the instruction

Eg: INT(Interrupt) , CALL, RET (Return from CALL), JUMP

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 3



❖ADDRESSING MODES FOR SEQUANTIAL CONTROL TRANSFER INSTRUCTIONS:

1. **Immediate** : Immediate data is a part of instruction

MOV AX , 0005H

↓

Immediate data(8 bit or 16 bit size)

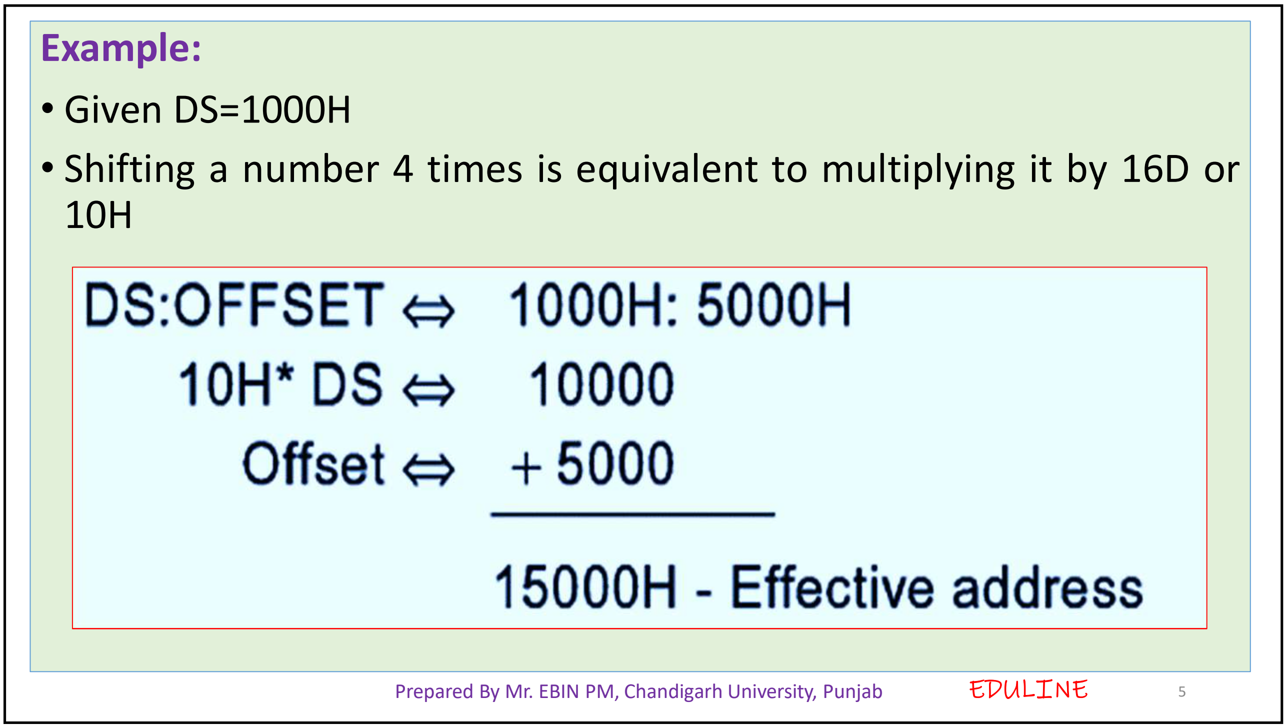
2. **Direct** : 16 bit memory address(offset/displacement) is directly specified in the instruction.

MOV AX , [5000H]

- Here data resides in a memory location in the data segment
- Effective address= offset address + segment address (content of DS)

10H*DS+5000H

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 4



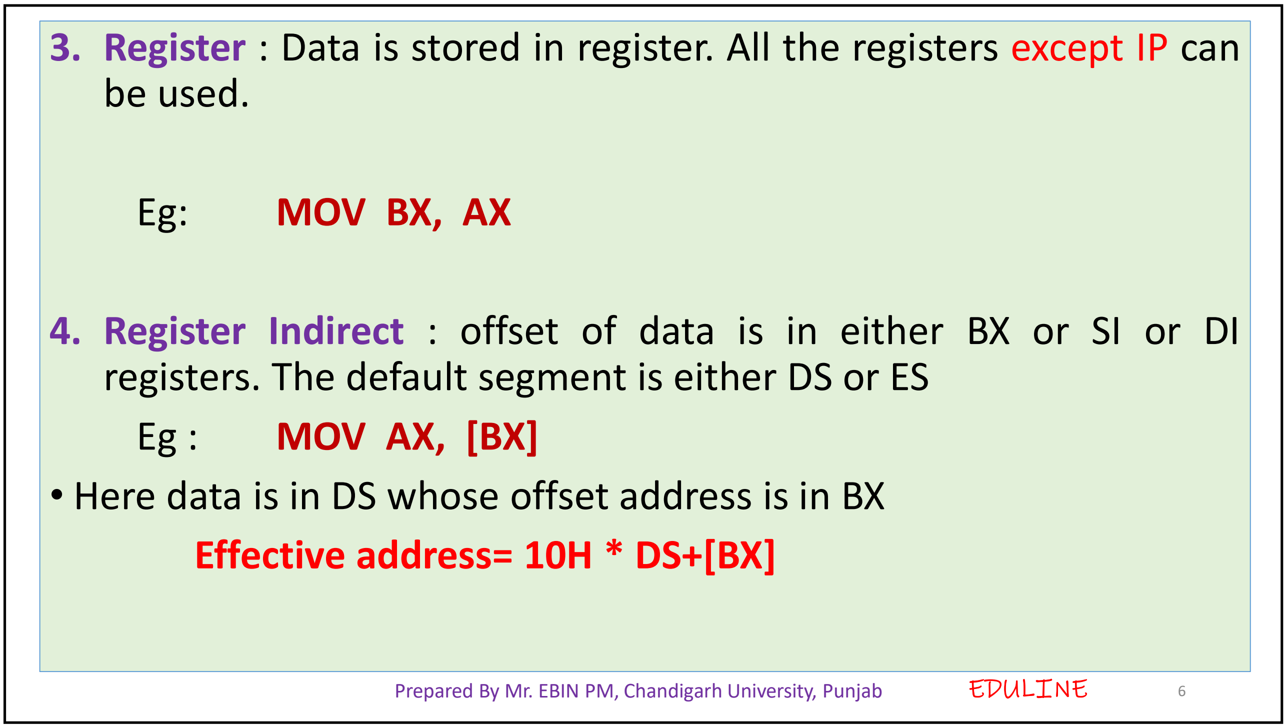
Example:

- Given DS=1000H
- Shifting a number 4 times is equivalent to multiplying it by 16D or 10H

```
DS:OFFSET ⇔   1000H: 5000H
   10H* DS ⇔    10000
    Offset ⇔  + 5000
             ____________

              15000H - Effective address
```

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 5



3. **Register** : Data is stored in register. All the registers except IP can be used.

   Eg: **MOV BX, AX**

4. **Register Indirect** : offset of data is in either BX or SI or DI registers. The default segment is either DS or ES

   Eg : **MOV AX, [BX]**

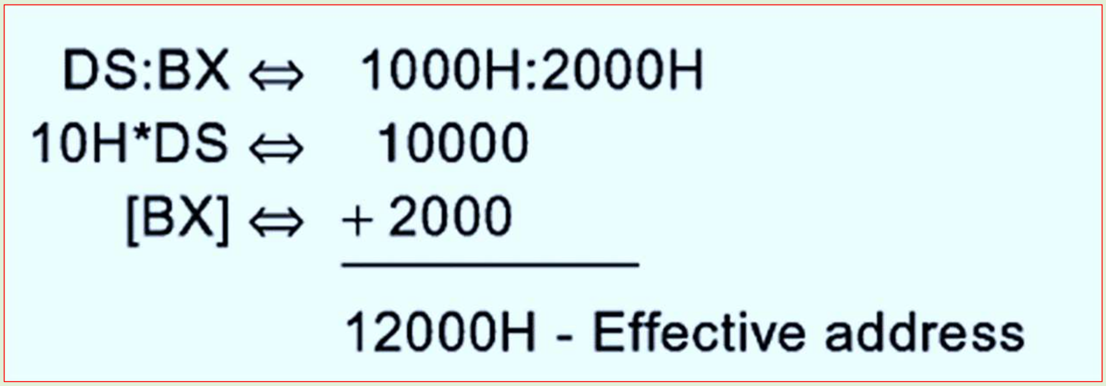
- Here data is in DS whose offset address is in BX

  Effective address= 10H * DS+[BX]

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 6

Example :

- Given DS=1000H and BX=2000H



$$\begin{array}{rcl} \text{DS:BX} & \Leftrightarrow & \text{1000H:2000H} \\ \text{10H*DS} & \Leftrightarrow & 10000 \\ \text{[BX]} & \Leftrightarrow & +\,2000 \\ \hline & & \text{12000H - Effective address} \end{array}$$

5. Indexed : offset of the operand is stored in one of the index registers.

- For SI (source index), default segment is DS
- For DI(destination index), default segment is ES

Eg: **MOV AX, [SI]**

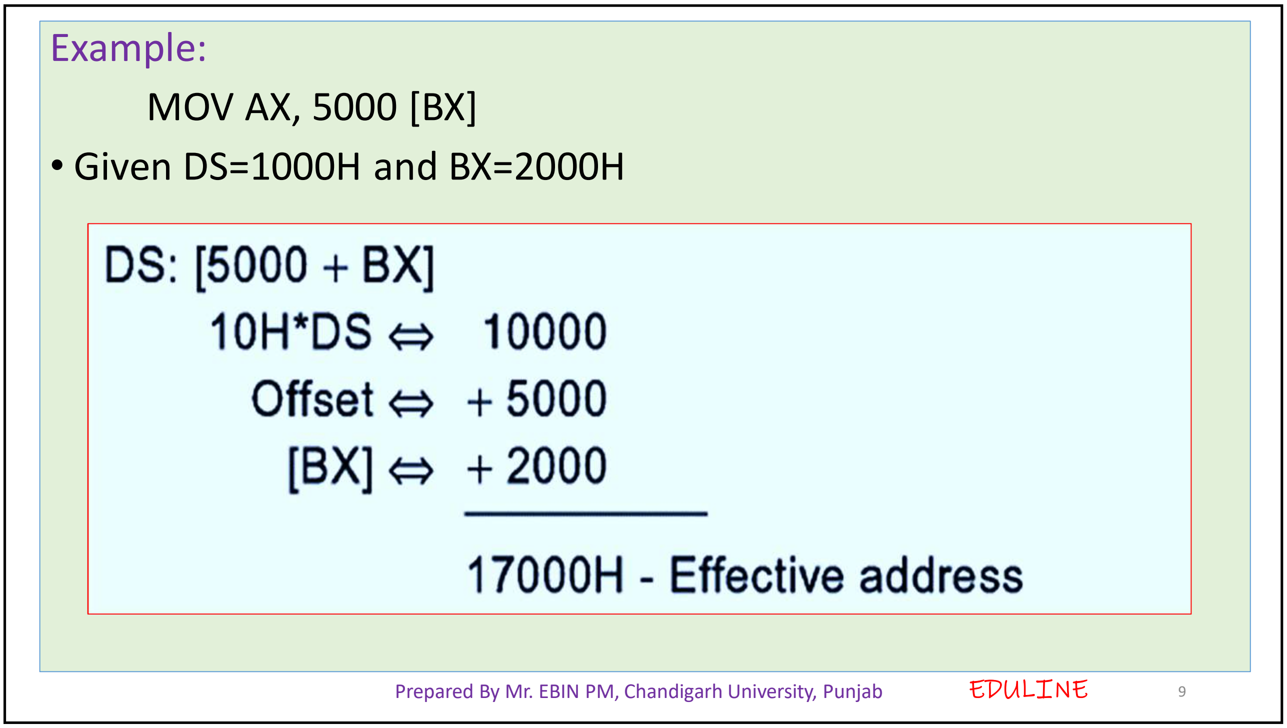
Effective address= 10H*DS+[SI]

6. Register relative : Data is available by adding the displacement with the content of any one of the register BX, BP, SI and DI

- Default segment is DS or ES

Eg: **MOV AX, 50H [BX]**

Effective address= 10H*DS+50H+[BX]



Example:

MOV AX, 5000 [BX]

- Given DS=1000H and BX=2000H

DS: [5000 + BX]

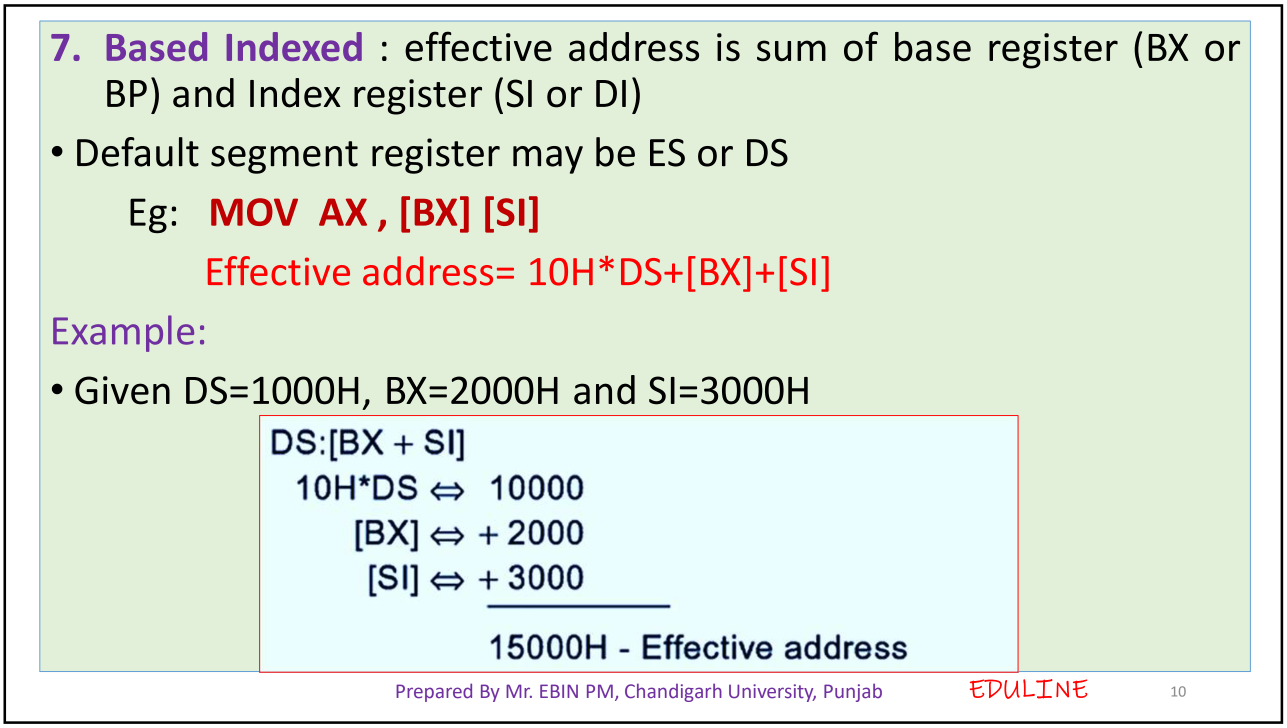
10H*DS ⇔ 10000

Offset ⇔ + 5000

[BX] ⇔ + 2000

17000H - Effective address

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 9



7. Based Indexed : effective address is sum of base register (BX or BP) and Index register (SI or DI)

- Default segment register may be ES or DS

Eg: **MOV AX , [BX] [SI]**

Effective address= 10H*DS+[BX]+[SI]

Example:

- Given DS=1000H, BX=2000H and SI=3000H

DS:[BX + SI]

10H*DS ⇔ 10000

[BX] ⇔ + 2000

[SI] ⇔ + 3000

15000H - Effective address

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 10

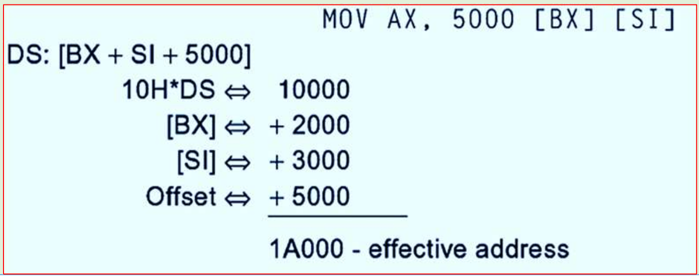
8. Relative Based Indexed : effective address is formed by adding displacement with the sum of content of any of base registers (BX or BP) and any one of the index registers

Eg: **MOV AX, 50H [BX] [SI]**

Effective address= 10H*DS+50H+[BX]+[SI]

Example:

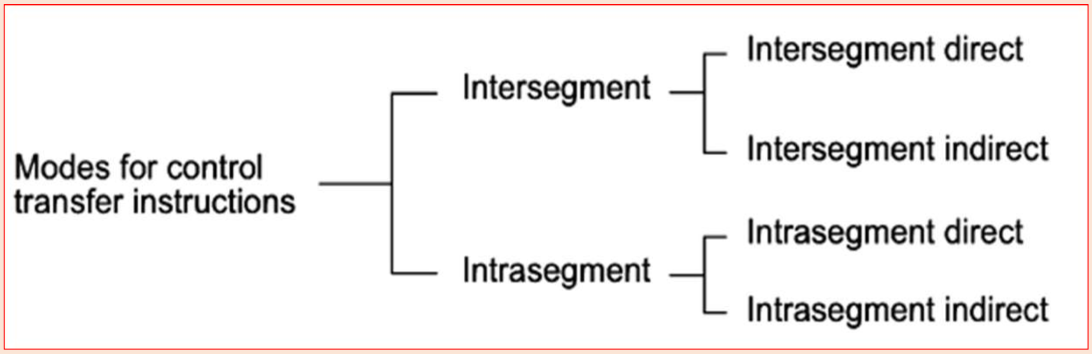
- Given DS=1000H, BX=2000H and SI=3000H



```
                      MOV AX, 5000 [BX] [SI]
DS: [BX + SI + 5000]
          10H*DS ⇔   10000
            [BX] ⇔ + 2000
            [SI] ⇔ + 3000
          Offset ⇔ + 5000
                   ___________
                   1A000 - effective address
```

ADDRESSING MODES FOR CONTROL TRANSFER INSTRUCTIONS

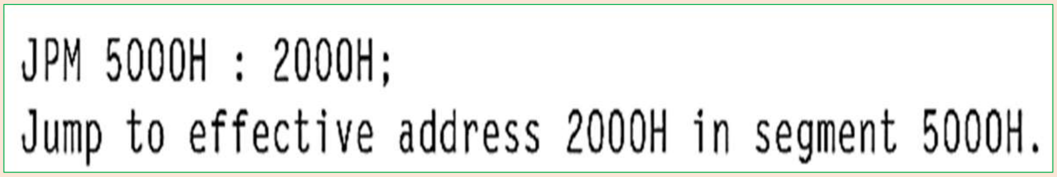
- Intersegment: Destination location is in different segment.
- Intrasegment: Destination location is in same segment.



❖Intersegment Direct

- Destination is in different segment
- Provides branching from one code segment to another code segment
- CS and IP of destination address are specified directly in the instruction.

Example:



```
JPM 5000H : 2000H;
Jump to effective address 2000H in segment 5000H.
```

❖Intersegment Indirect

- Destination lies in different segment
- Destination location is passed to the instruction indirectly.

Example:

```
JMP [2000H];
```

- Jump to an address in the other segment specified at effective address 2000H in DS

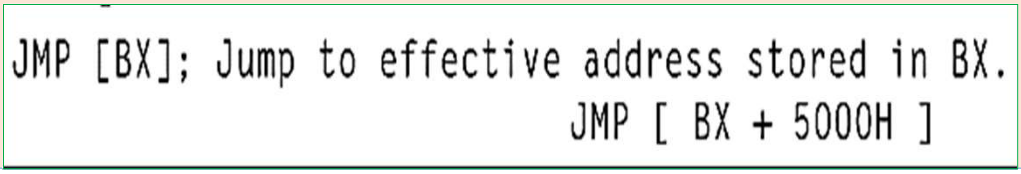
❖Intrasegment Direct

- Destination lies in same segment
- Displacement is computed using the content of the IP

❖Intrasegment Indirect

- Destination lies in same segment
- Destination location is passed to the instruction indirectly.
- Branch address is found as the content of a register

Example



```
JMP [BX]; Jump to effective address stored in BX.
                              JMP [ BX + 5000H ]
```

INSTRUCTION SET OF 8086

1. Data copy/ transfer instruction

- Transfer data from source operand to destination operand

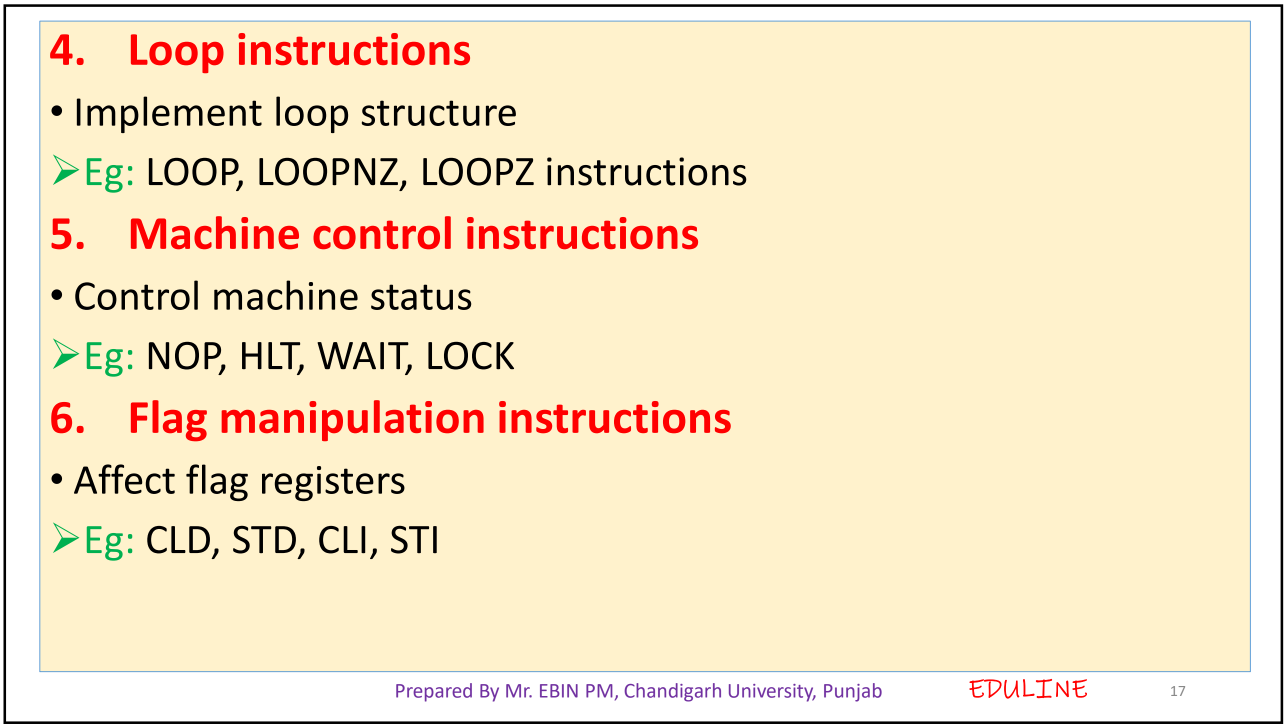
➢Eg: store, move , load, exchange, I/O instructions

2. Arithmetic & Logical instructions

3. Branch instructions

- Transfer control of execution to a specified address

➢Eg: jump, interrupt, call, return instruction



4. Loop instructions

- Implement loop structure

➢Eg: LOOP, LOOPNZ, LOOPZ instructions

5. Machine control instructions

- Control machine status

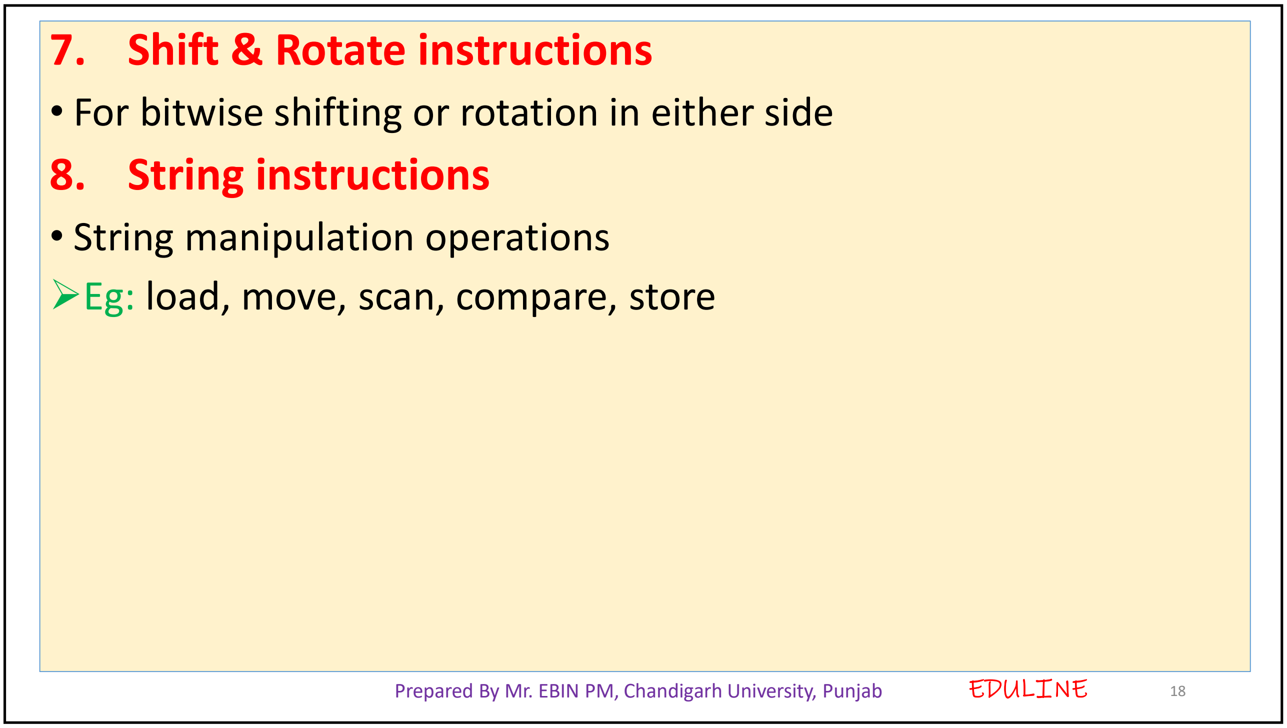
➢Eg: NOP, HLT, WAIT, LOCK

6. Flag manipulation instructions

- Affect flag registers

➢Eg: CLD, STD, CLI, STI

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 17



7. Shift & Rotate instructions

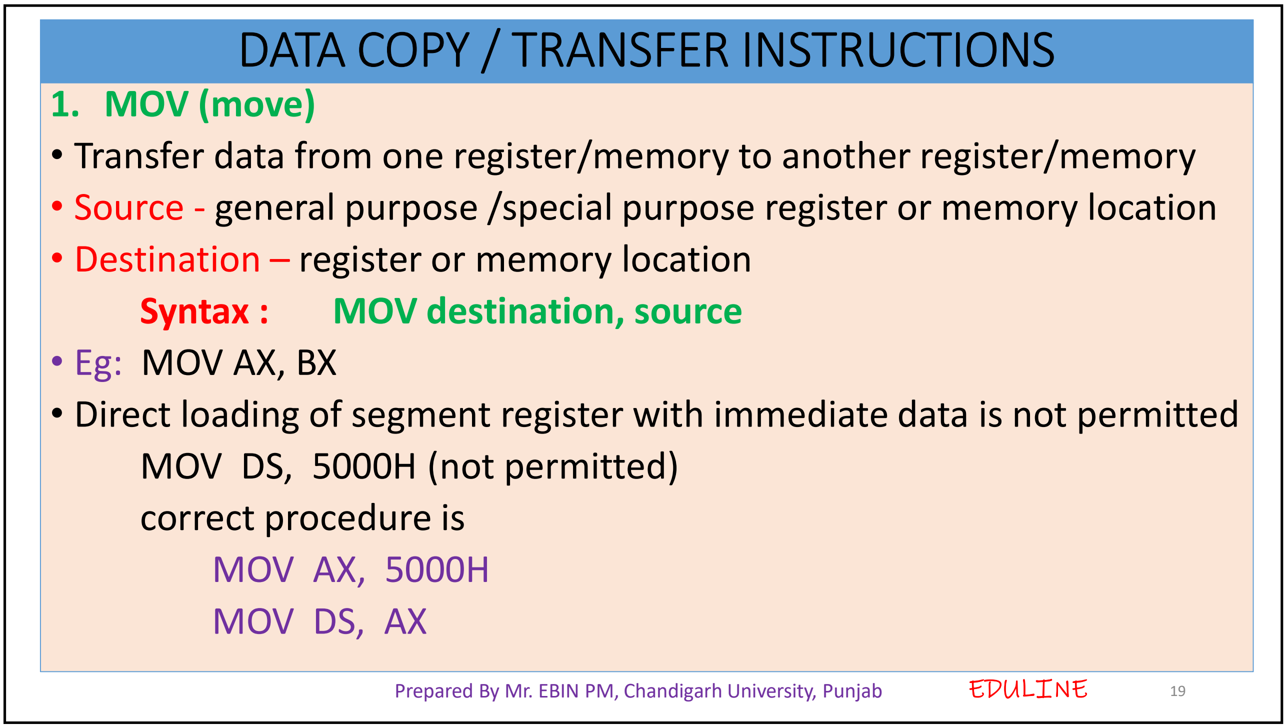
- For bitwise shifting or rotation in either side

8. String instructions

- String manipulation operations

➢Eg: load, move, scan, compare, store

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 18



DATA COPY / TRANSFER INSTRUCTIONS

1. MOV (move)

- Transfer data from one register/memory to another register/memory
- Source - general purpose /special purpose register or memory location
- Destination – register or memory location

Syntax : **MOV destination, source**

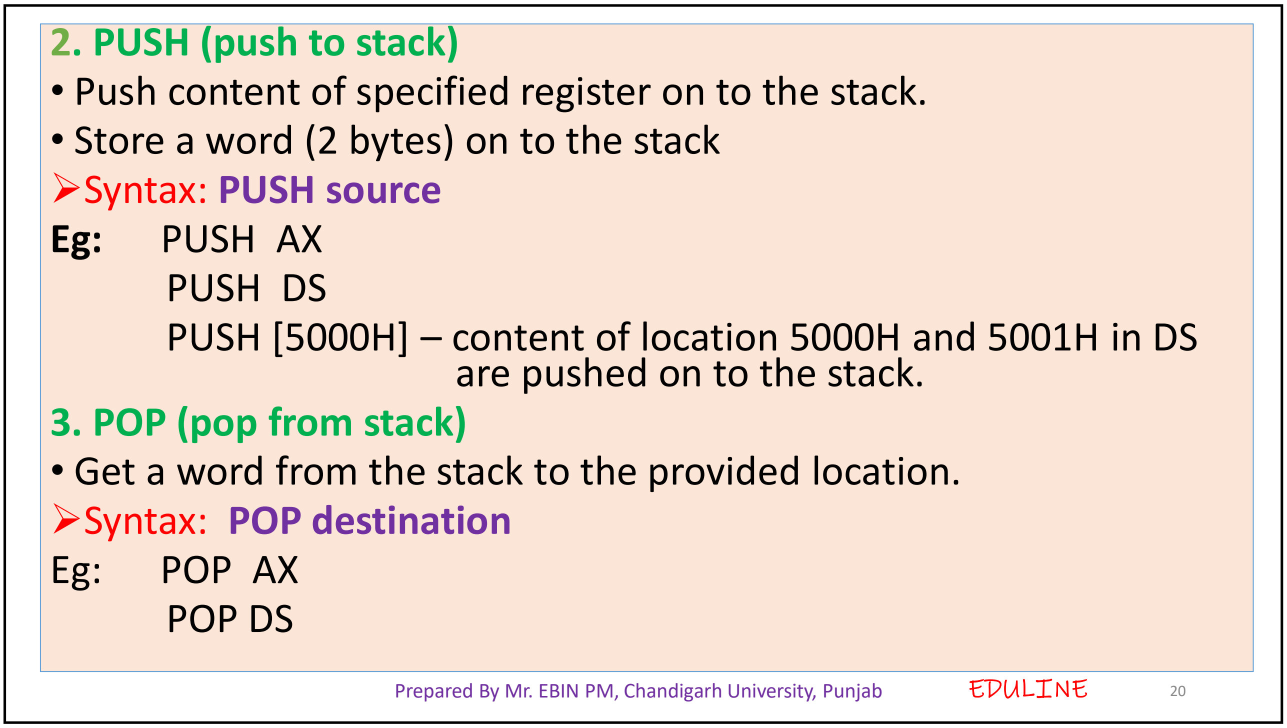
- Eg: MOV AX, BX
- Direct loading of segment register with immediate data is not permitted

MOV DS, 5000H (not permitted)

correct procedure is

```
MOV  AX,  5000H
MOV  DS,  AX
```

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 19



2. PUSH (push to stack)

- Push content of specified register on to the stack.
- Store a word (2 bytes) on to the stack

➢Syntax: **PUSH source**

Eg: PUSH AX

PUSH DS

PUSH [5000H] – content of location 5000H and 5001H in DS are pushed on to the stack.

3. POP (pop from stack)

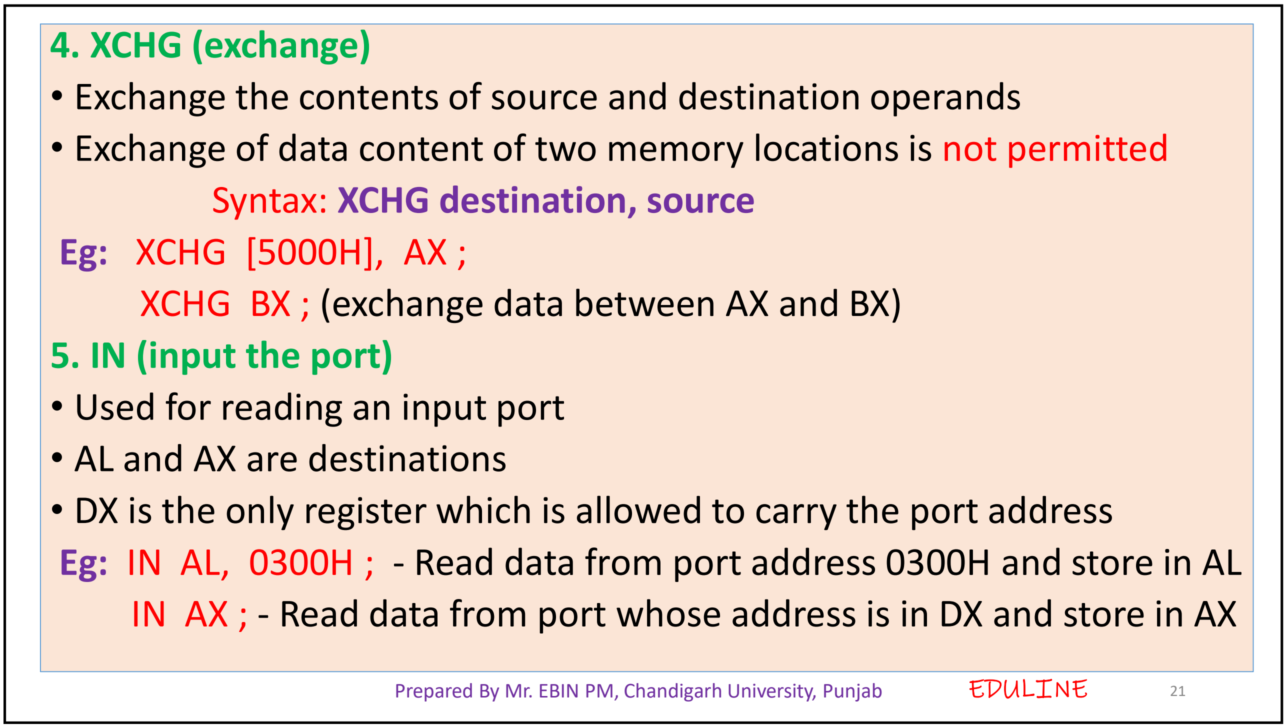
- Get a word from the stack to the provided location.

➢Syntax: **POP destination**

Eg: POP AX

POP DS

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 20



4. XCHG (exchange)

- Exchange the contents of source and destination operands
- Exchange of data content of two memory locations is not permitted

Syntax: **XCHG destination, source**

Eg: XCHG [5000H], AX ;

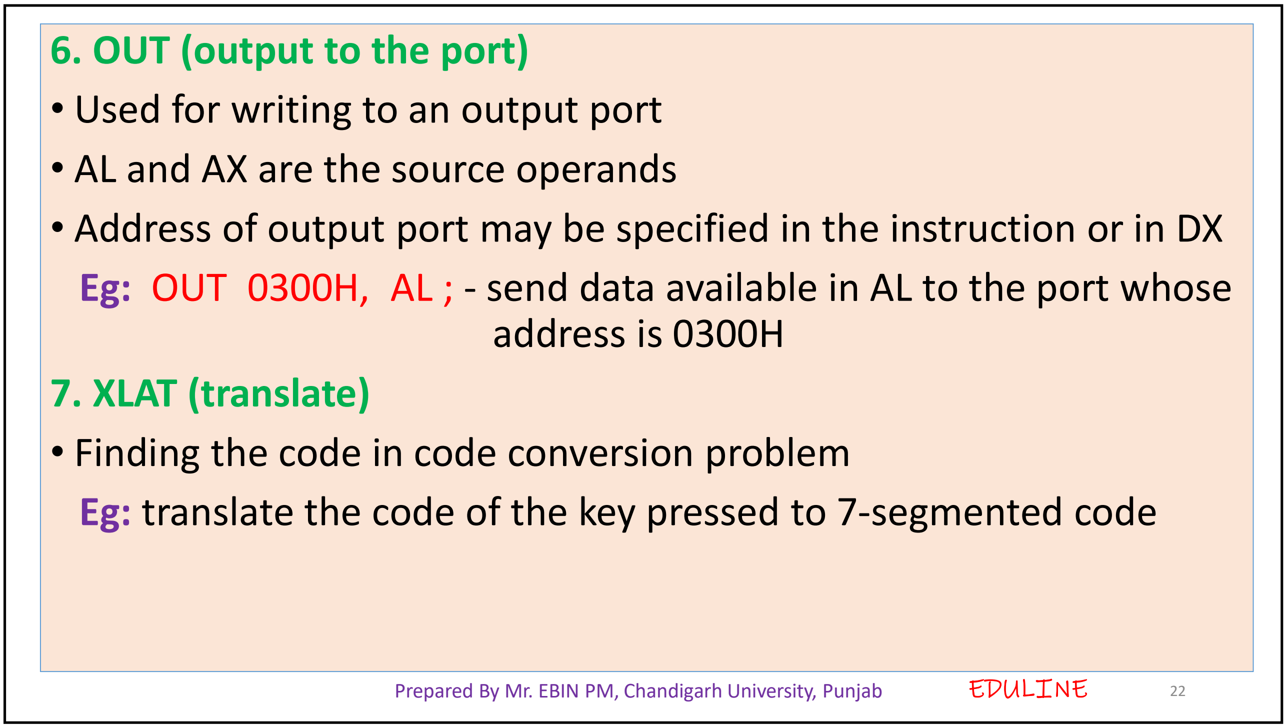
XCHG BX ; (exchange data between AX and BX)

5. IN (input the port)

- Used for reading an input port
- AL and AX are destinations
- DX is the only register which is allowed to carry the port address

Eg: IN AL, 0300H ; - Read data from port address 0300H and store in AL

IN AX ; - Read data from port whose address is in DX and store in AX



6. OUT (output to the port)

- Used for writing to an output port
- AL and AX are the source operands
- Address of output port may be specified in the instruction or in DX

Eg: OUT 0300H, AL ; - send data available in AL to the port whose address is 0300H

7. XLAT (translate)

- Finding the code in code conversion problem

Eg: translate the code of the key pressed to 7-segmented code

8. LEA (Load Effective Address)

- Load offset of an operand in specified register
- Used to save pointer(address) of a value

Eg: LEA BX, ADR; - offset of label ADR will be transferred to register BX

9. PUSHF (push Flag to stack)

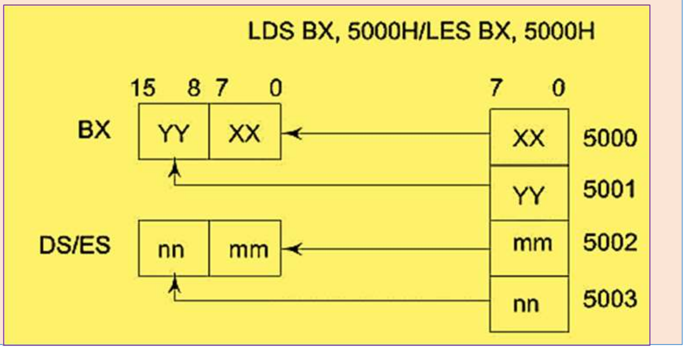
- Pushes the flag register on to the stack
- First the upper byte then the lower byte will push on to the stack
- SP is decremented by 2 , for each push operation

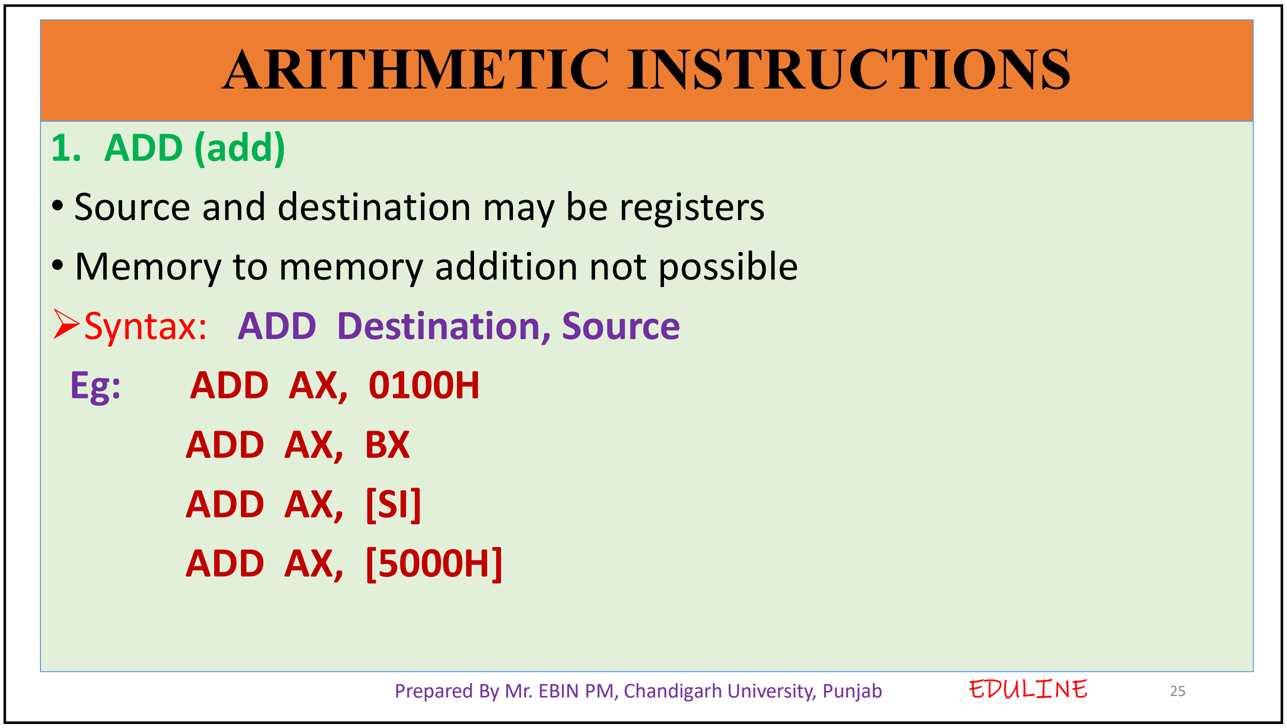
10. POPF (pop Flag from stack)

- Used to copy a word at the top of the stack to the flag register.
- SP is incremented by 2 ,for each pop operation

11. LDS/LES (Load pointer to DS/ES)

- It loads DS or ES register & destination register with the content of memory location(source)





ARITHMETIC INSTRUCTIONS

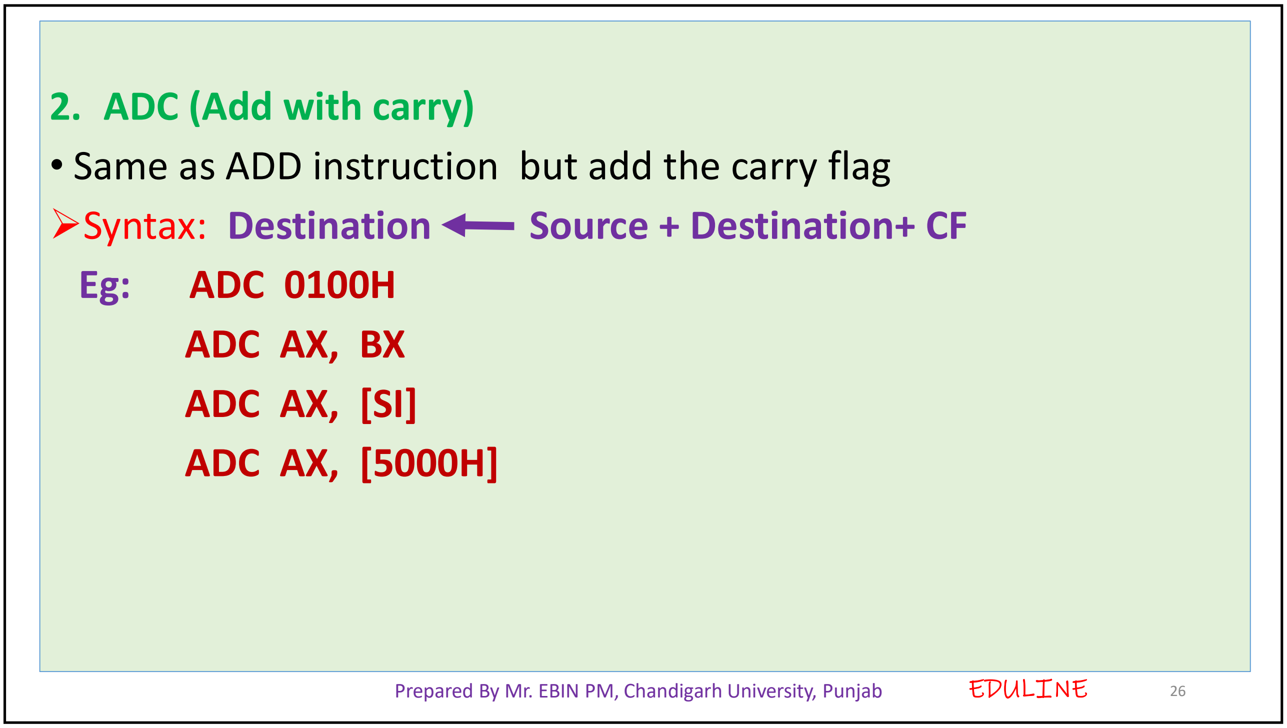
1. ADD (add)

- Source and destination may be registers
- Memory to memory addition not possible

➢Syntax: ADD Destination, Source

Eg: ADD AX, 0100H
ADD AX, BX
ADD AX, [SI]
ADD AX, [5000H]

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 25



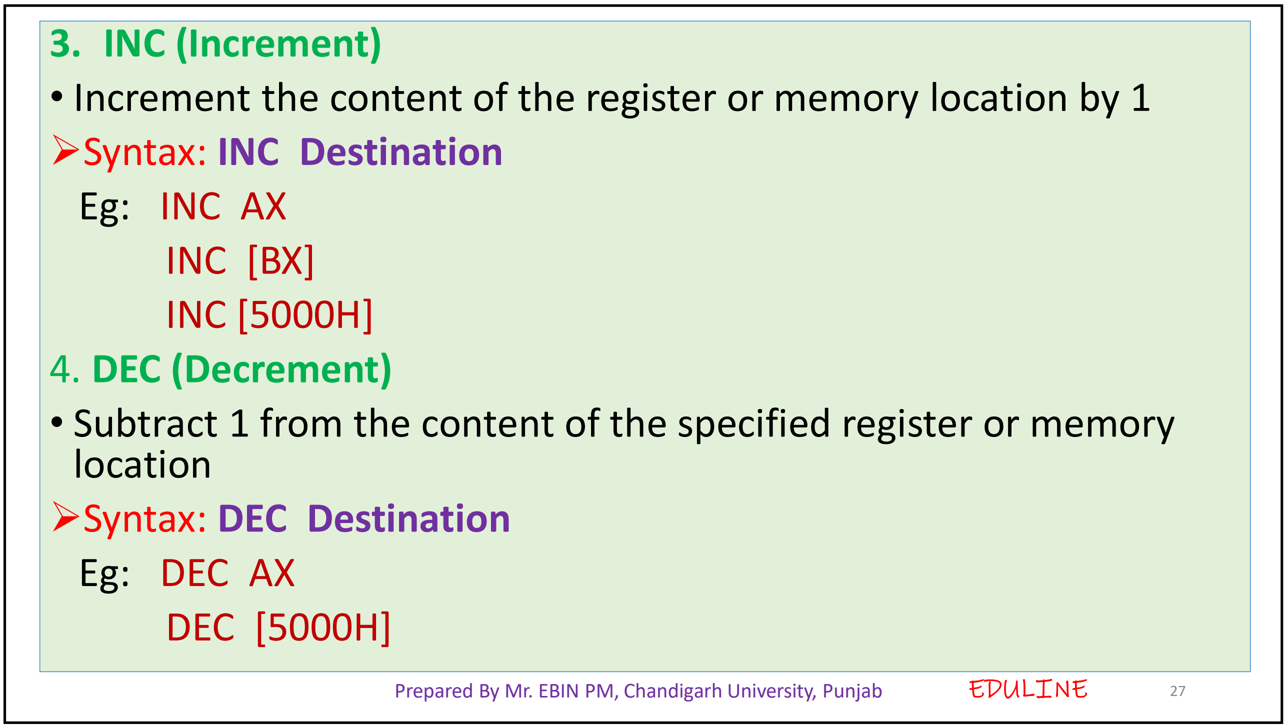
2. ADC (Add with carry)

- Same as ADD instruction but add the carry flag

➢Syntax: Destination ← Source + Destination+ CF

Eg: ADC 0100H
ADC AX, BX
ADC AX, [SI]
ADC AX, [5000H]

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 26



3. INC (Increment)

- Increment the content of the register or memory location by 1

➢Syntax: **INC Destination**

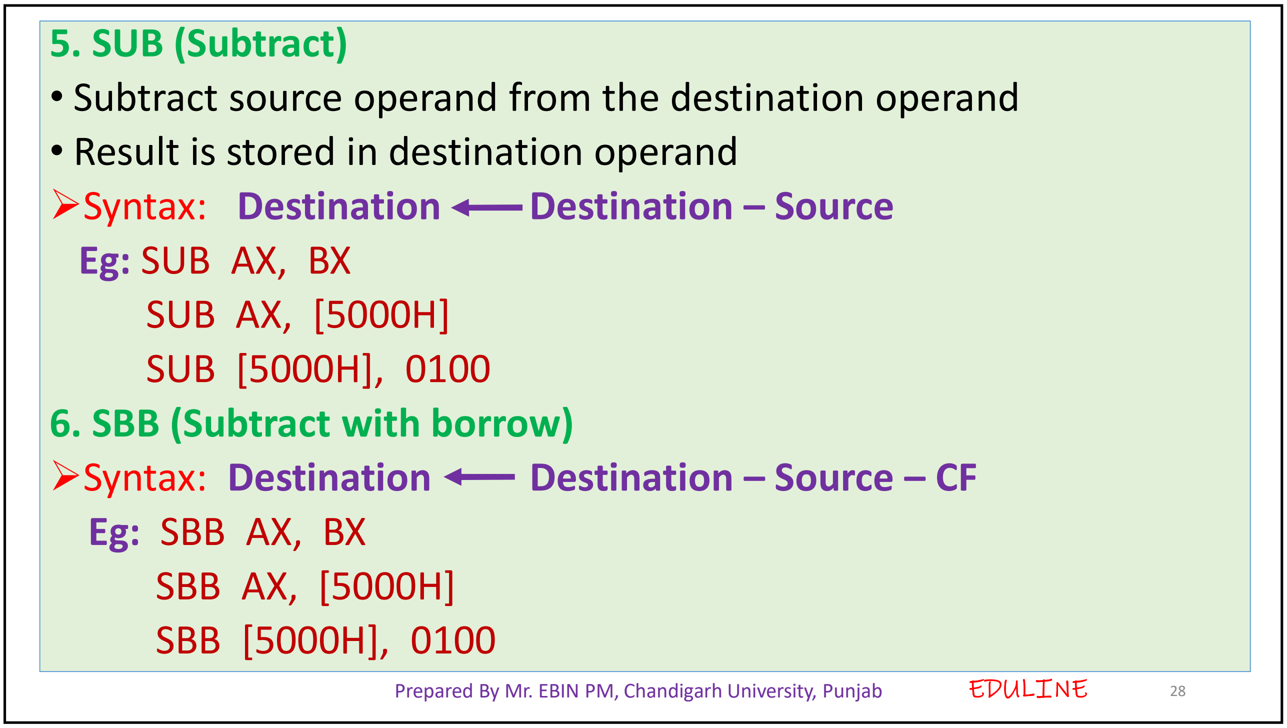
Eg: INC AX
INC [BX]
INC [5000H]

4. DEC (Decrement)

- Subtract 1 from the content of the specified register or memory location

➢Syntax: **DEC Destination**

Eg: DEC AX
DEC [5000H]



5. SUB (Subtract)

- Subtract source operand from the destination operand
- Result is stored in destination operand

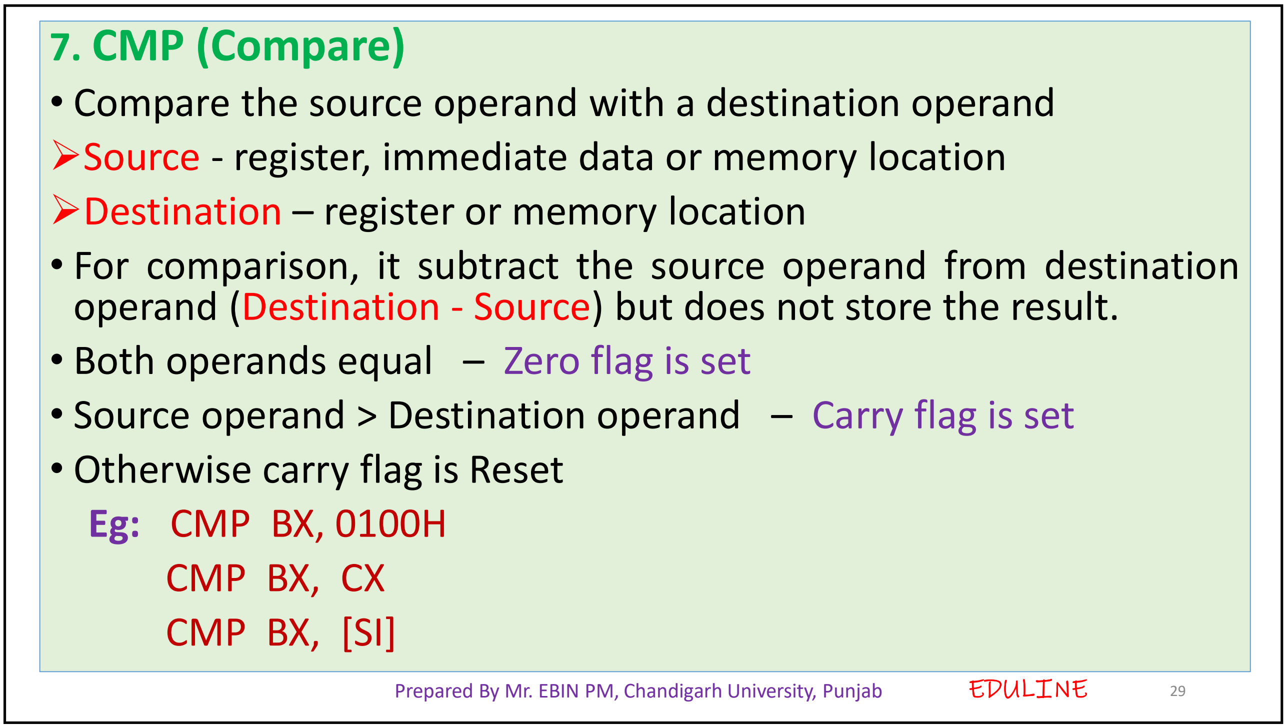
➢Syntax: **Destination ⟵ Destination – Source**

Eg: SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100

6. SBB (Subtract with borrow)

➢Syntax: **Destination ⟵ Destination – Source – CF**

Eg: SBB AX, BX
SBB AX, [5000H]
SBB [5000H], 0100



7. CMP (Compare)

- Compare the source operand with a destination operand

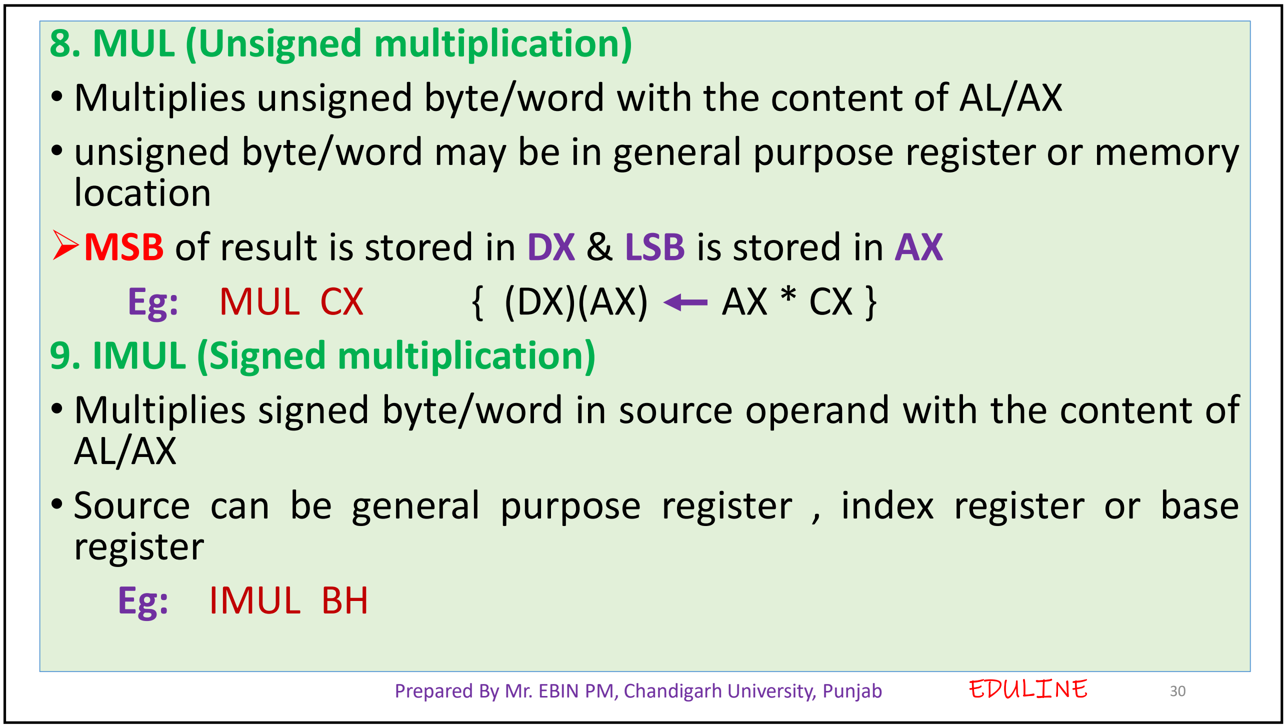
➢Source - register, immediate data or memory location

➢Destination – register or memory location

- For comparison, it subtract the source operand from destination operand (Destination - Source) but does not store the result.
- Both operands equal – Zero flag is set
- Source operand > Destination operand – Carry flag is set
- Otherwise carry flag is Reset

Eg:
```
CMP  BX, 0100H
CMP  BX,  CX
CMP  BX,  [SI]
```

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 29



8. MUL (Unsigned multiplication)

- Multiplies unsigned byte/word with the content of AL/AX
- unsigned byte/word may be in general purpose register or memory location

➢**MSB** of result is stored in **DX** & **LSB** is stored in **AX**

Eg: MUL CX { (DX)(AX) ← AX * CX }

9. IMUL (Signed multiplication)

- Multiplies signed byte/word in source operand with the content of AL/AX
- Source can be general purpose register , index register or base register

Eg: IMUL BH

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 30

10. DIV (Unsigned Division)

- Divides unsigned word / double word by 16 bit/8 bit operand

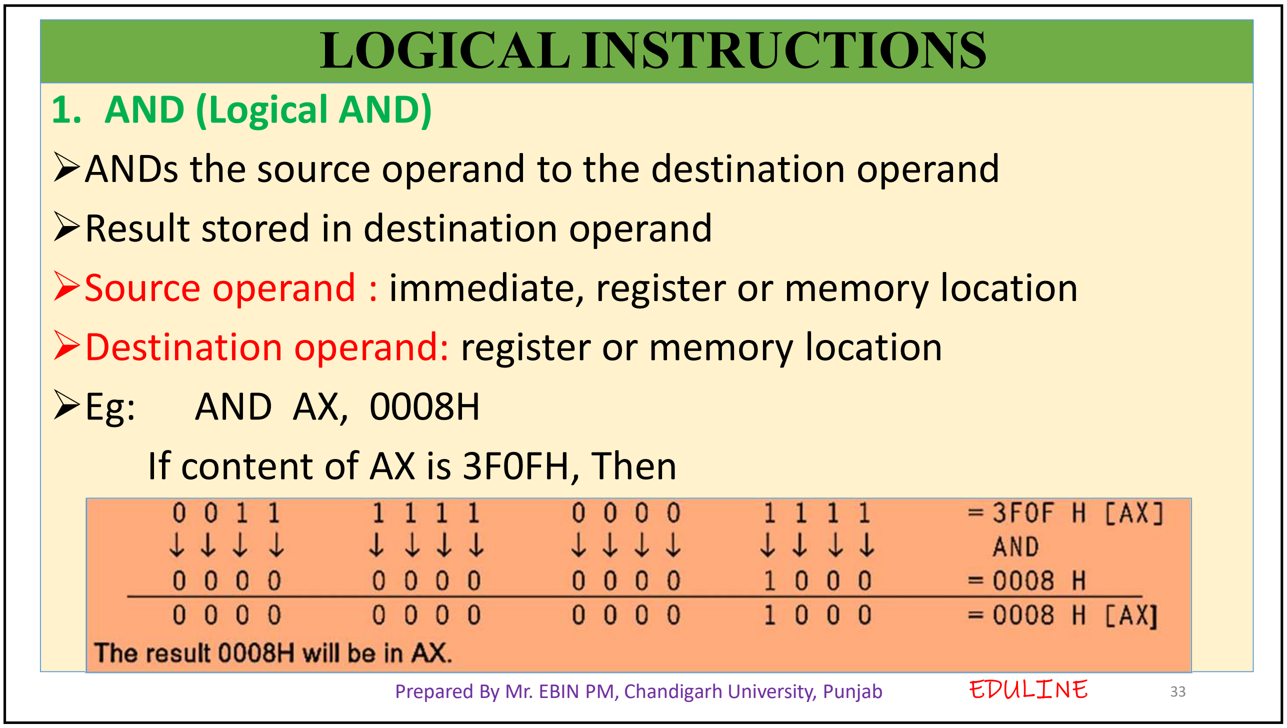
➢Quotient will be stored in AL

➢Reminder will be stored in AH

11. IDIV (Signed Division)

- Same as DIV instruction but signed
- Result will also be signed number

- AAA – ASCII ADJUST AFTER ADDITION
- AAS – ASCII ADJUST AFTER SUBTRACTION
- AAM – ASCII ADJUST FOR MULTIPLICATION
- AAD – ASCII ADJUST FOR DIVISION
- DAA – DECIMAL ADJUST ACCUMULATOR
- DAS –DECIMAL ADJUST AFTER SUBTRACTION
- NEG – NEGATE (To find 2’s complement)



LOGICAL INSTRUCTIONS

1. AND (Logical AND)

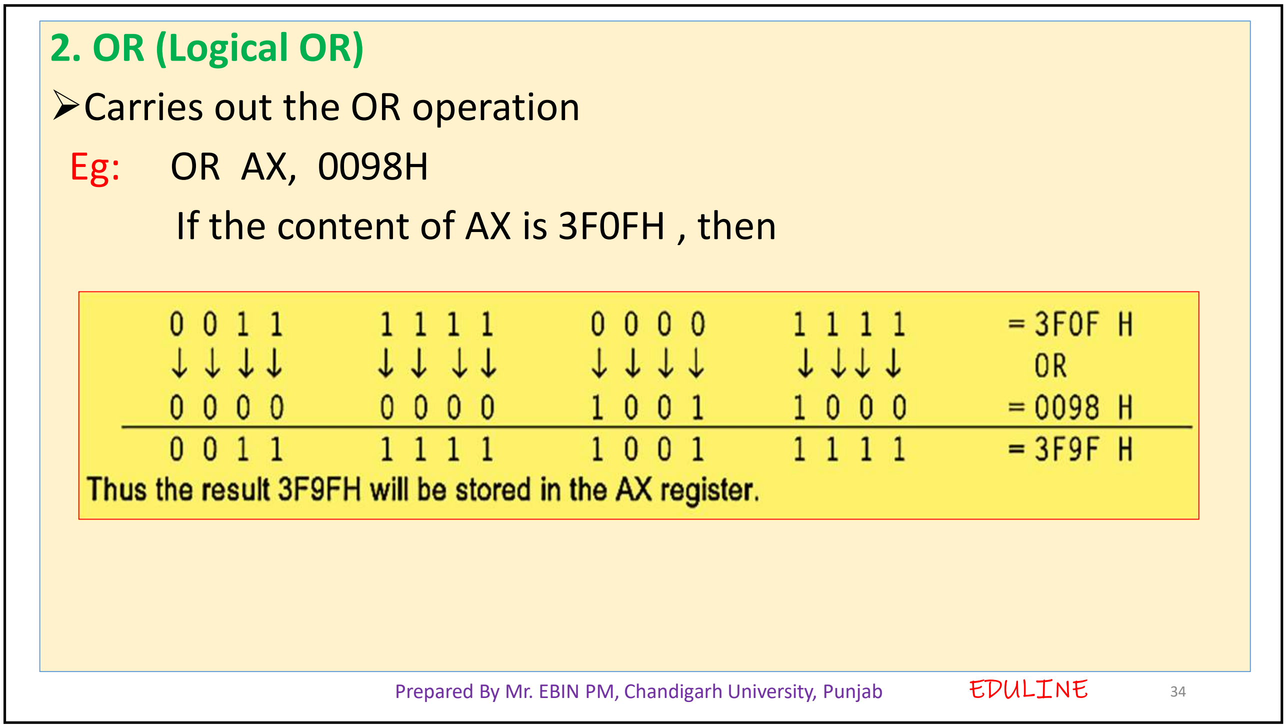
- ANDs the source operand to the destination operand
- Result stored in destination operand
- Source operand : immediate, register or memory location
- Destination operand: register or memory location
- Eg: AND AX, 0008H

  If content of AX is 3F0FH, Then

```
 0 0 1 1     1 1 1 1     0 0 0 0     1 1 1 1     = 3F0F H [AX]
 ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓       AND
 0 0 0 0     0 0 0 0     0 0 0 0     1 0 0 0     = 0008 H
-------------------------------------------------------------
 0 0 0 0     0 0 0 0     0 0 0 0     1 0 0 0     = 0008 H [AX]
```

The result 0008H will be in AX.

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 33



2. OR (Logical OR)

- Carries out the OR operation

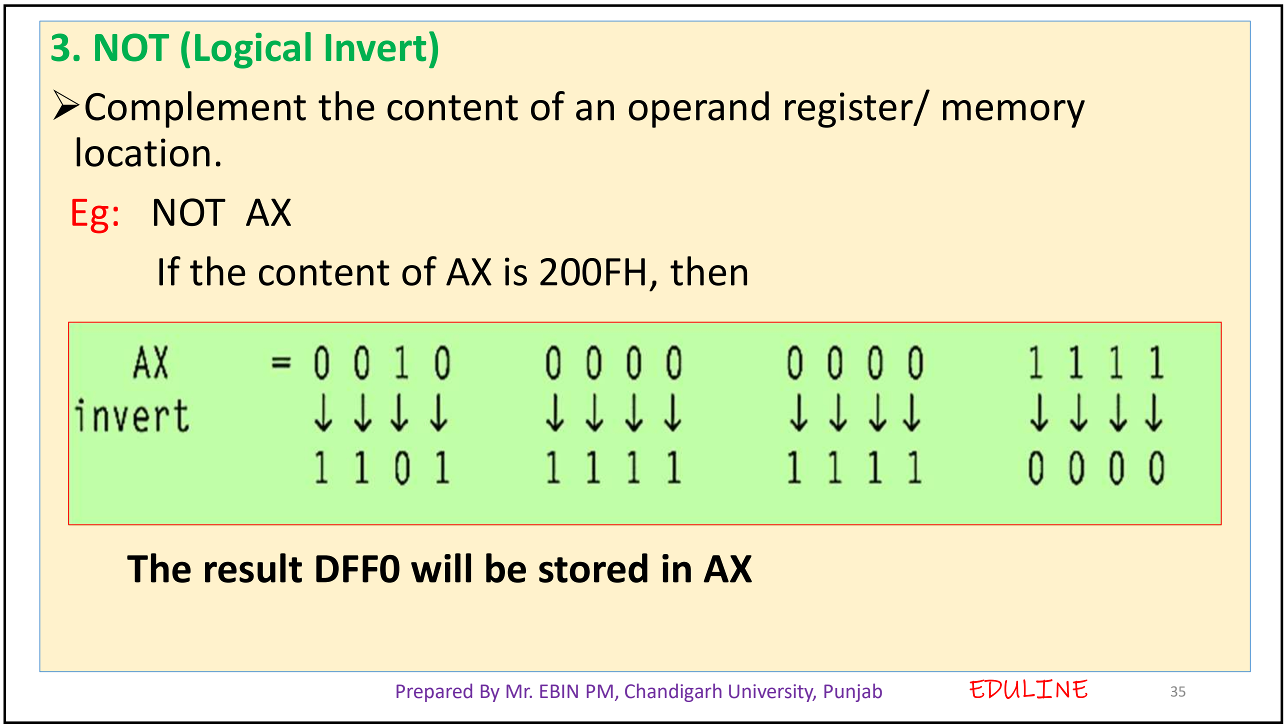
Eg: OR AX, 0098H

If the content of AX is 3F0FH , then

```
 0 0 1 1     1 1 1 1     0 0 0 0     1 1 1 1     = 3F0F H
 ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓       OR
 0 0 0 0     0 0 0 0     1 0 0 1     1 0 0 0     = 0098 H
-------------------------------------------------------------
 0 0 1 1     1 1 1 1     1 0 0 1     1 1 1 1     = 3F9F H
```

Thus the result 3F9FH will be stored in the AX register.

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 34



3. NOT (Logical Invert)

- Complement the content of an operand register/ memory location.

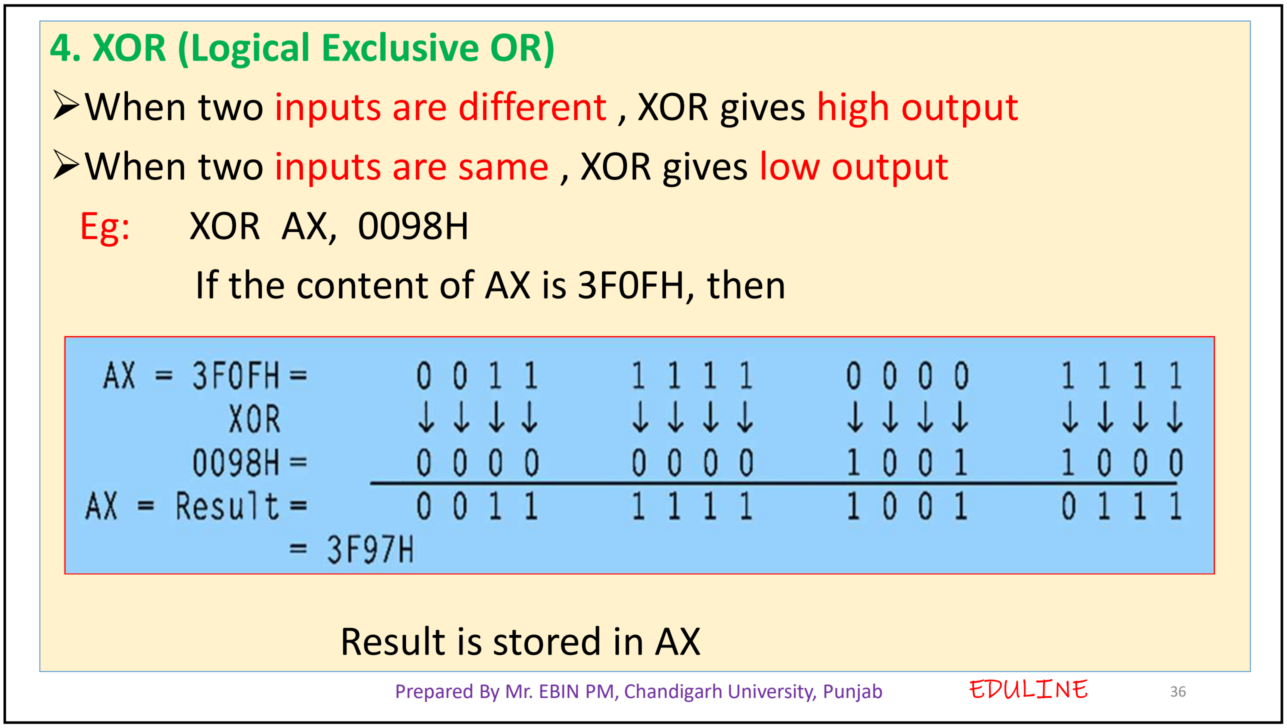
Eg: NOT AX

If the content of AX is 200FH, then

```
  AX      = 0 0 1 0    0 0 0 0    0 0 0 0    1 1 1 1
invert      ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓
            1 1 0 1    1 1 1 1    1 1 1 1    0 0 0 0
```

The result DFF0 will be stored in AX

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 35



4. XOR (Logical Exclusive OR)

- When two inputs are different , XOR gives high output
- When two inputs are same , XOR gives low output

Eg: XOR AX, 0098H

If the content of AX is 3F0FH, then

```
 AX = 3F0FH =    0 0 1 1    1 1 1 1    0 0 0 0    1 1 1 1
         XOR     ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓
      0098H =    0 0 0 0    0 0 0 0    1 0 0 1    1 0 0 0
               ------------------------------------------
AX = Result =    0 0 1 1    1 1 1 1    1 0 0 1    0 1 1 1
            = 3F97H
```

Result is stored in AX

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 36

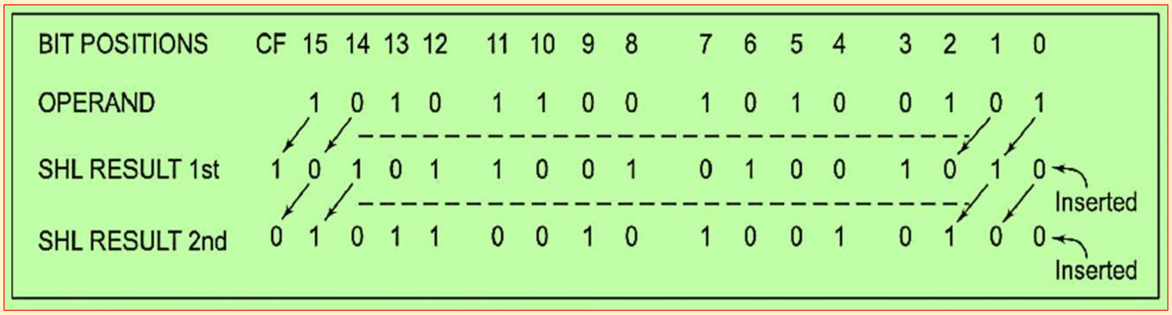
5. TEST (Logical Compare instruction)

- Performs bit by bit logical AND operation
- If both the operands are 1, result is set to 1 else 0
- The result is not available for further use
- Affected flags are CF, SF, ZF and PF

Eg: TEST AX, BX

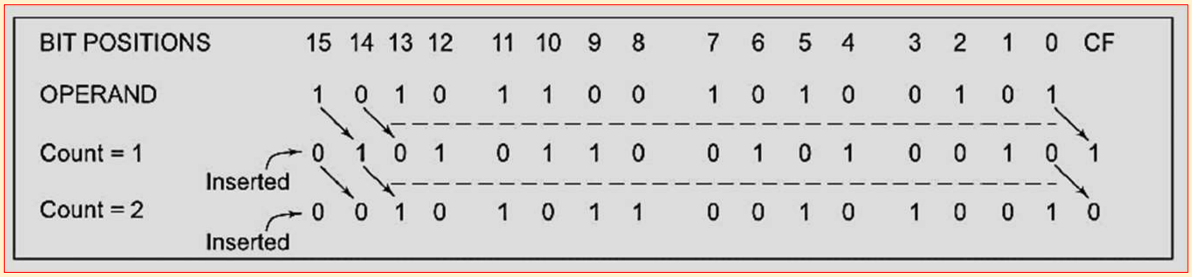
6. SHL/ SAL (Shift Logical / Arithmetic Left)

- Shift the operand bit by bit to the left and insert 0 in LSB
- The count is either 1 or specified by register CL



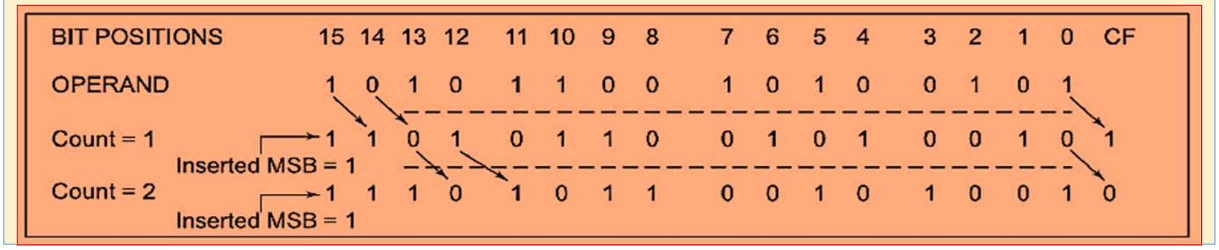
BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OPERAND		1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	
SHL RESULT 1st	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	0	Inserted
SHL RESULT 2nd	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	0	0	Inserted

7. SHR (Shift Logical Right)



BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	
Count = 1	0	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1
Count = 2	0	0	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0

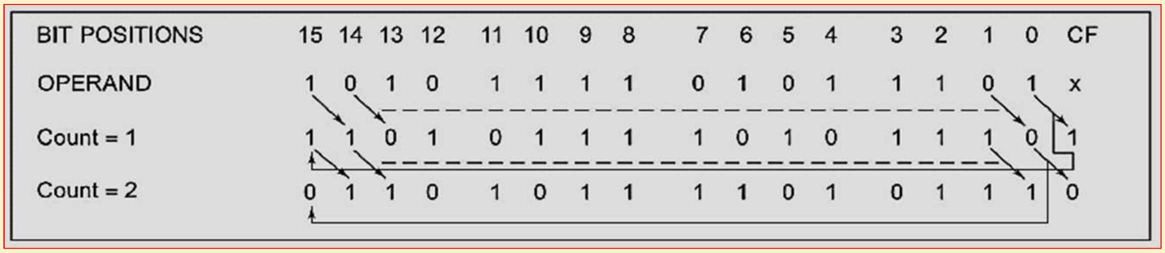
8. SAR (Shift Arithmetic Right)



BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1	
Count = 1	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	1
Count = 2	1	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0

9. ROR (Rotate Right without carry)

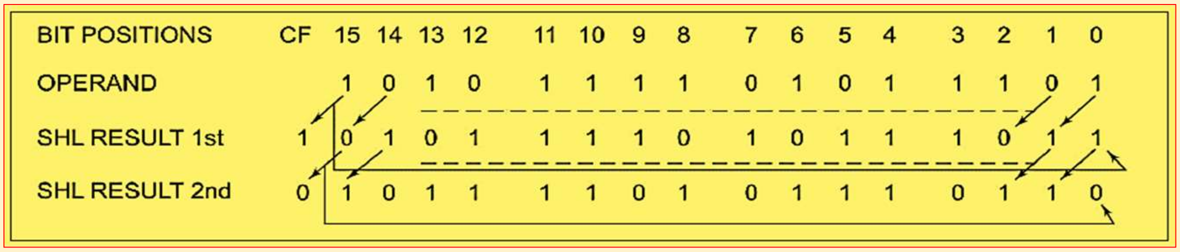
- The Least Significant bit is pushed in to the carry flag and simultaneously it is transferred in to the MSB position



BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF
OPERAND	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	x
Count = 1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1
Count = 2	0	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0

10 . ROL (Rotate Left without carry)

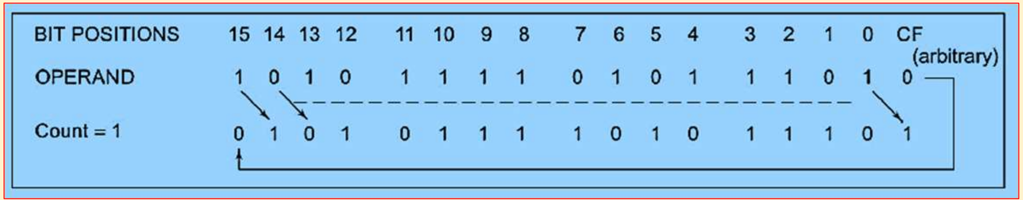
- The Most Significant Bit is pushed in to the carry flag as well as LSB position



BIT POSITIONS	CF	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERAND		1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1
SHL RESULT 1st	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1
SHL RESULT 2nd	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1	0

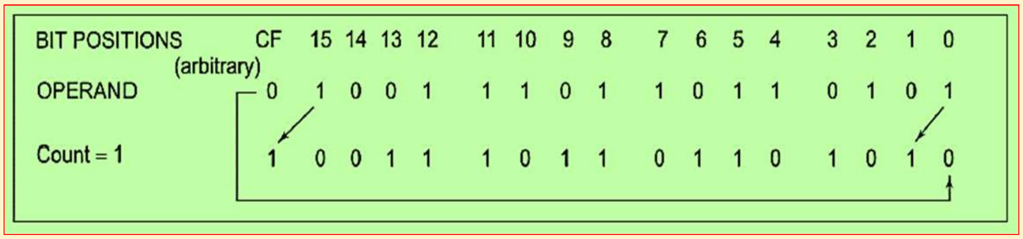
11. RCR (Rotate Right through carry)

- Carry flag is pushed in to the MSB of the operand and the LSB is pushed in to carry flag.

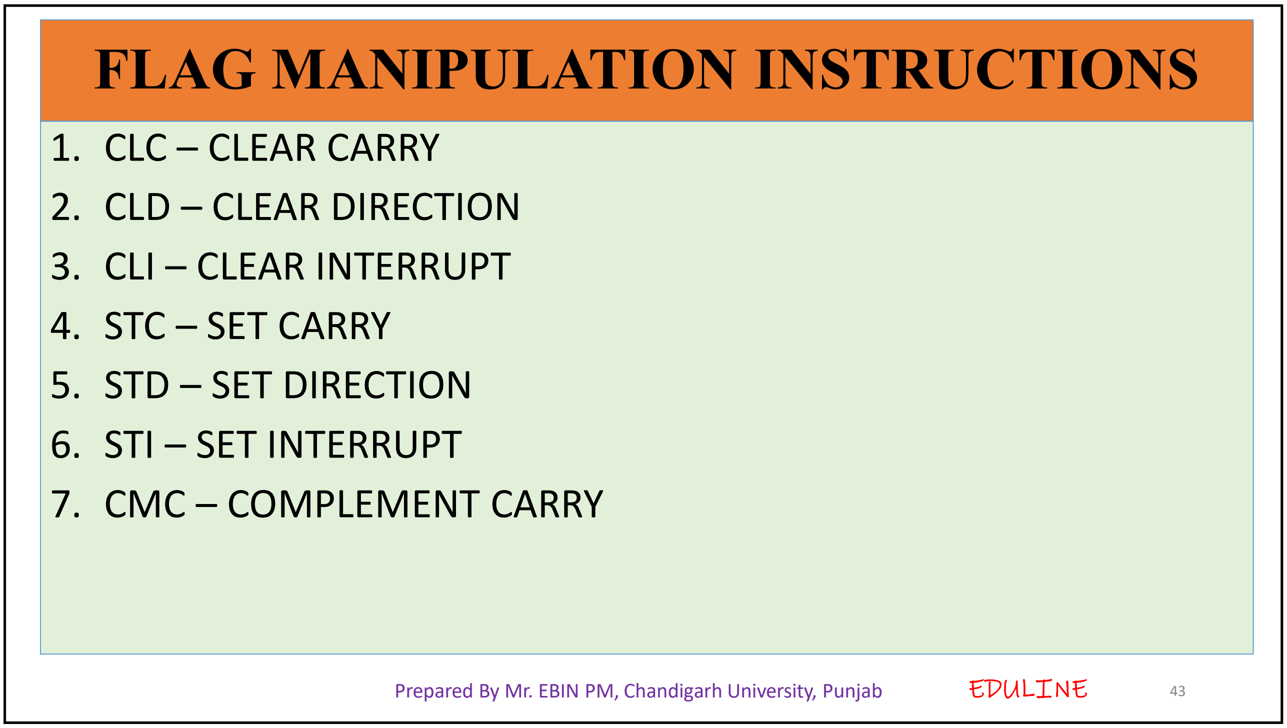


BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CF (arbitrary)
OPERAND	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	0
Count = 1	0	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1

12. RCL (Rotate Left through carry)

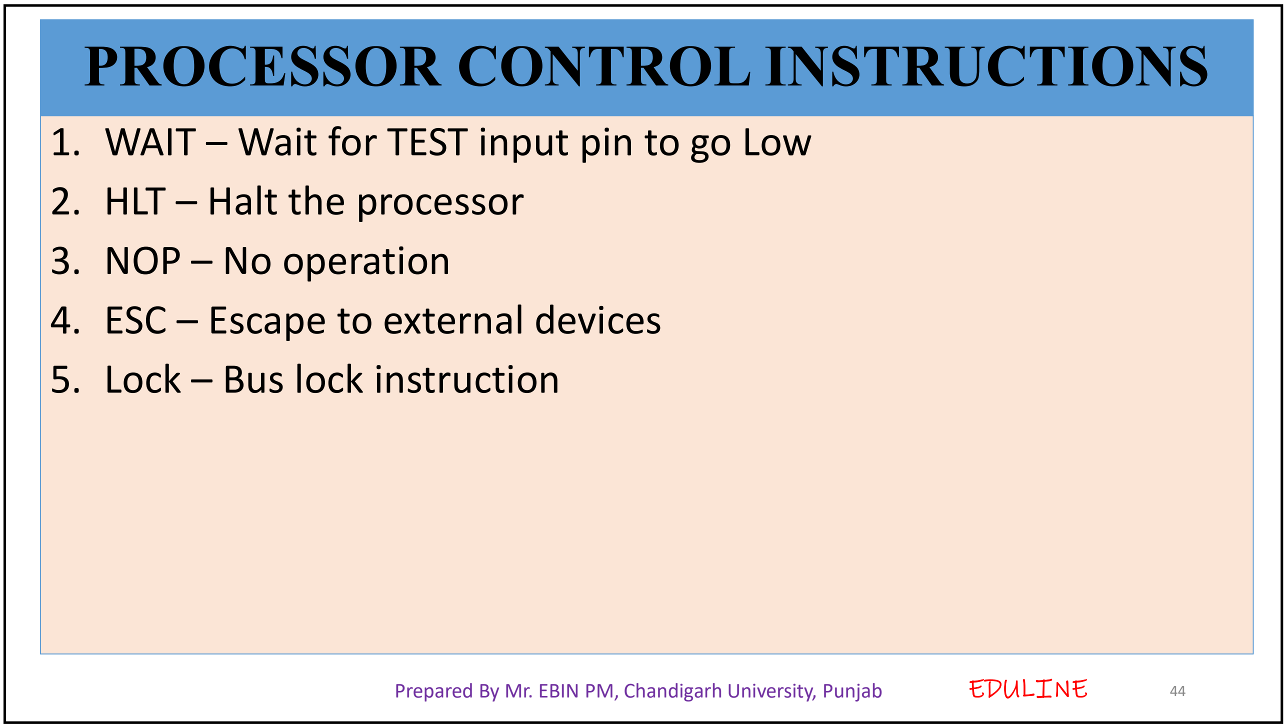


BIT POSITIONS	CF (arbitrary)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERAND	0	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0	1
Count = 1	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0	1	0



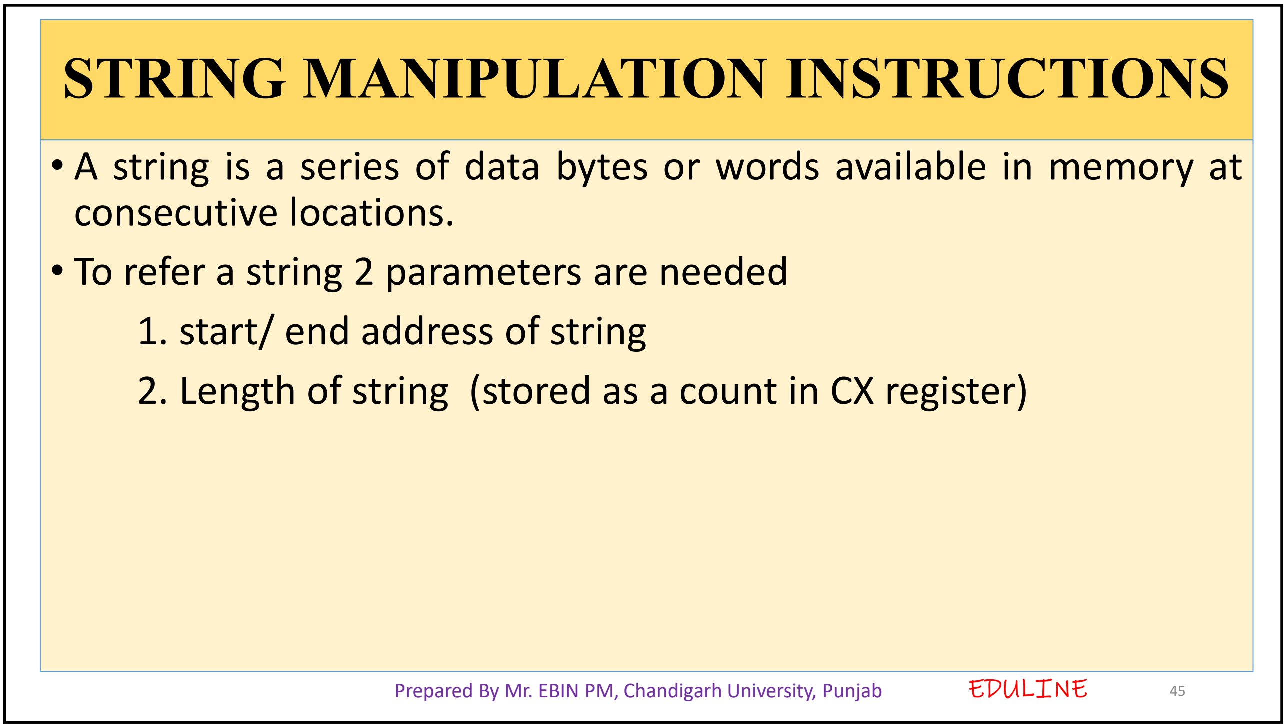
FLAG MANIPULATION INSTRUCTIONS

1. CLC – CLEAR CARRY
2. CLD – CLEAR DIRECTION
3. CLI – CLEAR INTERRUPT
4. STC – SET CARRY
5. STD – SET DIRECTION
6. STI – SET INTERRUPT
7. CMC – COMPLEMENT CARRY



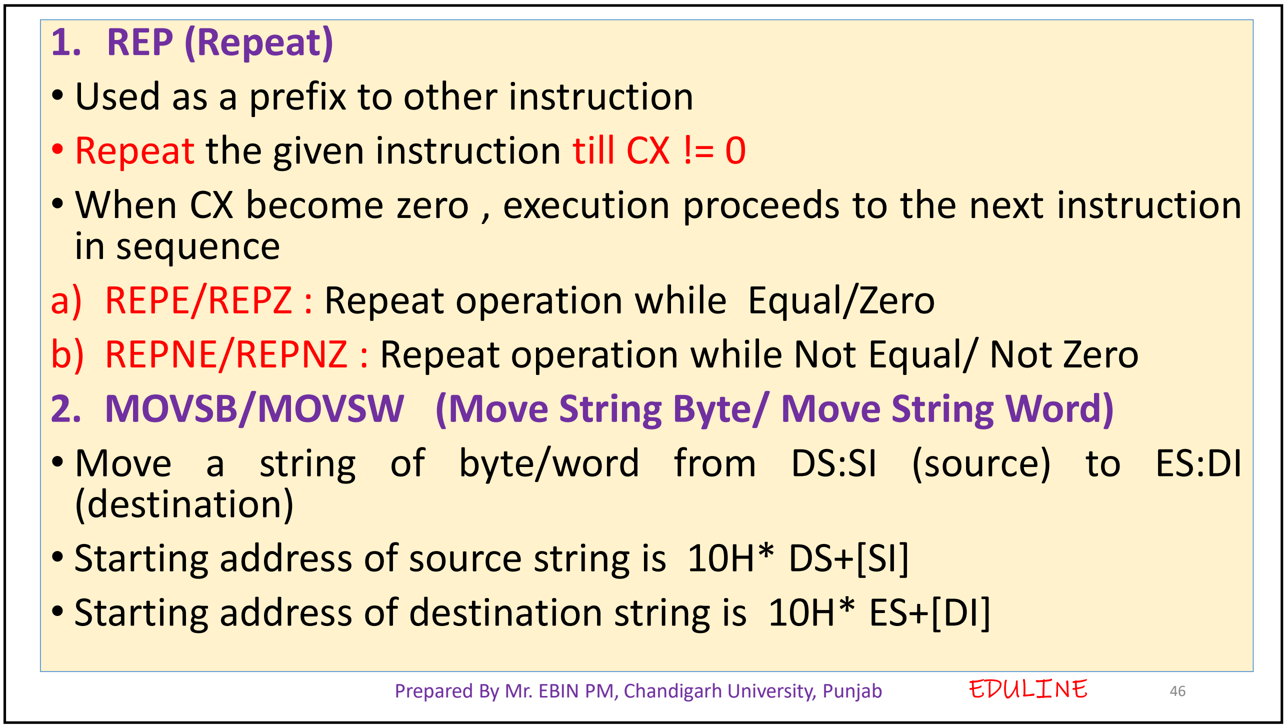
PROCESSOR CONTROL INSTRUCTIONS

1. WAIT – Wait for TEST input pin to go Low
2. HLT – Halt the processor
3. NOP – No operation
4. ESC – Escape to external devices
5. Lock – Bus lock instruction



STRING MANIPULATION INSTRUCTIONS

- A string is a series of data bytes or words available in memory at consecutive locations.
- To refer a string 2 parameters are needed
  1. start/ end address of string
  2. Length of string (stored as a count in CX register)



1. REP (Repeat)

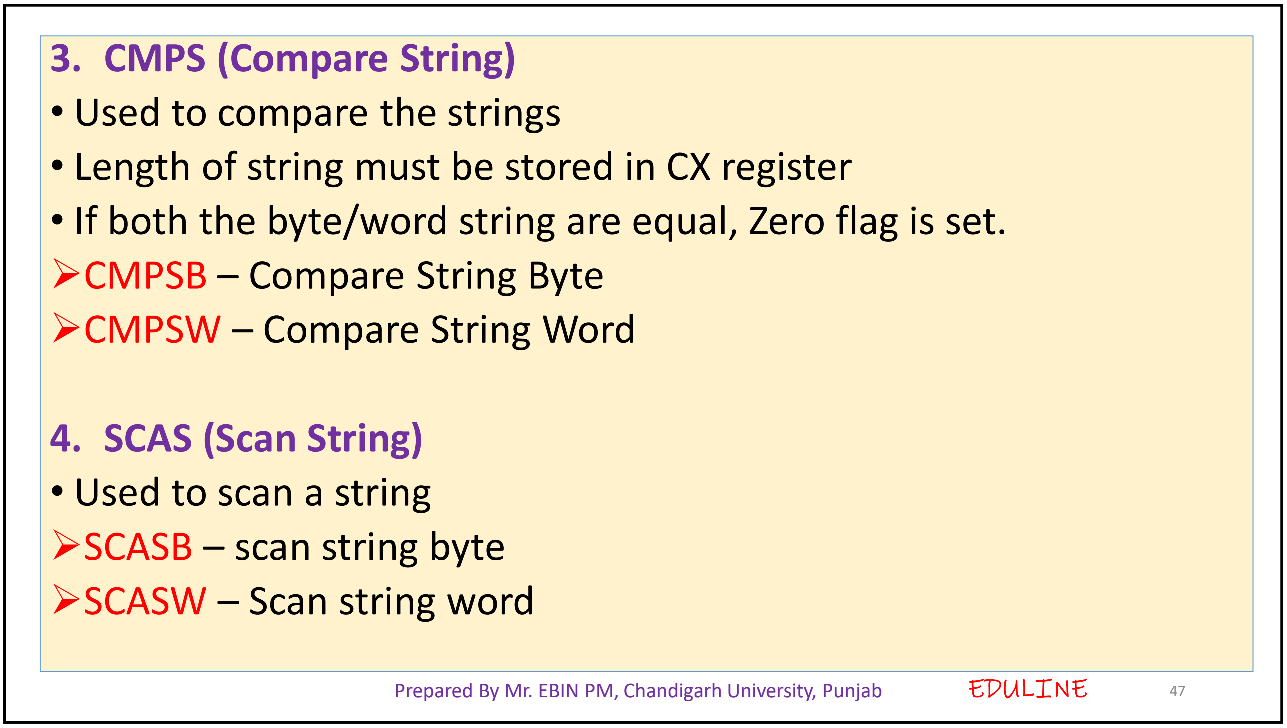
- Used as a prefix to other instruction
- Repeat the given instruction till CX != 0
- When CX become zero , execution proceeds to the next instruction in sequence

a) REPE/REPZ : Repeat operation while Equal/Zero

b) REPNE/REPNZ : Repeat operation while Not Equal/ Not Zero

2. MOVSB/MOVSW (Move String Byte/ Move String Word)

- Move a string of byte/word from DS:SI (source) to ES:DI (destination)
- Starting address of source string is 10H* DS+[SI]
- Starting address of destination string is 10H* ES+[DI]



3. CMPS (Compare String)

- Used to compare the strings
- Length of string must be stored in CX register
- If both the byte/word string are equal, Zero flag is set.

➢CMPSB – Compare String Byte

➢CMPSW – Compare String Word

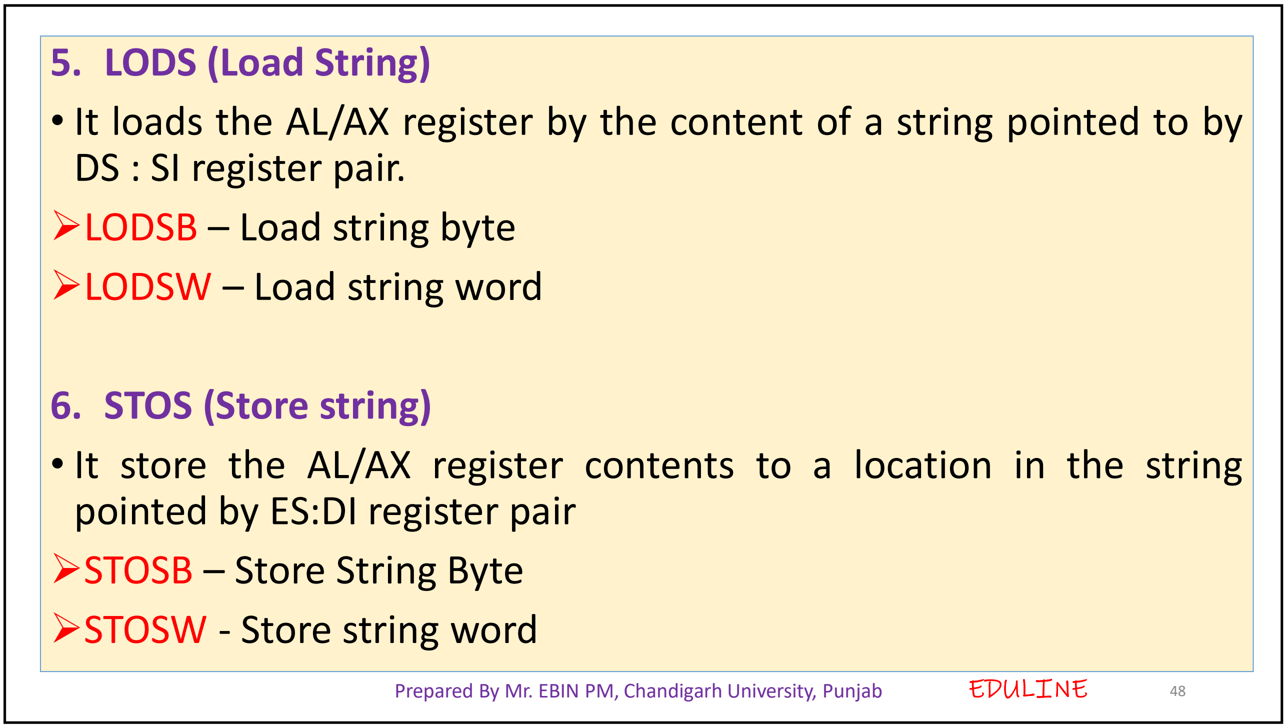
4. SCAS (Scan String)

- Used to scan a string

➢SCASB – scan string byte

➢SCASW – Scan string word

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 47



5. LODS (Load String)

- It loads the AL/AX register by the content of a string pointed to by DS : SI register pair.

➢LODSB – Load string byte

➢LODSW – Load string word

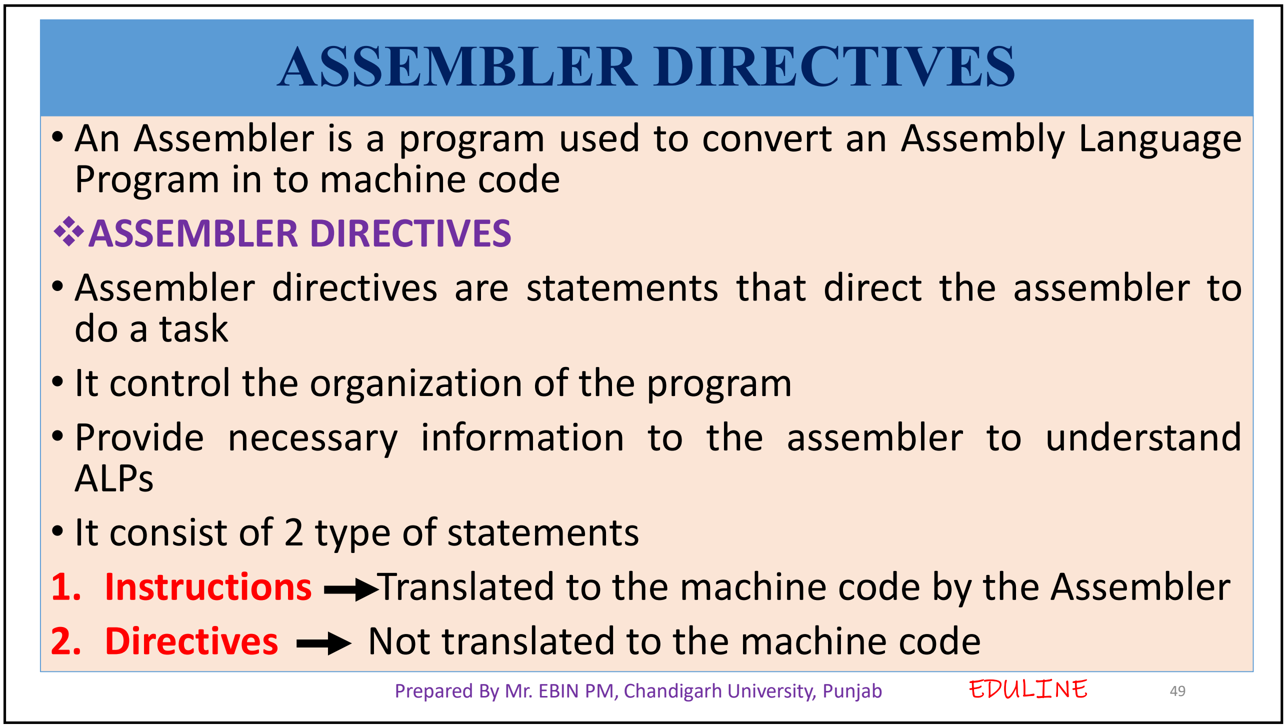
6. STOS (Store string)

- It store the AL/AX register contents to a location in the string pointed by ES:DI register pair

➢STOSB – Store String Byte

➢STOSW - Store string word

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 48



ASSEMBLER DIRECTIVES

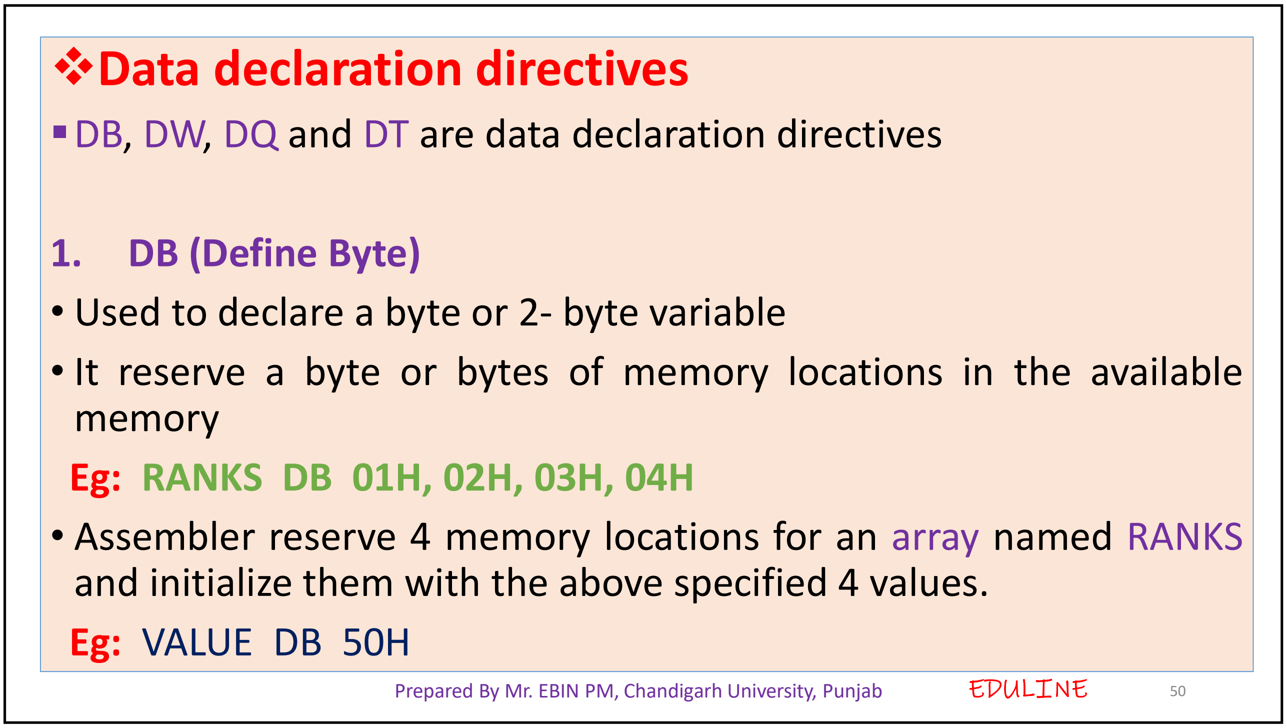
- An Assembler is a program used to convert an Assembly Language Program in to machine code

❖**ASSEMBLER DIRECTIVES**

- Assembler directives are statements that direct the assembler to do a task
- It control the organization of the program
- Provide necessary information to the assembler to understand ALPs
- It consist of 2 type of statements

1. **Instructions** ⟶Translated to the machine code by the Assembler
2. **Directives** ⟶ Not translated to the machine code

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 49



❖Data declaration directives

- DB, DW, DQ and DT are data declaration directives

1. **DB (Define Byte)**

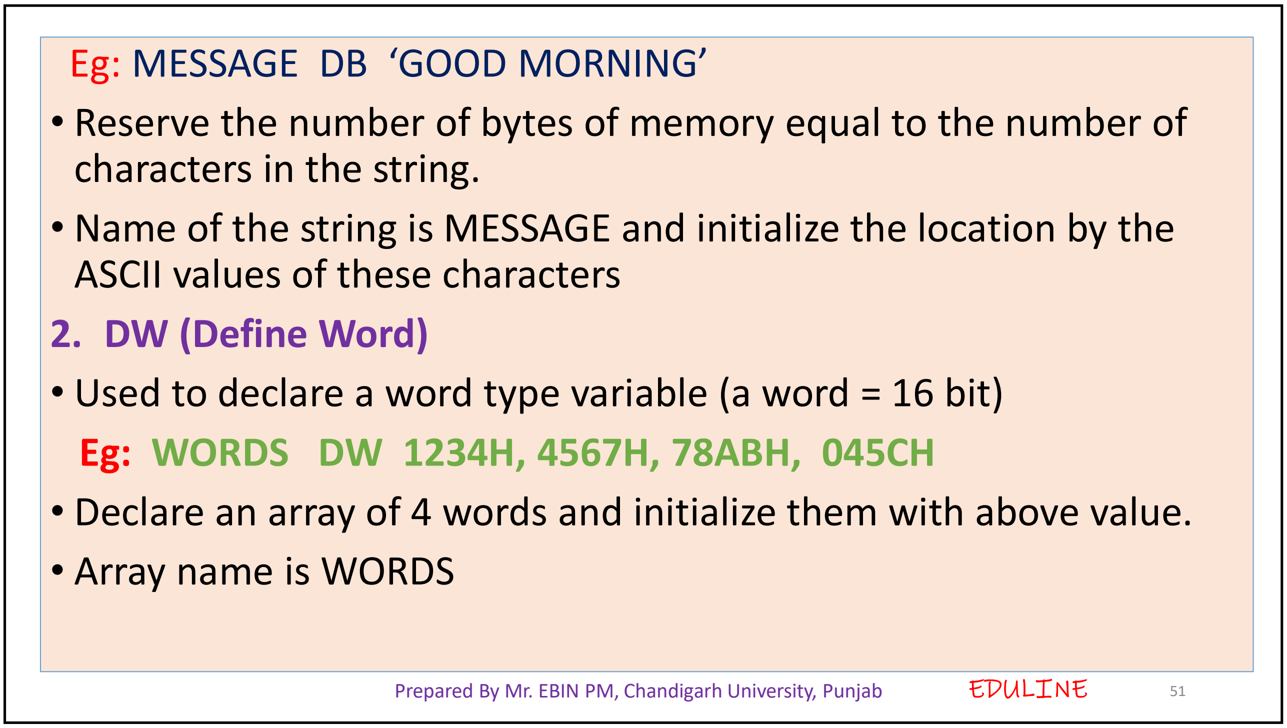
- Used to declare a byte or 2- byte variable
- It reserve a byte or bytes of memory locations in the available memory

Eg: RANKS DB 01H, 02H, 03H, 04H

- Assembler reserve 4 memory locations for an array named RANKS and initialize them with the above specified 4 values.

Eg: VALUE DB 50H

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 50



Eg: MESSAGE DB 'GOOD MORNING'

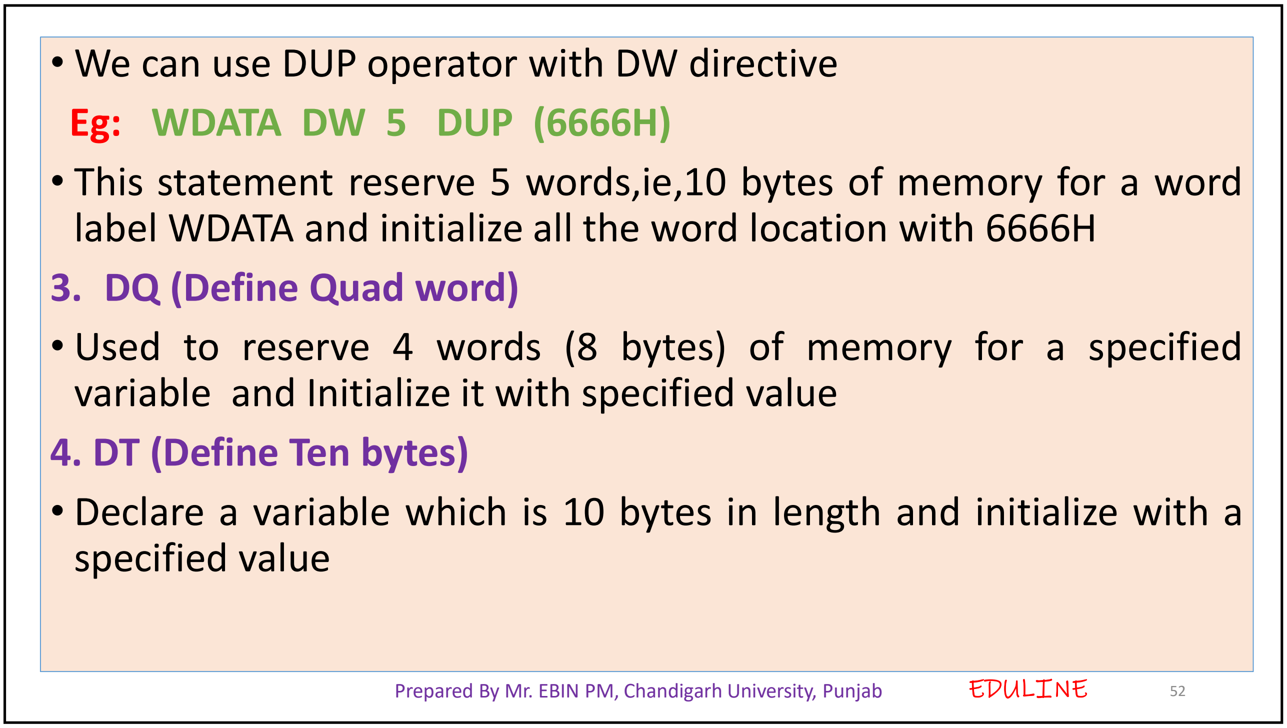
- Reserve the number of bytes of memory equal to the number of characters in the string.
- Name of the string is MESSAGE and initialize the location by the ASCII values of these characters

2. DW (Define Word)

- Used to declare a word type variable (a word = 16 bit)

Eg: WORDS DW 1234H, 4567H, 78ABH, 045CH

- Declare an array of 4 words and initialize them with above value.
- Array name is WORDS



- We can use DUP operator with DW directive

Eg: WDATA DW 5 DUP (6666H)

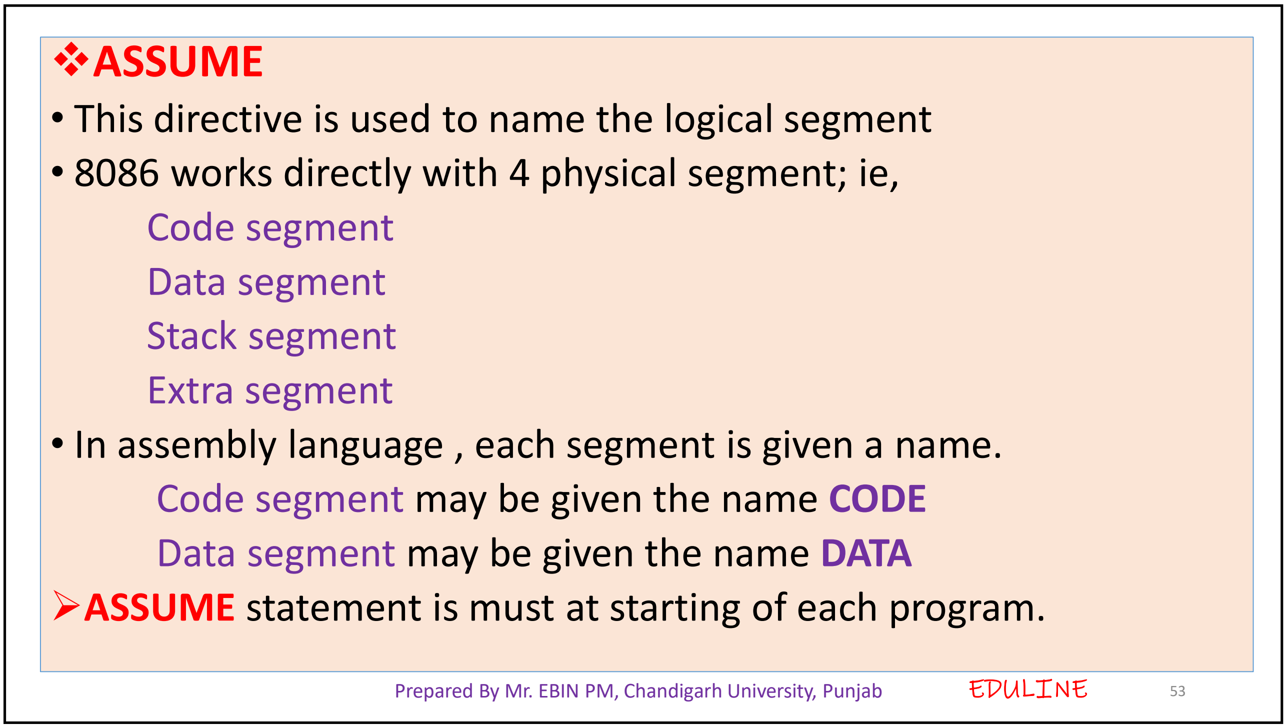
- This statement reserve 5 words,ie,10 bytes of memory for a word label WDATA and initialize all the word location with 6666H

3. DQ (Define Quad word)

- Used to reserve 4 words (8 bytes) of memory for a specified variable and Initialize it with specified value

4. DT (Define Ten bytes)

- Declare a variable which is 10 bytes in length and initialize with a specified value



❖ASSUME

- This directive is used to name the logical segment
- 8086 works directly with 4 physical segment; ie,

  Code segment

  Data segment

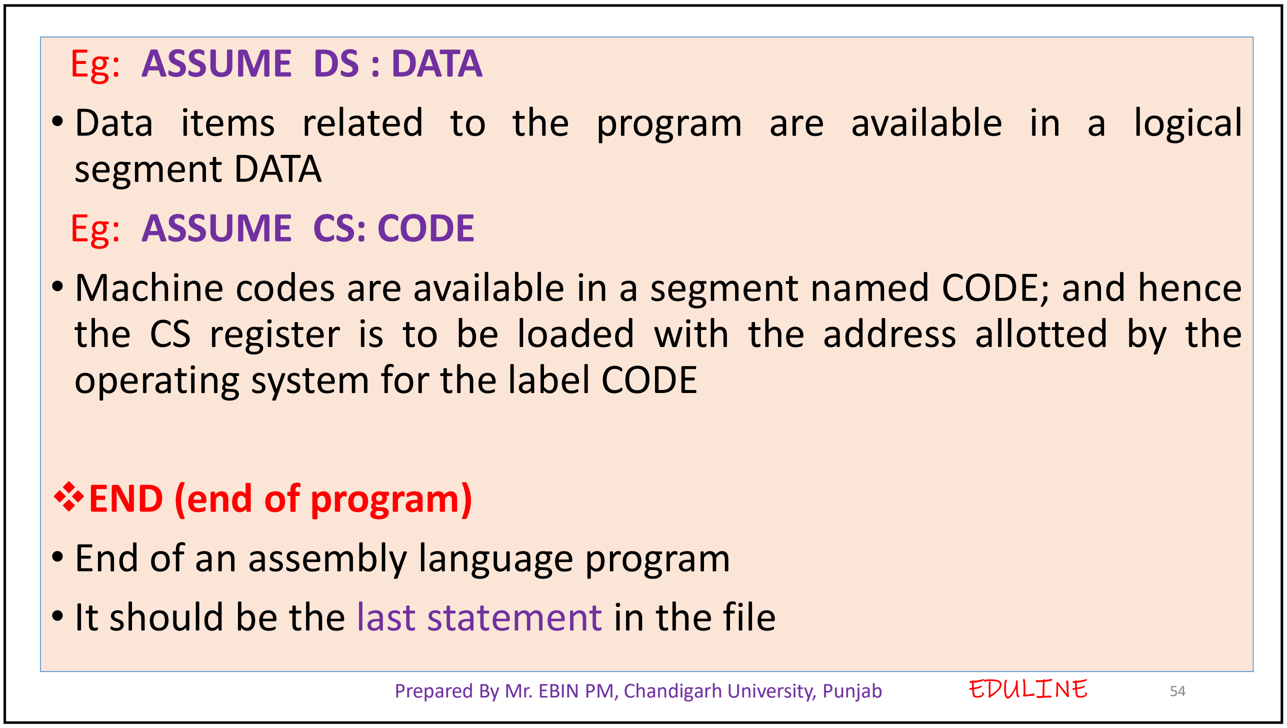
  Stack segment

  Extra segment
- In assembly language , each segment is given a name.

  Code segment may be given the name **CODE**

  Data segment may be given the name **DATA**

➢**ASSUME** statement is must at starting of each program.



Eg: **ASSUME DS : DATA**

- Data items related to the program are available in a logical segment DATA

Eg: **ASSUME CS: CODE**

- Machine codes are available in a segment named CODE; and hence the CS register is to be loaded with the address allotted by the operating system for the label CODE

❖END (end of program)

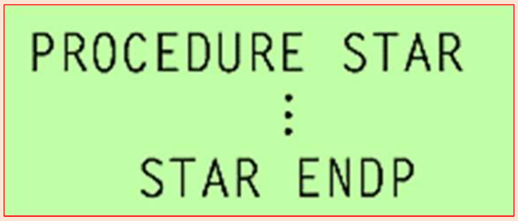
- End of an assembly language program
- It should be the last statement in the file

❖ENDP (end of procedure)

- It indicate end of a procedure(subroutines)
- A procedure has a name or label

➢Syntax: **procedure_name ENDP**

Eg:

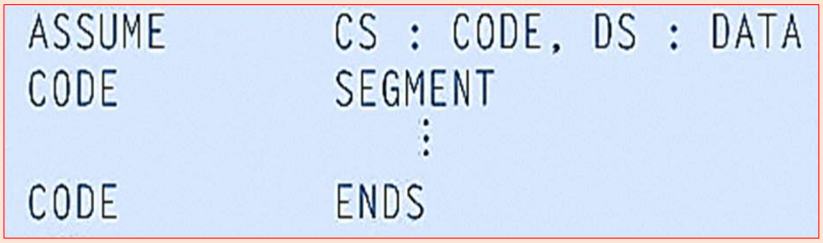


```
PROCEDURE STAR
      :
   STAR ENDP
```

❖ENDS (end of segment)

- End of logical segment

➢Syntax: **segment_name ENDS**



```
ASSUME        CS : CODE, DS : DATA
CODE          SEGMENT
                  :
CODE          ENDS
```

❖SEGMENT

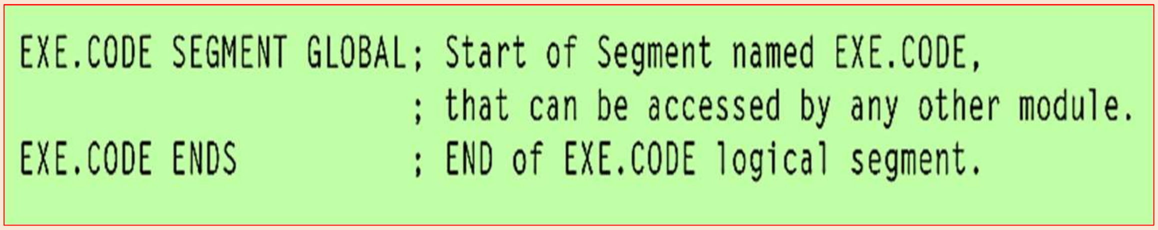
- To indicate the start of a logical segment

Syntax: **segment_name SEGMENT**

- Segment may be assigned a type like **PUBLIC** or **GLOBAL**

➢**PUBLIC** ➔can be used by other modules of the program while linking

➢**GLOBAL**➔can be accessed by any other modules



```
EXE.CODE SEGMENT GLOBAL; Start of Segment named EXE.CODE,
                       ; that can be accessed by any other module.
EXE.CODE ENDS          ; END of EXE.CODE logical segment.
```

❖EQU (Equate)

- Used to give a name to some value or to a symbol.
- Each time the assembler finds the name in the program , it will replace the name with the value or symbol you given to that name.

Eg: **LABEL EQU 0500H -** Assigns the constant 0500H with the label LABEL

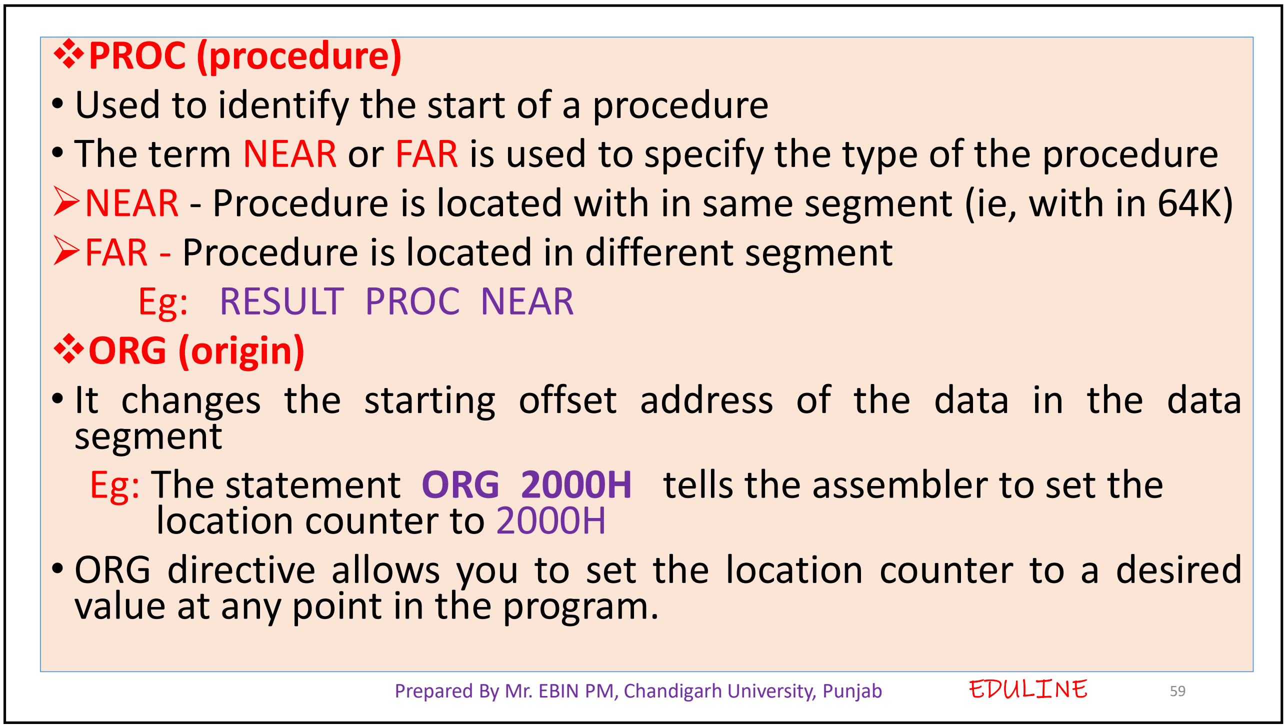
Eg: **FACTOR EQU 03H**

- You has to write this statement at the starting of your program. Later in the program you can use this as follows

ADD AL, FACTOR

- When it codes this instruction, the assembler will code it as

ADD AL, 03H



❖PROC (procedure)

- Used to identify the start of a procedure
- The term NEAR or FAR is used to specify the type of the procedure

➢NEAR - Procedure is located with in same segment (ie, with in 64K)

➢FAR - Procedure is located in different segment

Eg: RESULT PROC NEAR

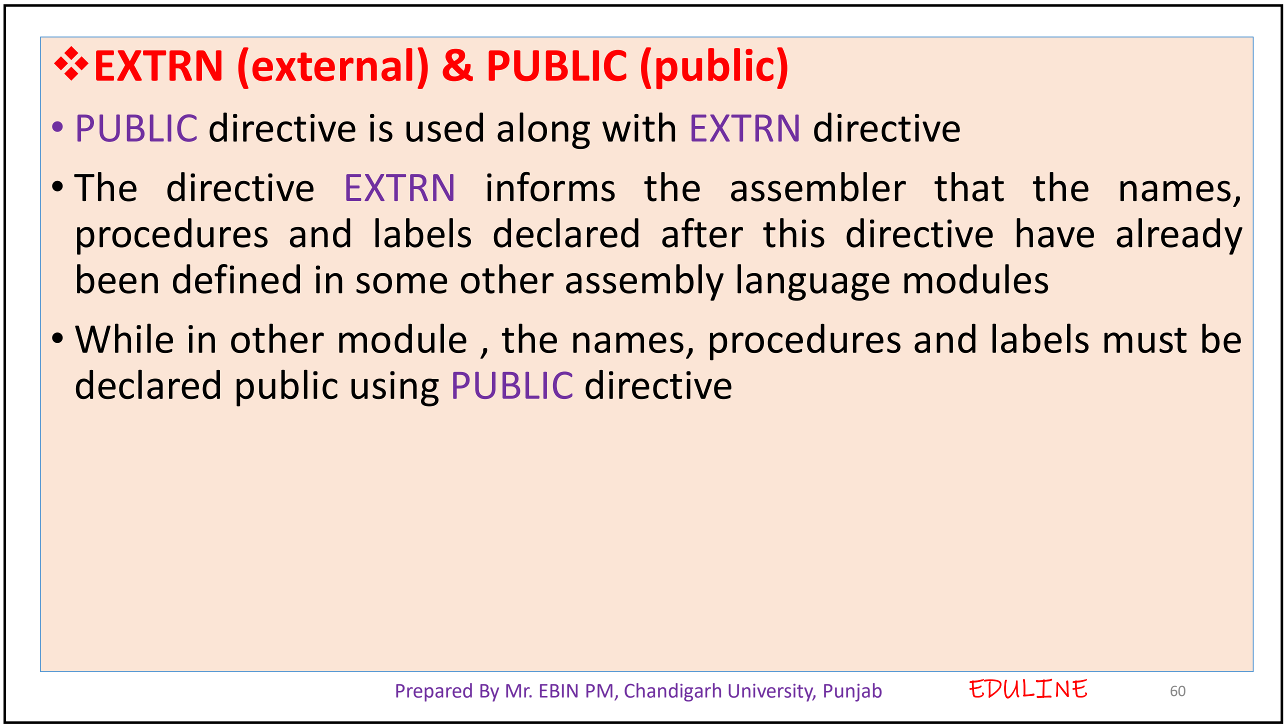
❖ORG (origin)

- It changes the starting offset address of the data in the data segment

Eg: The statement **ORG 2000H** tells the assembler to set the location counter to 2000H

- ORG directive allows you to set the location counter to a desired value at any point in the program.

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 59

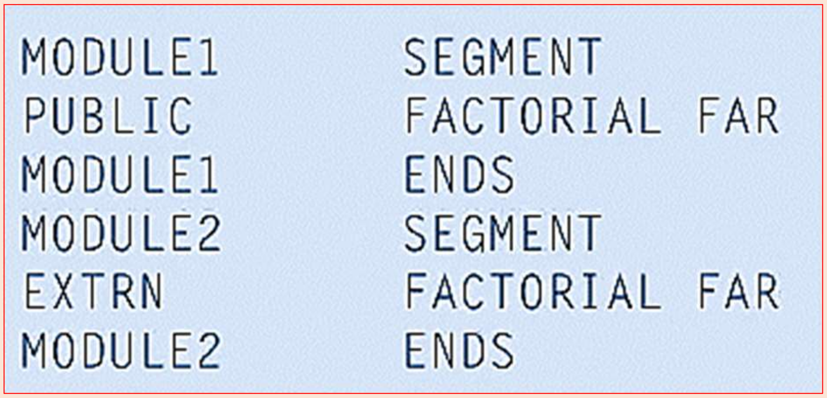


❖EXTRN (external) & PUBLIC (public)

- PUBLIC directive is used along with EXTRN directive
- The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules
- While in other module , the names, procedures and labels must be declared public using PUBLIC directive

Prepared By Mr. EBIN PM, Chandigarh University, Punjab EDULINE 60

Eg: If one wants to call a procedure FACTORIAL appearing in module 1 from module 2 , in module1, it must be declared public using the statement PUBLIC FACTORIAL, and in module2 it must be declared external using the statement EXTRN FACTORIAL



```
MODULE1       SEGMENT
PUBLIC        FACTORIAL  FAR
MODULE1       ENDS
MODULE2       SEGMENT
EXTRN         FACTORIAL  FAR
MODULE2       ENDS
```

❖GROUP (group the related segments)

- This directive form a logical group of segments with similar purpose or type

  Eg: PROGRAM GROUP CODE, DATA, STACK

- Here CODE, DATA and STACK segment must lie with in a 64Kbytes memory segment, that is named as PROGRAM
- For the ASSUME statement , one can use the label PROGRAM rather than CODE, DATA and STACK. ie,

  ASSUME CS: PROGRAM DS: PROGRAM SS: PROGRAM

❖PTR (pointer)

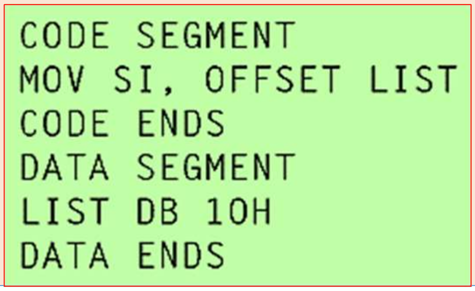
- Used to specify the data type byte or word
- If the prefix is BYTE , then the particular label, variable or memory operand is an 8- bit quantity.
- If word is the prefix, then it is 16-bit quantity

Eg: **MOV AL, BYTE PTR [SI]** Moves content of memory location addressed by SI (8 bit) to AL

Eg: **MOV BX, WORD PTR [2000H]** Moves 16 bit content of memory location 2000H to BX . ie, [2000H] to BL , [2001H] to BH

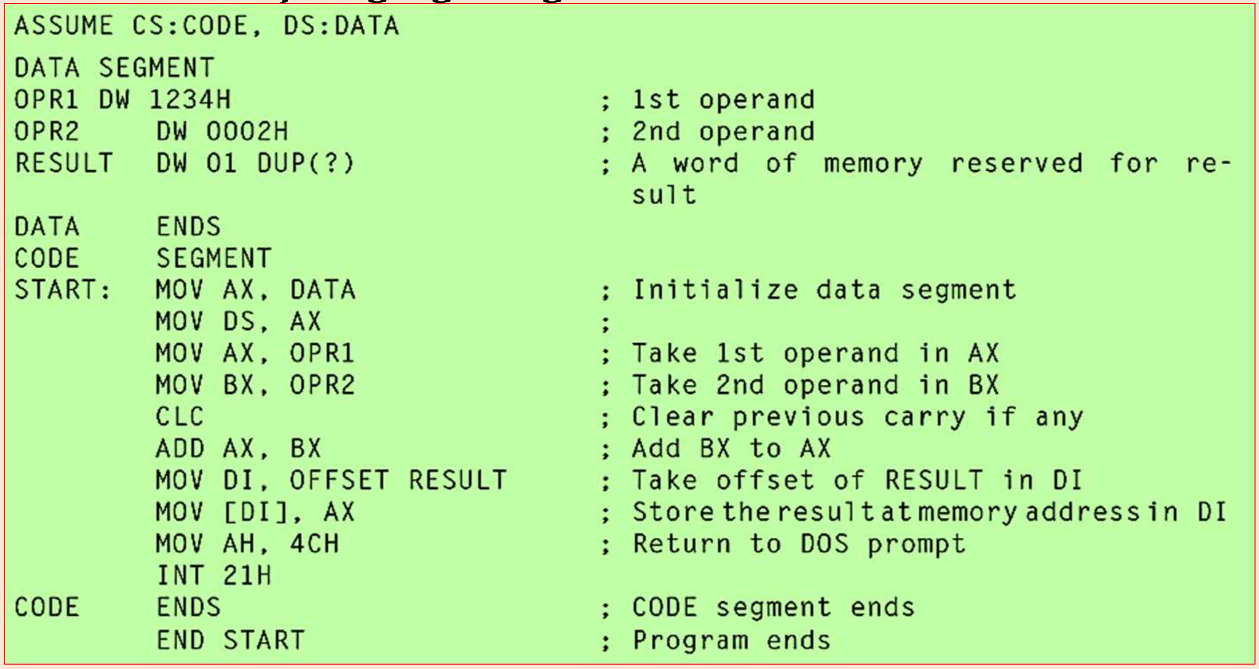
❖OFFSET (Offset of a Label)

- When the assembler comes across the OFFSET operator along with a label, It first compute the 16 – bit displacement of the label and replaces the string ‘ OFFSET LABEL ‘ by the computed displacement
- This operator is used with arrays , strings, labels and procedures to decide their offsets in their default segments.



```
CODE SEGMENT
MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
LIST DB 10H
DATA ENDS
```

Assembly Language Program to add two 16 –bit numbers



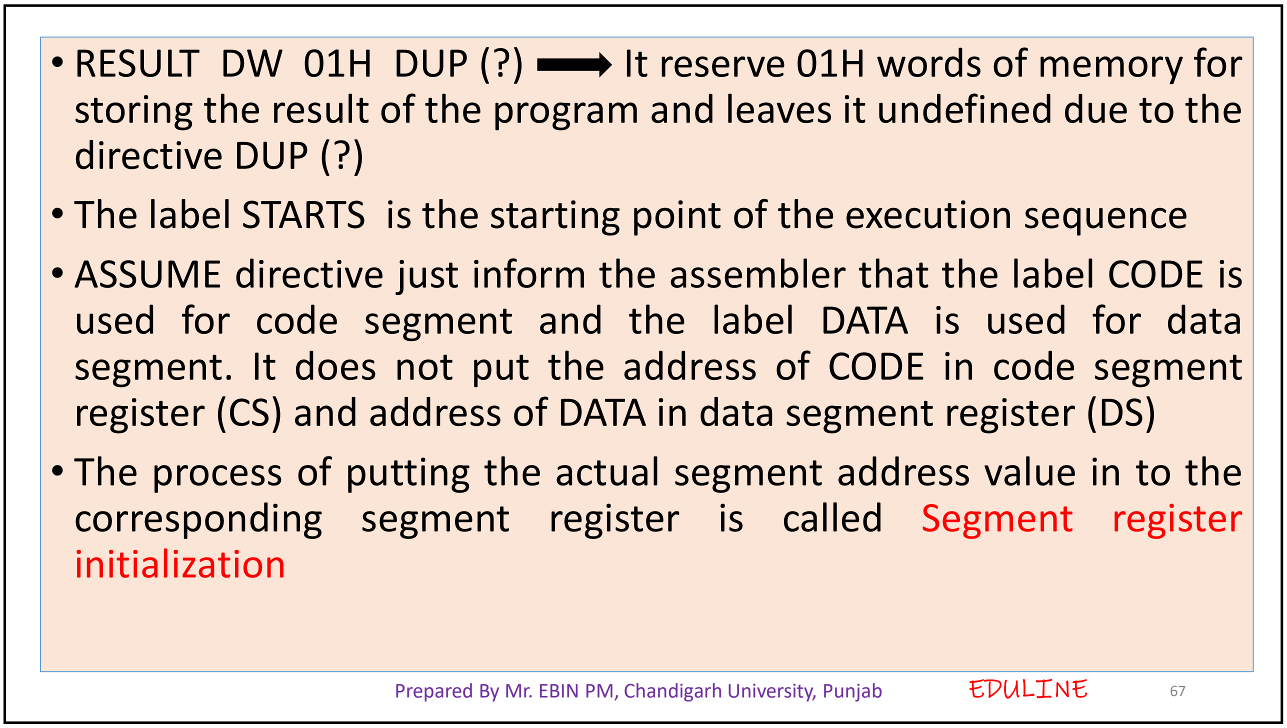
```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
OPR1 DW 1234H                  ; 1st operand
OPR2    DW 0002H               ; 2nd operand
RESULT  DW 01 DUP(?)           ; A word of memory reserved for re-
                                 sult
DATA    ENDS
CODE    SEGMENT
START:  MOV AX, DATA           ; Initialize data segment
        MOV DS, AX             ;
        MOV AX, OPR1           ; Take 1st operand in AX
        MOV BX, OPR2           ; Take 2nd operand in BX
        CLC                    ; Clear previous carry if any
        ADD AX, BX             ; Add BX to AX
        MOV DI, OFFSET RESULT  ; Take offset of RESULT in DI
        MOV [DI], AX           ; Store the result at memory address in DI
        MOV AH, 4CH            ; Return to DOS prompt
        INT 21H
CODE    ENDS                   ; CODE segment ends
        END START              ; Program ends
```

- Some data may be required for the program. So DATA segment is needed
- CODE segment contains actual instruction . It is compulsory
- If stack facility is used we need STACK segment
- In the first line of program, ASSUME directive declares that the label CODE refers to the code segment & the label DATA refers to the data segment

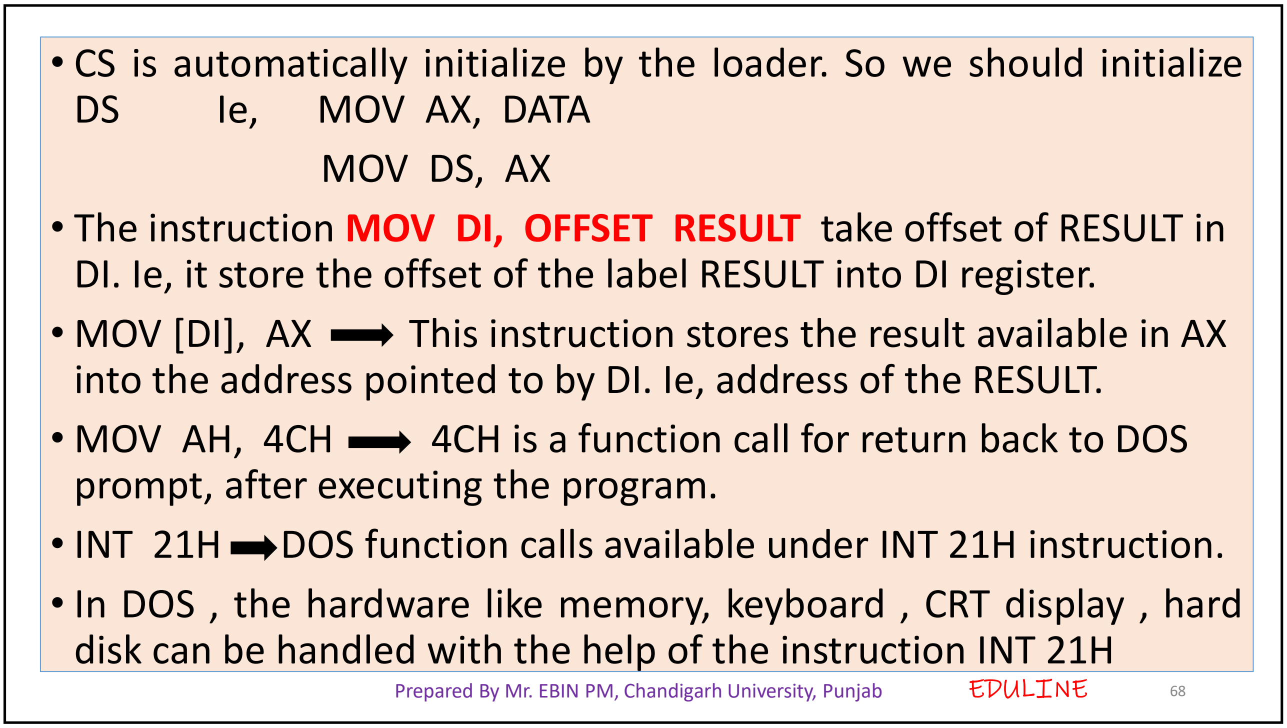
➢CODE = Logical name of code segment

➢DATA = Logical name of data segment

- OPR1 is first operand & OPR2 is second operand



- RESULT DW 01H DUP (?) ➡ It reserve 01H words of memory for storing the result of the program and leaves it undefined due to the directive DUP (?)
- The label STARTS is the starting point of the execution sequence
- ASSUME directive just inform the assembler that the label CODE is used for code segment and the label DATA is used for data segment. It does not put the address of CODE in code segment register (CS) and address of DATA in data segment register (DS)
- The process of putting the actual segment address value in to the corresponding segment register is called Segment register initialization



- CS is automatically initialize by the loader. So we should initialize DS Ie, MOV AX, DATA

  MOV DS, AX
- The instruction **MOV DI, OFFSET RESULT** take offset of RESULT in DI. Ie, it store the offset of the label RESULT into DI register.
- MOV [DI], AX ➡ This instruction stores the result available in AX into the address pointed to by DI. Ie, address of the RESULT.
- MOV AH, 4CH ➡ 4CH is a function call for return back to DOS prompt, after executing the program.
- INT 21H ➡DOS function calls available under INT 21H instruction.
- In DOS , the hardware like memory, keyboard , CRT display , hard disk can be handled with the help of the instruction INT 21H