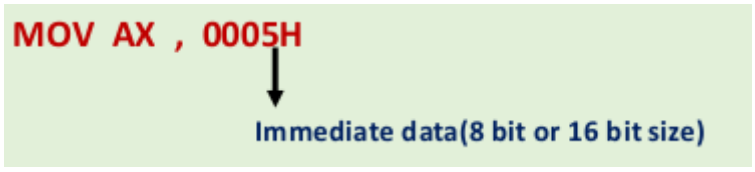
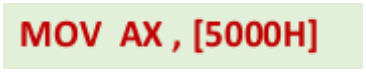
Microprocessors Module 2 Important topics

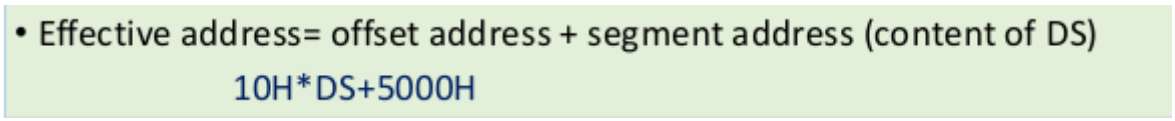
Addressing modes and Calculation of effective address

- The different ways in which a source operand is denoted in an instruction is known as addressing modes.
- Types of instruction
  - Sequential control transfer instructions
    - Transfer control to next instruction immediately after it
    - Example Arithmetic , logical
  - Control transfer instructions
    - Transfer control to some predefined address/ address specified in the instruction
    - Example CALL, JUMP, RET

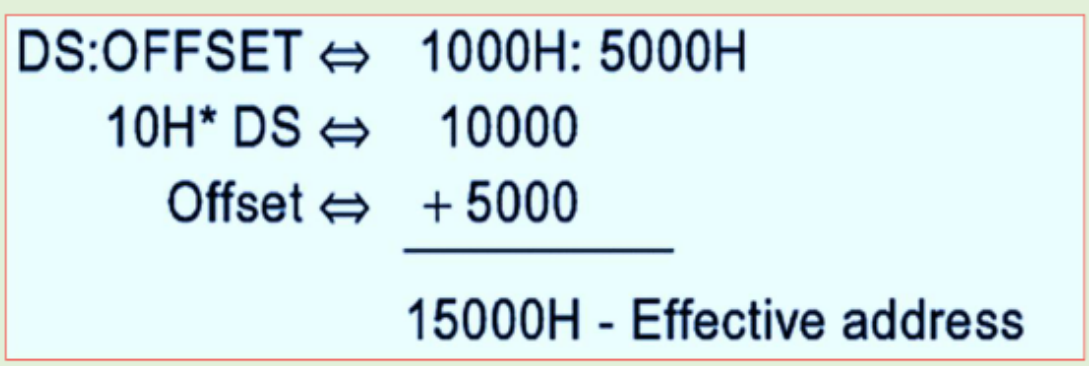
1. Addressing modes for sequential control transfer

- Immediate
  - Immediate data is part of instruction
  - immediate mode gives the computer the data right away, like handing over a number
  - Immediate data can be 8 bit or 16 bit
  - 

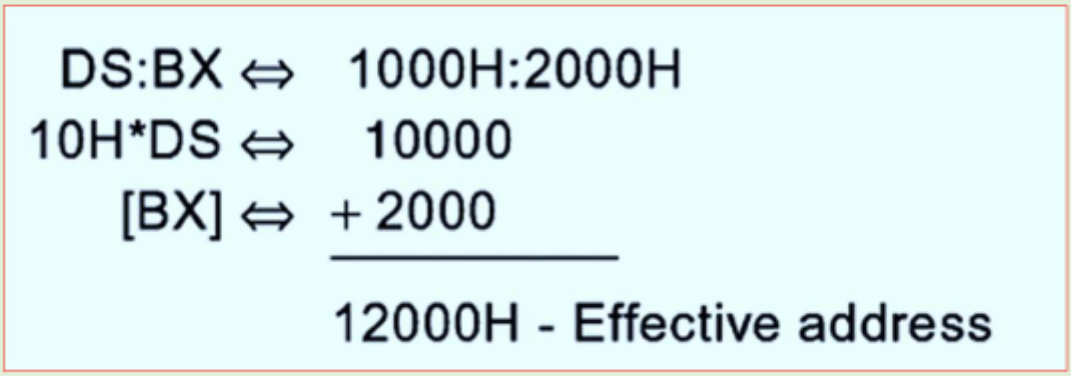
- Direct
  - Memory address directly specified in the instruction
  - direct mode tells the computer where to find the data in its memory, similar to giving directions to a specific location.
  - 

  - Here the location is 5000H, The data inside 5000H Location is to be transferred to AX
  - 

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

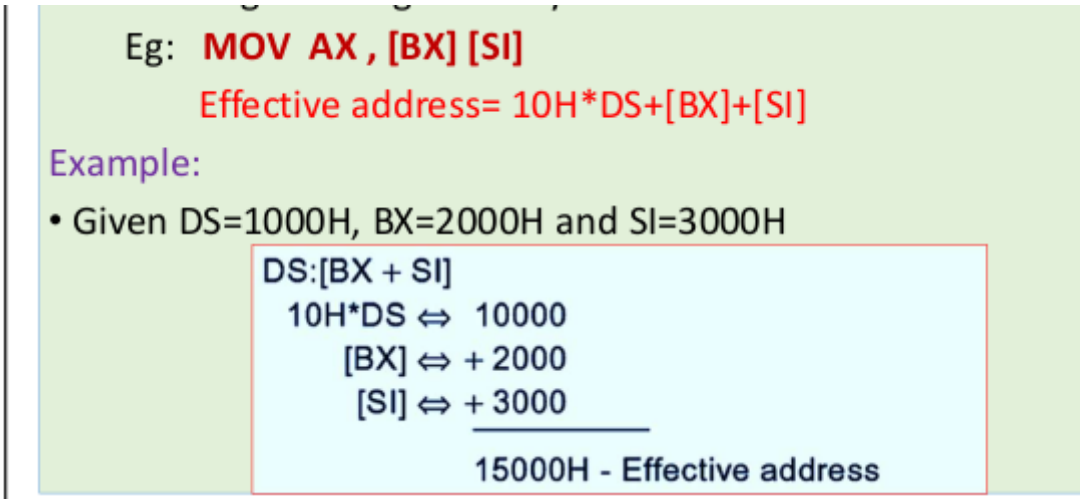


- Register
  - Data is stored in register. All the registers except IP can be used
  - Eg: **MOV BX, AX**
- Register Indirect
  - this instruction is saying, "Go to the memory location specified by the value in `BX`, get whatever is there, and put it into `AX`."
  - **MOV AX, [BX]**
  - **Effective address= 10H * DS+[BX]**
  - **Example :**
    - Given DS=1000H and BX=2000H

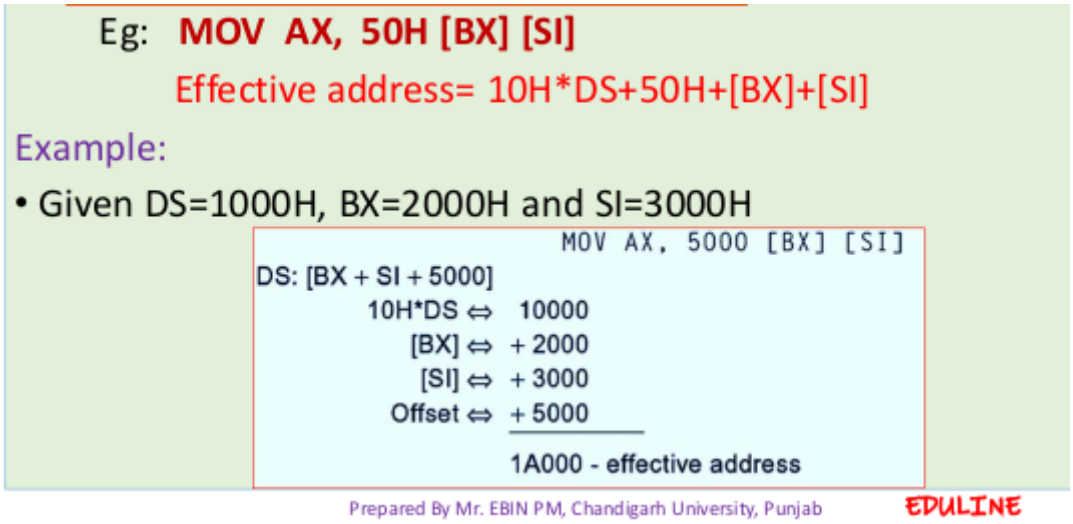


- Indexed
  - offset of the operand is stored in one of the index registers.
  - For SI (Source index)
    - Default segment is DS
  - For DI (Destination index)
    - Default segment is ES
  - **MOV AX, [SI]**
    Effective address= 10H*DS+[SI]
- Register Relative
  - Data is available by adding the 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
- Based Indexed
  - Effective address is sum of base register and Index register

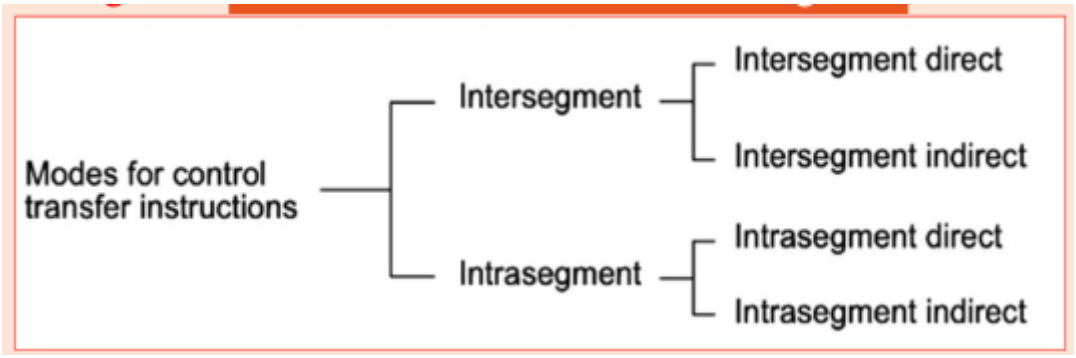


- Relative Based Indexed
  - effective address is formed by adding displacement with the sum of content of any of base registers (BX or BP) and any one of the index registers



2. Addressing modes for control transfer instruction

- Its classified into 2 types
  - Intersegment
  - Destination location is in different segment.
  - Intrasegment
  - Destination location is in same segment.



Intersegment Direct

- Destination location is in different segment.
- Provides branching from one code segment to another code segment
- Example
  - ```
    JPM 5000H : 2000H;
    Jump to effective address 2000H in segment 5000H.
    ```

Intersegment Indirect

- 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

- Displacement in computed using content of IP

Intrasegment InDirect

- Destination location is passed to the instruction indirectly.
  - Example

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

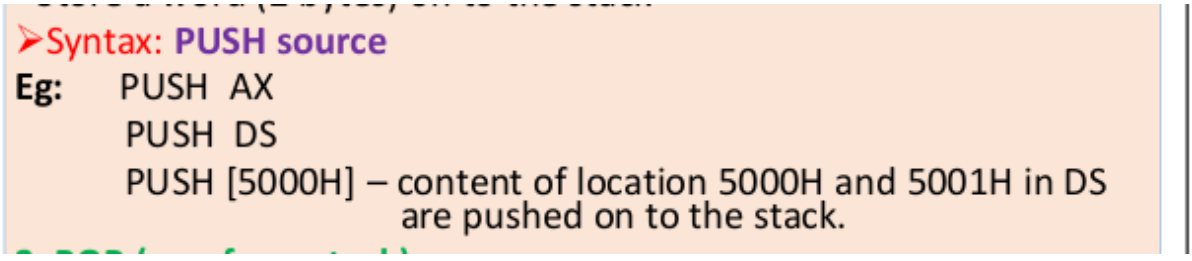
Instruction set

- Data copy / Transfer instruction
  - Transfer data from source operand to destination operand
  - Example
    - Storage, Move, load etc
- Arithmetic and logical instruction
- Branch instruction
    ```

    - Transfer control of execution to specified address
        - Jump, interrupt, call
- Loop Instructions
    - Implement loop structure
        - LOOP, LOOPNZ, LOOPZ instruction
- Machine control instruction
    - Control machine status
        - Example: WAIT, HLT
- Flag Manipulation
    - Affect flag register
        - **CLD (Clear Direction Flag):**
        - **STD (Set Direction Flag):**
        - **CLI (Clear Interrupt Flag):**
        - **STI (Set Interrupt Flag):**
- Shift and rotate instruction
    - For bitwise shifting and rotation
- String instruction
    - String manipulation operation
        - Load, Move, Scan, Compare, Store

Data Copy/Transfer Instruction

- MOV
    - Transfer data from one register/memory to another register/memory
    - 

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

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

- ➢Syntax: **POP destination**
  Eg: POP AX
  POP DS
- XCHG
  - 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)
- 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
- OUT(output to 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
- XLAT (translate)
  - Finding the code in code conversion problem
  - Eg: translate the code of the key pressed to 7-segmented code
- LEA (Load Effective Address)
  - Load offset of an operand in specified register
- PUSHF (Push Flag to Stack)
- POPF(Pop Flag from Stack)
- LDS/LES (Load Pointer to DS/ES)

Arithmetic Instructions

- ADD
- ADC (Add with carry)
- INC (Incrment)
  - Eg INC AX
- DEC (Decrement)

    - Eg DEC AX
- SUB
- SBB (Sub with borrow)
- CMP (Compare)
    - Compare the source operand with destination operand
    - Eg CMP BX, CX
- MUL
- IMUL (Signed multiplication)
- DIV
- IDIV (Signed division)

Logical Instructions

- AND
- OR
- NOT
- XOR
    - When two inputs are different , XOR gives high output
    - When two inputs are same , XOR gives low output
- TEST
    - Performs bit by bit logical AND operation
- SHL/SAL (Shift Logical/ Arithmetic Left)
- SHR (Shift Logical Right)
- SAR (Shift Arithmetic Right)
- ROR (Rotate Right without Carry)
- ROL(Rotate Left without carry)
- RCR (Rotate Left through Carry)

Flag manipulation instruction

- CLC - Clear Carry
- CLD - Clear Direction
- CLI - Clear Interrupt
- STC - Set Carry
- STD - Set Direction
- STI - Set Interrupt
- CMC - Complement Carry

Processor Control Instructions

- WAIT - Wait for TEST Input pin to go low
- HLT - Halt the processor
- NOP - No operation
- ESC - Escape to external devices
- Lock - Bus lock instruction

String Manipulation instruction

- To refer a string 2 parameters are needed
    - Start/End Address of string
    - Length of string
- REP (Repeat)
    - Repeat the Given Instruction till CX!=0
        - Here CX means Counter
- MOVSB/MOVSW (Move String byte/ Move string word)
    - Move a string of byte/word from source to destination
- 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.
- SCAS (Scan String)
    - Used to scan a string
- LODS (Load String)
    - It loads the AL/AX register by the content of a string pointed to by DS : SI register pair.
- STOS (Store string)
    - It store the AL/AX register contents to a location in the string pointed by ES:DI register pair

Assembler directives

- Assembler is a program used to convert assembly language into machine code
- Assembler directives are statements that direct the assembler to do a task
    - It control the organization of the program
    - Provides necessary information to assember to understand assembly lanugage programs
- There are 2 type of statements

- Instructions
    - Translated to the machine code by the Assembler
- Directives
    - Not translated to the machine code

Data declaration directives

- The data declaration directives are
- DB(Define Byte)
    - Used to declare a byte or 2 byte
- DW (Define Word)
    - Used to declare a word type variable
- DQ (Define Quad Word)
    - Used to reserve 4 words (8 bytes)
- DT (Define Ten bytes)

Assume

- Used to name the logical segment
- In assembly language , each segment is given a name
    - Code segment may be given the name CODE
        - ASSUME CS : CODE
    - Data segment may be given the name DATA
        - ASSUME DS : DATA

END (End of program)

ENDP (End of procedure)

ENDS (End of segment)

Segment

- Indicates the start of a logical statement

EQU (Equate)

Used to give a name to some value or to a symbol.

PROC (procedure)

Used to identify the start of a procedure

ORG (origin)

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

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

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

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.

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

Assembly Language Program to add two 16 bit numbers

```
DATA SEGMENT
    N1 DW 1731H     ; Define a 16-bit data word N1 with initial value 1731H
    N2 DW 9212H     ; Define a 16-bit data word N2 with initial value 9212H
    N3 DW ?         ; Define a 16-bit data word N3 with an uninitialized value
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE;DS:DATA
    ; Set code and data segment registers

START:
    MOV AX,DATA     ; Load the address of the data segment into AX
    MOV DS,AX       ; Set the data segment register to the loaded address
    XOR AX,AX       ; Clear AX register (AX = 0)
    MOV BX,AX       ; Clear BX register (BX = 0)

    MOV AX,N1       ; Load the value of N1 into AX
    ADD AX,N2       ; Add the value of N2 to AX
    MOV N3,AX       ; Store the result in N3

    JNC STOP        ; Jump to STOP if there is no carry after the addition
    INC BX          ; Increment BX if there is a carry

STOP:
    MOV CX,AX       ; Move the result of the addition (AX) into CX
    MOV AH,4CH      ; Set the function code for program termination
    INT 21H         ; Invoke interrupt 21H to terminate the program

CODE ENDS
END START          ; End of the program with START as the entry point
```