Microprocessors & Microcontrollers

Second Year

Electrical Engineering Department

College of Engineering

Basrah University

2024-2025

2: Instruction Set and Assembly Language Programming Of 8086

Instructions Set of 8086

- Program is a set of instructions written to solve a problem. Instructions are the directions which a microprocessor follows to execute a task or part of a task. Broadly computer language can be divided into two parts as high-level language and low- level language. Low-level languages are machine specific. Low-level language is further divided into machine language and assembly language.
- Machine language is the only language which a machine can understand. Instructions in this language are written in binary bits as a specific bit pattern. The computer interprets this bit pattern as an instruction to perform a particular task. The entire program is a sequence of the binary numbers. This is a machine-friendly language but not user friendly. Debugging is another problem associated with machine language.
- To overcome these problems, programmers develop another way in which instructions are written in English alphabets. This new language is known as Assembly language. As microprocessor can only understand the machine language so assembly language are translated into machine language either manually or by a program known as assembler.

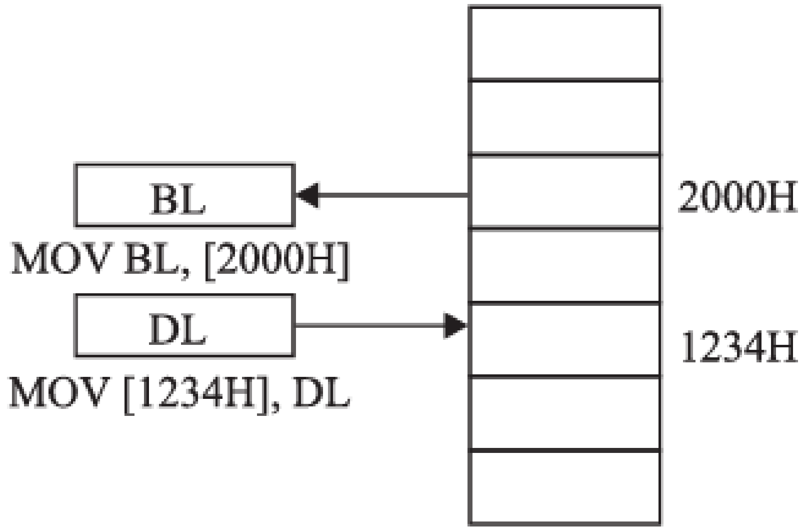
Addressing Modes Of 8086

- The addressing modes are the ways of specifying an operand in an instruction. In 8086 the addressing modes are broadly categorized into two groups, i.e. data addressing modes and address addressing modes. Data addressing modes are for defining a data operand in the instruction whereas address addressing modes are the ways of specifying a branch address in control transfer instructions.
- The 8086 microprocessor introduces many new techniques to access the memory by introduction of many more types of addressing modes. These addressing modes provide flexibility to the processor to access memory, which in turn allows the user to access variables, arrays, records, pointers, and other complex data types in a more flexible manner.
- In immediate addressing mode the operands are specified within the instruction itself. The immediate operand can only be the source operand. For example,
  - MOV AX, 2500H
  - Here the immediate data is 2500H.

- Most 8086 instructions can operate on the 8086's general purpose register set. The content of a register can be accessed by specifying the name of the register as an operand to the instruction. For example, the following MOV instruction of 8086 copies the data from the source operand to the destination operand.
  - MOV AX, BX
  - MOV DL, AL
  - MOV SI, DX

- The 8- and l6-bit registers are the valid operands for this instruction. The only restriction is that both operands must be of the same size.
- The registers are the best place to keep often used variables. Instructions using the registers are shorter and faster than those that access memory.
- Segment registers can never be used as data registers to hold arbitrary values. They should only contain segment addresses.

- The l6-bit memory address is always written inside the square brackets. For example, the instruction MOV BL, [2000H], transfers the content of the memory location 2000H in the BL register. Similarly, the instruction MOV [l2S4H], DL transfers the content of the DL register in the memory location specified by l2S4H. Figure 4.l shows the direct addressing mode.



In this addressing mode, the memory address is specified by some pointer, index or base registers. These registers are written inside the square brackets. There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

```
MOV  DX, [BX]
MOV  DX, [BP]
MOV  DX, [SI]
MOV  DX, [DI]
```

Data Transfer Instructions

- The Data Transfer Instructions are used for transferring data from source location to destination location. The Arithmetic Instructions are used to perform arithmetic operations like addition, subtraction, multiplication and division. Logical Instructions perform the logical operations like AND, OR, XOR operations. Shift and Rotate Instructions are used to perform the logical and arithmetic shift operations and left and right shifting. String Instructions performs the string related operations.
- ***MOV destination, source***
- The MOV Instruction copies the second operand (source) to the first operand (destination) without modifying the contents of the source. In true sense these are not the data transfer instructions but data copy instruction because the source is not modified.
- The source operand can be an immediate value, general-purpose register or memory location. The destination register can be a general-purpose register, or memory location. Both operands must be the same size, which can be a byte or a word.

- The following types of operands are supported:

  MOV REG, memory

  MOV memory, REG

  MOV REG, REG

  MOV memory, immediate

  MOV REG, immediate

- Here the register may be any of the general purposes registers, i.e. AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, and SP. The memory may be specified by any of the memory-related addressing modes. Immediate data can only be specified at the source location.
- A data cannot be transferred from a memory to another memory, from memory to an IO, from an IO to another IO and from IO to memory. IO can communicate with Accumulator only.
- The MOV instruction cannot set the value of the CS and IP registers. Also, it cannot copy value of one segment register to another segment register. For example, if we want to initialize the Data Segment by a memory location 02500H, then first we have to load the value 2500H into AX register and then transferring the contents of AX to DS register with the help of the following instructions.
- Example

```
MOV  AX, 2500H,
MOV  DS, AX
```

Arithmetic Instructions

- The 8086 provides many arithmetic operations: addition, subtraction, negation, multiplication, division/modulo (remainder), and comparing two values.
- **ADD operation**: This instruction add a data from source operand to a data from destination and save the result in the destination operand. The source and destination must be of the same type, means they must be a byte type or a word type.
- **Example:** The following example demonstrates how the ADD instruction can be used to perform the operation (1 + 5 + 7):

```
MOV AX, 1
ADD AX, 5
ADD AX, 7
```

- **SUB operation**: This instruction subtracts the source from the destination along with the value of the carry flag. The result is stored in the destination. This instruction is used to subtract the data which are of large in size, i.e. double word type.

- **Example:** The following example demonstrates how the SUB instruction can be used to perform the operation (5678H – 3421H + 10H):

```
MOV AX, 5678H
SUB  AX, 3421H
ADD AX, 10H
```

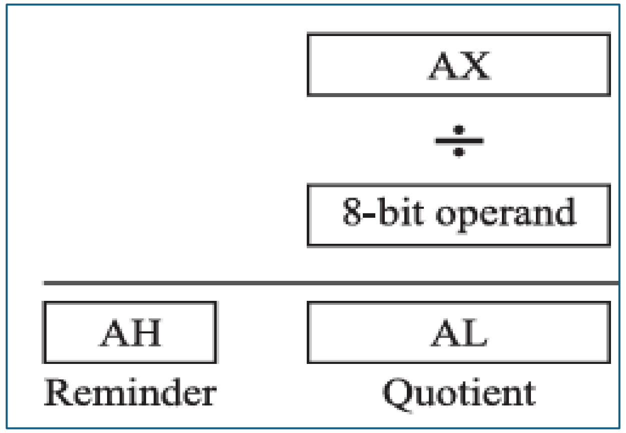
- **INC operation:** This instruction increment the destination operand by l.

```
MOV AX, 5678H
INC AX
```

- **DEC operation:** This instruction decrement the destination operand by l.

```
MOV AX, 5678H
DEC AX
```

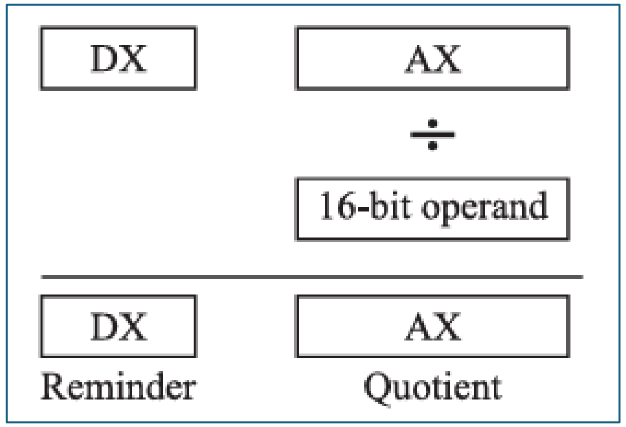
- **DIV operation:** This instruction divides the contents of the AX by a specified source operand. The AX is the implied destination operands.
- After the division, the quotient will be stored into AX and the remainder into DX. When the divisor is of 8 bits, the dividend is AX. And in this case the quotient will be stored in AL and the remainder in AH.



Example

```
MOV AX, 0007H
MOV CL, 02H
DIV CL
```

After this program, the result is available in AL (= 3H) and the remainder is present in AH (= 01H).



Example

```
MOV AX, 0007H
MOV CX, 02H
DIV CX
```

After this program, the result is available in AX (= 3H) and the remainder is present in DX (= 01H).

- **MUL operation**: This instruction multiplies the contents of the AL or the AX by a specified source operand. The AL and the AX are the implied destination operands for 8-bit and l6-bit multiplication.

Example

```
MOV AL, 0FDH
MOV CL, 02H
MUL CL
The result will be in AX = 01FAH
```

Assembly - Conditions

- Conditional execution in assembly language is accomplished by several looping and branching instructions. These instructions can change the flow of control in a program.
- **The CMP instruction** compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not.

Example

```
MOV DX, 10H
CMP DX, 00 ; Compare the DX value with zero
JE L7 ; If yes, then jump to label L7
.
.
L7: ...
```

Example

```
MOV DX, 05H
INC DX
CMP DX, 10 ; Compares whether the counter has reached 10
JLE LP1 ; If it is less than or equal to 10, then jump to LP1
```

Example:

```
CMP AL, BL
JE EQUAL
CMP AL, BH
JE EQUAL
CMP AL, CL
JE EQUAL
NON_EQUAL: ...
EQUAL: ...
```

Example: The following program displays the largest of two variables

```
MOV  AX, 10
CMP AX, 20
JG  End
MOV AX, 20
End: ret
```

Example: The following program displays the lowest of two variables

```
MOV  AX, 10
CMP AX, 20
JL  End
MOV AX, 20
End: ret
```

Example: Write a program to find the summation of the sequence 1, 2, 3, …. 10

```
        MOV CX, 1
        MOV  AX, 0
p1:     CMP CX, 0AH
        JG  stop
        ADD AX, CX
        INC CX
        JMP p1
stop: ret
```

Example: Write a program to find the summation of the sequence $1^2 + 2^2 + 3^2 + \cdots + 10^2$

```
        MOV CX, 1
        MOV  BX, 0
p1:     CMP CX, 0AH
        JG  stop
        MOV AX, CX
        MUL CX
        ADD  BX, AX
        INC CX
        JMP p1
stop: ret
```

Data Defining

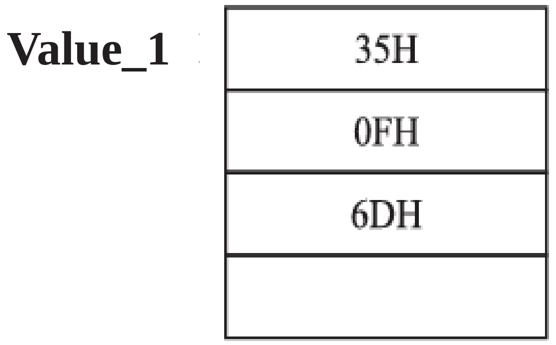
- These directives are used to define the type of data stored in the memory. These directives are DB, DW.
- **DB (Define byte):** The define byte directive is used to allocate and initializes one or more bytes of data. Here name is the symbol assigned to the variable which represents the address of the memory where the data is stored in a particular segment.
- For example:

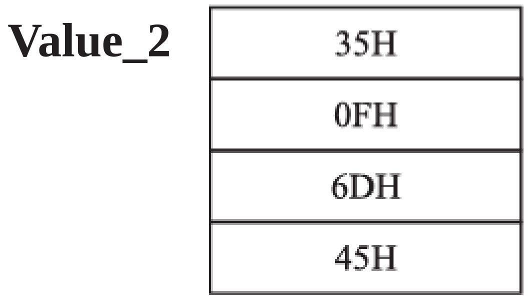
  Value_1 DB S5H, 0FH, 6DH

  In this example Value_1 is the name given to a memory location from where these three data are stored as shown in following Figure
- **DW (Define double byte or define word):** This directive is used to allocate or initialize one or more data in word (l6-bit) format.
- For example:

  Value_2 DW 0F35H, 456DH

  These two words will be stored in memory as shown in the following Figure





```
ORG 100h
MOV AH, var1
MOV AL, var2
MOV sum_0, AH
ADD sum_0, AL
MOV BH, sum_0
RET    ; stops the program.
var1  DB      2h
var2  DB      3h
sum_0 DB    ?
```

ORG 100h is a compiler directive (it tells compiler how to handle the source code). This directive is very important when you work with variables. It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes), so compiler should calculate the correct address for all variables when it replaces the variable names with their offsets.

Why executable file is loaded at offset of 100h? Operating system keeps some data about the program in the first 256 bytes of the CS (code segment), such as command line parameters and etc.

STRING INSTRUCTIONS

- String is a group of bytes/words and their memory is always allocated in a sequential order.
- H.W: write a report about the instructions with examples under this group?