06/1/2022 8086 Microprocessor.

## Jump Instruction

↳ The jump instructions are used to change the sequence of the program execution.

↳ There are two types of jump instructions
1. Unconditional Jump
2. Conditional Jump.

### ① Unconditional Jump Instruction

JMP target address.
↳ is specified in the program using Labels.

For example :

```
    MOV AX, 0000H
    MOV BX, 0004H.
AGAIN: ADD AX, BX
    JMP AGAIN
```

* AGAIN is the label used to point to the address of the instruction ADD AX, BX
* The address pointed by label is evaluated by the assembler using.

MA = BA + EA
↳ IP ± displacement.
BA → CS
Code Segment

## ② Conditional Jump Instruction

↳ In conditional jump instruction, if the condition is satisfied then the program sequence is transfered to the address specified in the instruction else if the condition is not satisfied then the instructions are executed sequentially.

↳ The various conditional jump instructions are :-

| *Instruction* | *Condition* | *Operation* |
|---|---|---|
| JO | O = 1 | Jump on overflow set |
| JNO | O = 0 | Jump on overflow clear |
| JB / JNAE | C = 1 | Jump if below / Jump if not above or equal |
| JAE / JNB | C = 0 | Jump if above or equal / Jump if not below |
| JE / JNZ | Z = 1 | Jump if equal / Jump if not zero |
| JNE / JNZ | Z = 0 | Jump if not equal / Jump if not zero |
| JBE / JNA | C = 1or Z = 1 | Jump if below or equal / Jump if not above |
| JA / JNBE | C = 0 and Z = 0 | Jump if above / Jump if not below or equal |
| JS | S = 1 | Jump on sign set |
| JNS | S = 0 | Jump on sign clear |
| JP / JPE | P = 1 | Jump on parity bit set (parity even) |
| JNP / JPO | P = 0 | Jump on parity bit clear (parity odd) |
| JL / JNGE | S = 1 or O = 1 | Jump if less / Jump if not greater than or equal to |
| JGE / JNL | S = O | Jump if greater than or equal to / Jump if not less |
| JLE / JNG | Z = 1 or S and O = 1 | Jump if less than or equal to / Jump if not greater than |
| JG / JNLE | Z = 0 or S = O | Jump if greater than / Jump if not less than or equal to |

# Loop Instruction

The LOOP instruction executes the part of the program from the level or address specified in the instruction up to loop instruction, CX number of times. After each iteration, CX is decremented automatically.

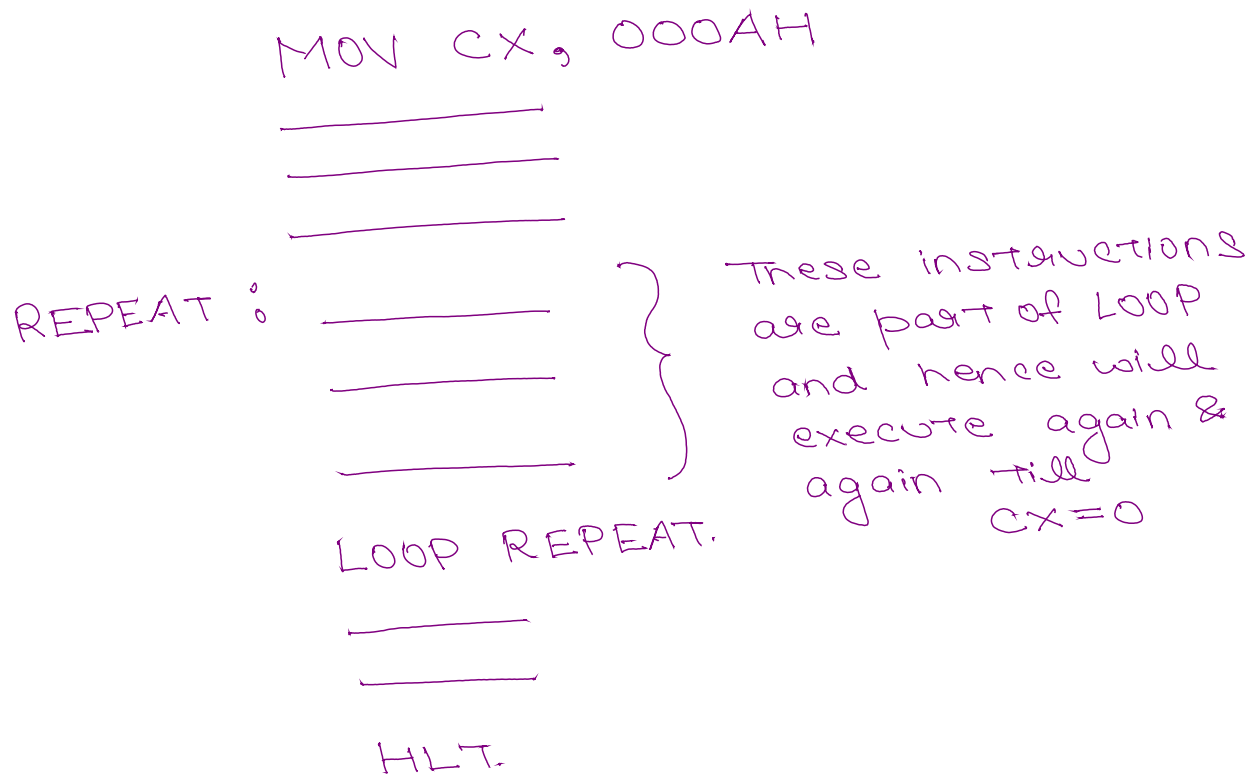
If CX becomes zero, the execution of Loop instruction is completed and then the next instruction of the program will be executed.

For eg:

```
        MOV  CX, 0004H      → CX = 0004H
        MOV  AL, 00H        → AL = 00H
AGAIN:  ADD  AL, 03H        → AL = 03
        LOOP AGAIN          → CX = CX-1 if CX ≠ 0 goto AGAIN
        MOV  AH, AL         → AH = 0CH
        HLT
```

Every time LOOP instruction decrement CX and check whether it has become zero; if CX≠0 then LOOP continues; if CX = 0 LOOP terminates.

General LOOP execution :-




In this program the loop will execute 10 times as CX is initialized with the value 000AH which is 10 in decimal.

## CALL and RET Instructions

↳ The CALL and RET (return) instructions are used with subroutine or a procedure.

↳ There are two types of subroutines or procedures :-

① Intrasegment Subroutine (subroutine within Segment)

② Intersegment Subroutine (subroutine in different Segment)

→ Calling intrasegment Subroutine is NEAR CALL

→ calling intersegment Subroutine is FAR CALL.

→ Syntax for NEAR CALL (within Segment)

```
CALL target_address.
```

→ Syntax for FAR CALL (outside segment)

```
CALL FAR target_address
```

→ target_address is usually specified in the program using Labels.

→ For NEAR CALL :-

Only IP register is stored onto stack.

→ For FAR CALL :-

Both IP register & CS register are stored onto stack.

| CALL target | CALL FAR target. |
|---|---|
| • PUSH IP<br>• JMP target | • PUSH CS<br>• PUSH IP<br>• Jmp target. |

→ Return from intraSegment Subroutine is known as NEAR RETURN

Syntax : RET

→ Return from intersegment subroutine is known as Far Return

Syntax : RET FAR

| RET | RET FAR |
|---|---|
| • POP IP | • POP IP<br>• POP CS |

## Software Interrupt Instruction

Syntax : INT n

↳ INT n is a software interrupt to be serviced.

↳ There are 256 interrupts corresponding to the type (n) from 00H to FFH.

↳ When an INT n instruction is executed following events takes place

(i) PUSHF (stores flags onto stack).

(ii) PUSH CS

(iii) PUSH IP

(iv) CS : IP loaded with vector address of type n

For example: INT 10H

## INTO Instruction

→ This instruction is interrupt on overflow

→ if overflow flag = 1 then automatically INT 4 instruction is invoked.

## IRET Instruction

→ This instruction is Return from Interrupt service routine.

→ When IRET instruction is executed, the value of IP, CS and flags are retrieved from the stack to continue the execution of the main program.

→ IRET instruction is always the last instruction of Interrupt service routine (ISR).

## STRING INSTRUCTIONS

→ A series of data byte is known as a string of bytes and a series of data word is known as string of words.

→ For moving a string of bytes or word, 8086 µp has string instructions.

## ① MOVSB (MOVE String byte)

→ Moves a string of byte, one byte at a time from source memory DS:SI to destination memory ES:DI.

→ Value of SI and DI register are incremented or decremented by 1 each time depending on Direction Flag. (DF)

→ This instruction has no operand.

→ if DF = 0 then

SI = SI + 1

DI = DI + 1

if DF = 1 then

SI = SI − 1

DI = DI − 1

→ SI must contain the start of source string and DI must contain the start of destination string.

→ The number of string bytes or elements to be moved must be loaded in CX register.

→ Special prefix known as repeat is attached in from of MOVSB instruction.

For eg: REP MOVSB

Q Write instruction to move a string of 9 bytes from source address DS:SI to destination address ES:DI.

Assume: DS = 4000H ; ES = 6000H ; SI = 0100H ; DI = 0200H.

```
MOV AX, 4000H
MOV DS, AX
MOV AX, 6000H
MOV ES, AX
MOV CX, 0009H
MOV SI, 0100H
MOV DI, 0200H
CLD
REP MOVSB
```

CLD is used to clear Direction flag.

DF = 0