# *Jump Instructions*

REVIEW OF ASSEMBLY LANGUAGE

# *JUMP INSTRUCTIONS*

**Jump Instructions** are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions.

**There are two types of Jump instructions**:

1. Unconditional Jump Instructions
2. Conditional Jump Instructions

# UNCONDITIONAL JUMP INSTRUCTIONS

These instructions are used to jump on a particular location unconditionally, i.e. there is no need to satisfy any condition for the jump to take place. There are three types of procedures used for unconditional jump. They are:

- **NEAR** (Intra-segment): This procedure targets within the same code segment.
- **SHORT** (Intra-segment): This procedure also targets within the same code segment, the jump range is limited to -128 to 127 from the current address.
- **FAR** (Inter-segment): In this procedure, the target is outside the segment and the size of the pointer is double word.

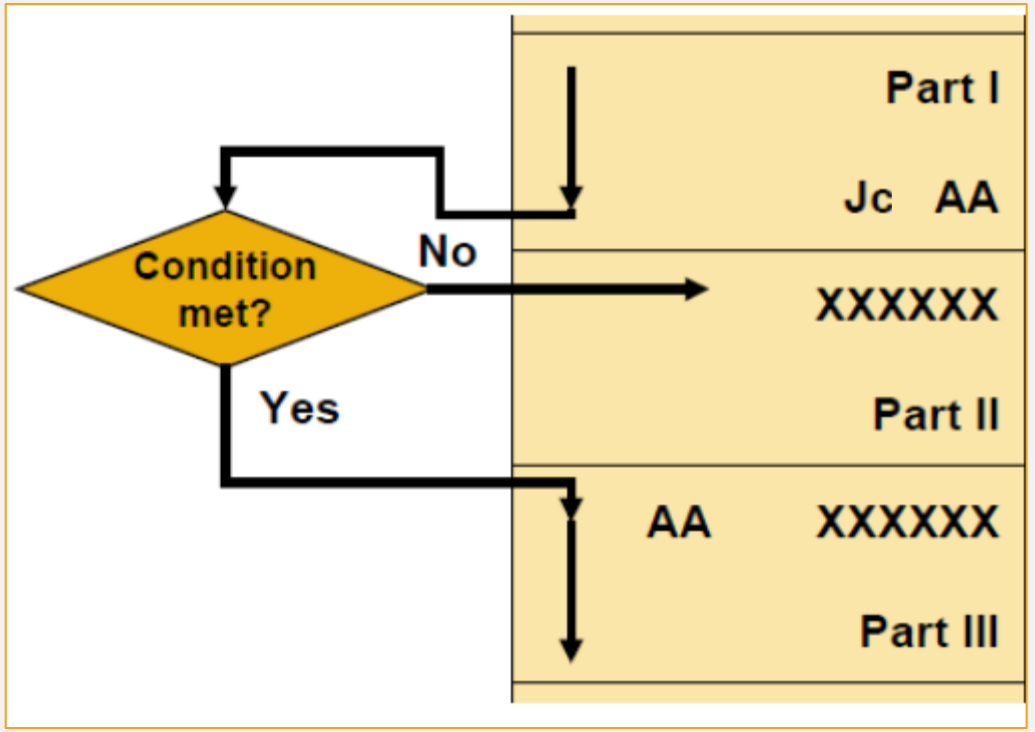
# UNCONDITIONAL JUMP INSTRUCTIONS

| Example | Operand |
|---|---|
| JMP address | Address |
| **No flags** are affected by this instruction | |

```
MOV AH,20H
MOV BL,30H
JMP NEXT
ADD BL,AH
SUB AL,CH
XCHG AL,BL
NEXT: AND AL,BL
RET
```

# CONDITIONAL JUMP INSTRUCTIONS

- In these types of instructions, the processor must check for the particular condition. If it is true, then only the jump takes place else the normal flow in the execution of the statements is maintained.




- The ALU operations set flags in the **status word** (Flag register). The conditional jump statements tests the flag and jump if the flag is set.

- The conditional jump instructions can be divided into **three** groups:

# CONDITIONAL JUMP INSTRUCTIONS

1. Jumps based on general comparison

| Mnemonic | Description | Flags/Registers |
|---|---|---|
| JZ | Jump if zero | ZF = 1 |
| JE | Jump if equal | ZF = 1 |
| | | |
| JNZ | Jump if not zero | ZF = 0 |
| JNE | Jump if not equal | ZF = 0 |

| Mnenomic | Description | Flags/Registers |
|---|---|---|
| JC | Jump if carry | CF = 1 |
| JNC | Jump if not carry | CF = 0 |
| | | |
| JCXZ | Jump if CX = 0 | CX = 0 |
| JECXZ | Jump if ECX = 0 | ECX = 0 |

| Mnenomic | Description | Flags/Registers |
|---|---|---|
| JP | Jump if Parity even | PF = 1 |
| JNP | Jump if Parity odd | PF = 0 |

# CONDITIONAL JUMP INSTRUCTIONS

## 2. Jumps based on comparisons of unsigned operands

| Mnenomic | Description | Flag(s) |
|---|---|---|
| JA | Jump if above (op1 > op2) | CF = 0 & ZF = 0 |
| JNBE | Jump if not below or equal | CF = 0 & ZF = 0 |
| | | |
| JAE | Jump if above or equal | CF = 0 |
| JNB | Jump if not below | CF = 0 |

| Mnenomic | Description | Flag(s) |
|---|---|---|
| JB | Jump if below (op1 < op2) | CF = 1 |
| JNAE | Jump if not above | CF = 1 |
| | | |
| JBE | Jump if below or equal | CF = 1 or ZF = 1 |
| JNA | Jump if not above | CF = 1 or ZF = 1 |

# CONDITIONAL JUMP INSTRUCTIONS

3. Jumps based on comparisons of signed operands

| Mnenomic | Description | Flag(s) |
|---|---|---|
| JG | Jump if greater | SF = 0 & ZF =0 |
| JNLE | Jump if not less than or equal | SF = 0 & ZF =0 |
| | | |
| JGE | Jump if greater than or equal | SF = OF |
| JNL | Jump if not less than | SF = OF |

| Mnenomic | Description | Flag(s) |
|---|---|---|
| JL | Jump if less | SF < > OF |
| JNGE | Jump if not greater than or equal | SF < > OF |
| | | |
| JLE | Jump if less than or equal | ZF = 1 or SF < > OF |
| JNG | Jump if not greater than | ZF = 1 or SF < > OF |

# COMPARE

The cmp (compare) instruction is identical to the sub (**subtract without borrow**) instruction with one important difference - it does not store the difference back into the destination operand.

The generic form of the instruction is:

cmp dest, src perform dest - src and set flags

| Condition | Carry Flag | Zero Flag | Sign Flag |
|---|---|---|---|
| AX = BX | 0 | 1 | 0 |
| AX > BX | 0 | 0 | 0 |
| AX < BX | 1 | 0 | 1 |

```
cmp    reg, reg
cmp    reg, mem
cmp    mem, reg
cmp    reg, immediate data
cmp    mem, immediate data
cmp    eax/ax/al, immediate data
```

# EXAMPLES

EX01: Write an ALP to transfer a block of 256 bytes stored at locations starting at 34000H to locations starting at 36000H.

```
        MOV AX , 3000H
        MOV DS , AX              ; DS=3000*10H=30000
        MOV BX , 0000H           ; Set the offset address
        MOV CX , 256
NEXT:   MOV AL , [BX + 4000H]
        MOV [BX + 6000H] , AL
        INC BX
        DEC CX
        JNZ NEXT
        HLT
```

# *EXAMPLES*

EX02: Write an ALP that adds an array of integers of size 256 byte stored at locations starting at 54000H with another array starting at address 56000H, store the result at locations starting at 58000H.

```
        MOV AX , 5000H
        MOV DS , AX
        MOV BX , 0000H
        MOV CX , 256
NEXT:   MOV AL , [BX + 4000H]
        ADD AL , [BX + 6000H]
        MOV [BX + 8000H] , AL
        INC BX
        DEC CX
        JNZ NEXT
        HLT
```

# EXAMPLES

EX03: Write an ALP to find the maximum byte of a block of 256 bytes starting at 53000H. Store the result at 56000H.

```
        MOV AX , 5000H
        MOV DS , AX
        MOV SI , 3000H
        MOV CX , 256
        MOV AH , 00H
NEXT:   CMP AH , [SI]
        JAE PASS
        MOV AH, [SI]
PASS:   INC SI
        DEC CX
        JNZ NEXT
        MOV DI , 6000       ; Store the maximum byte at 56000H
        MOV [DI] , AH
        HLT
```

# *EXAMPLES*

EX04: Write a program finds the factorial of 08H .

```
        MOV BX , 08
        MOV AX , 0001
LOOP:   MUL BX
        DEC BX
        JNZ LOOP
        HLT
```

# THANK YOU