Chapter 08

JUMP Instructions

Table of content

- Program Execution Transfer Instructions (Branch and Loop Instructions)

■ Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

- **Instructions to transfer the instruction during an execution without any condition −**

CALL − Used to call a procedure and save their return address to the stack.

RET − Used to return from the procedure to the main program.

JMP − Used to jump to the provided address to proceed to the next instruction.

- **Instructions to transfer the instruction during an execution with some conditions −**

JA/JNBE − Used to jump if above/not below/equal instruction satisfies.

JAE/JNB − Used to jump if above/not below instruction satisfies.

JBE/JNA − Used to jump if below/equal/ not above instruction satisfies.

JC − Used to jump if carry flag CF = 1

JE/JZ − Used to jump if equal/zero flag ZF = 1

JG/JNLE − Used to jump if greater/not less than/equal instruction satisfies.

JGE/JNL − Used to jump if greater than/equal/not less than instruction satisfies.

JL/JNGE − Used to jump if less than/not greater than/equal instruction satisfies.

JLE/JNG − Used to jump if less than/equal/if not greater than instruction satisfies.

JNC − Used to jump if no carry flag (CF = 0)

JNE/JNZ − Used to jump if not equal/zero flag ZF = 0

JNO − Used to jump if no overflow flag OF = 0

JNP/JPO − Used to jump if not parity/parity odd PF = 0

JNS − Used to jump if not sign SF = 0

JO − Used to jump if overflow flag OF = 1

JP/JPE − Used to jump if parity/parity even PF = 1

JS − Used to jump if sign flag SF = 1

- **Instructions to transfer the instruction during an execution without any condition – unconditional jump**

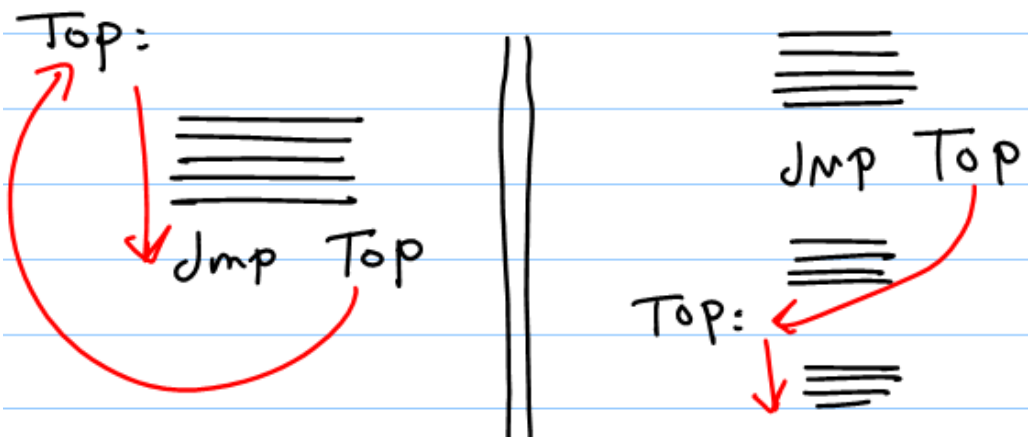
JMP − Used to jump to the provided address to proceed to the next instruction.

"JMP label" is the unconditional jump in which control is transferred unconditionally to the target label. Unconditional jump can take the following forms.

- **SHORT JUMP**, which is specified by the format "JMP SHORT label". In this jump, the address of the target location is within –128 to +127 bytes of the current IP. It works like the conditional jump.
  - **All conditional jumps are short jumps**. In a short jump the address of the target must be within –128 to +127 bytes of the memory

- A conditional jump is a two-byte instruction: one byte is the opcode of the J condition and the second value is a value between 00 to FF.
- In a backward jump the second byte is the 2's complement of the displacement value.

- **NEAR JUMP**, which is the default has the format "**JMP label**". It jumps within the current code segment. It is exactly same as the short jump, except that the target address can be anywhere within the range of +32767 to –32768.
  - `Jmp destination` ➔ jump to label
  - `JMP repeat` ➔ jump to label named repeat



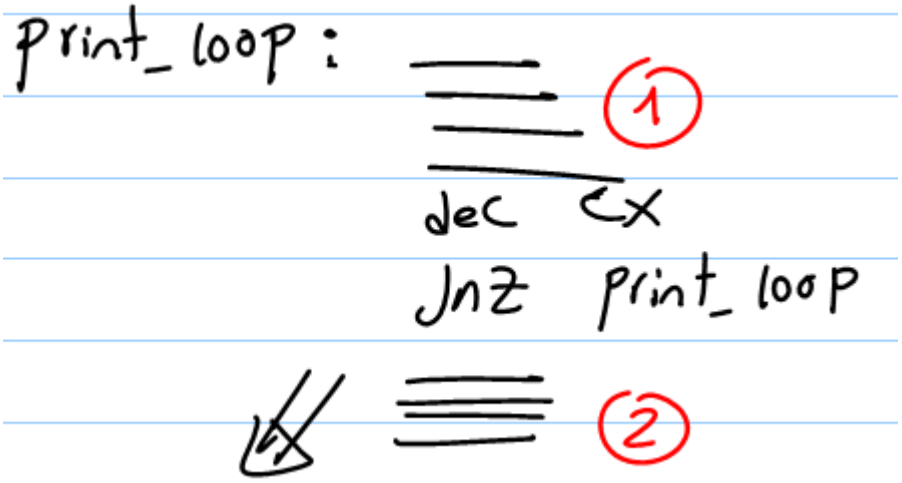
- **FAR JUMP**, has the format "**JMP FAR PTR label**". This is the jump out of the current code segment, meaning that not only the IP, but also the CS is replaced with new values.

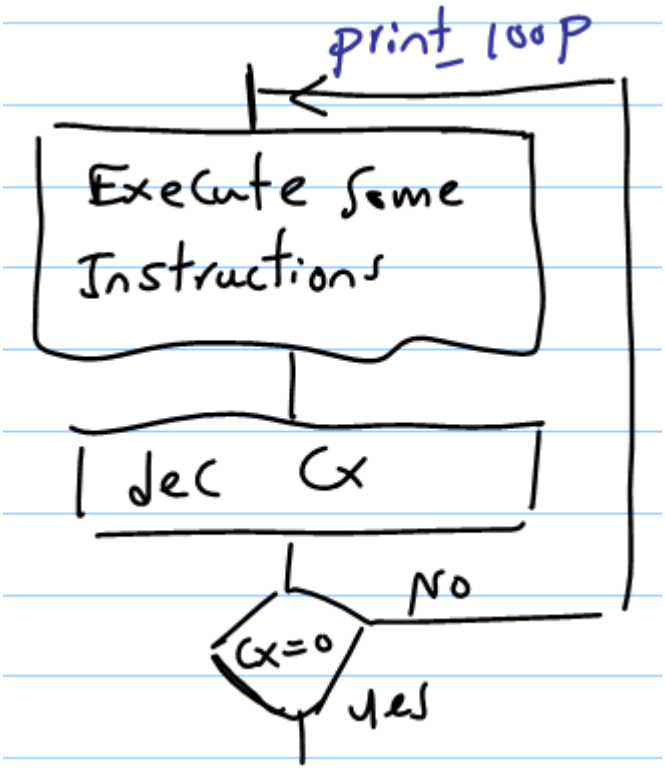
■ conditional jump

- In the conditional jump, control is transferred to a new location if a certain condition is met, but where is the condition? Processor use the value of certain flag bit to decide how to branch and jump
- If the condition happened, the processor will load the IP with the value of destination label
- Example:

JNZ `print_loop`

- Jump if the result of the above instruction is not zero to print_loop
- Or jump if the zero-flag bit is 0 to print_loop
- If CX = 0 ➔ execute section 2
- If NOT ➔ go to print_loop to execute section 1





- **Unsigned** conditional jump

Instruction	Flag tested	description

JA / JNBE	CF=0 and ZF=0	Jump if above // jump if not below or equal to
JAE / JNB	CF=0	Jump if above or equal to // jump if not below
JB / JNAE	CF=1	Jump if below // jump if not above or equal to
JBE / JNA	CF=1 OR ZF=1	Jump if below or equal to // jump if not above

- **signed** conditional jump

Instruction	Flag tested	description
JG / JNLE	ZF=0 & SF=OF	Jump if Greater than // jump if not less or equal to
	((SF XOR OF) or ZF) = 0	
JGE / JNL	SF=OF	Jump if greater than or equal to // jump if not less than
	((SF XOR OF) = 0	
JL / JNGE	SF XOR OF = 1	Jump if less than // jump if not greater than or equal to
JLE / JNG	((SF XOR OF) or ZF) = 1	Jump if less than or equal to // jump if not greater

- Note: "**above**" and "**below**" refer to the relationship of two unsigned values; "**greater**" and "**less**" refer to the relationship of two signed values.
- **single flag conditional jump**

JE / JZ	ZF = 1	Jump if equal // jump if zero
JNE / JNZ	ZF = 0	Jump if not equal // jump if not zero
JC	CF = 1	Jump if carry
JNC	CF = 0	Jump if no carry
JP / JPE	PF = 1	Jump if parity even
JNP / JPO	PF = 0	Jump if no parity // jump if parity odd

JS	SF = 1	Jump if sign (negative)
JNS	SF = 0	Jump if no sign (positive)
JO	OF = 1	Jump if overflow happened
JNO	OF = 0	Jump if no overflow

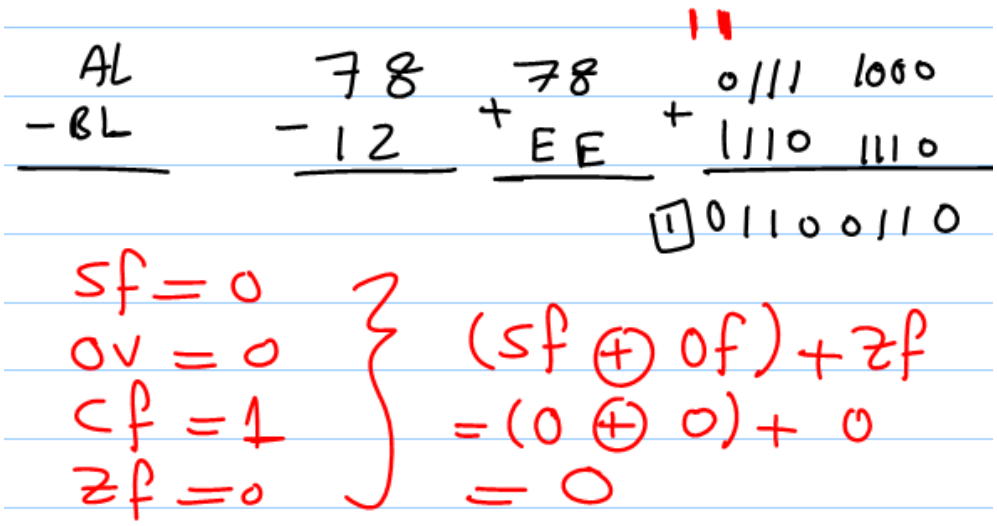
- **Example**

```
CMP AL, BL
JG  below; signed compare
```

Check whether (SF XOR OF) OR ZF = 0

if AL = 78H

BL = 12 H



Condition happened

You should not worry about these mathematics and calculations ➔ just JG ➔ jump if greater

Do not involve yourself with flags

- **Example**

`DEC CX` `JNZ repeat`	Not only CMP used

- **Example**

`If AX = 7FFFH`
`BX = 8000H`
`CMP AX, BX`
`JA below`
	Unsigned
	AX < BX
	Do not jump

`If AX = 7FFFH`
`BX = 8000H`
`CMP AX, BX`
`JG below`
	signed
	AX > BX
	jump

- **Example**

Assume that AX, and BX contain signed numbers, write a program to put the maximum number in CX

```
          MOV CX, AX
          CMP AX, BX
          JG  exit
          MOV CX,BX
Exit:     ---------
          ---------
          ---------
```

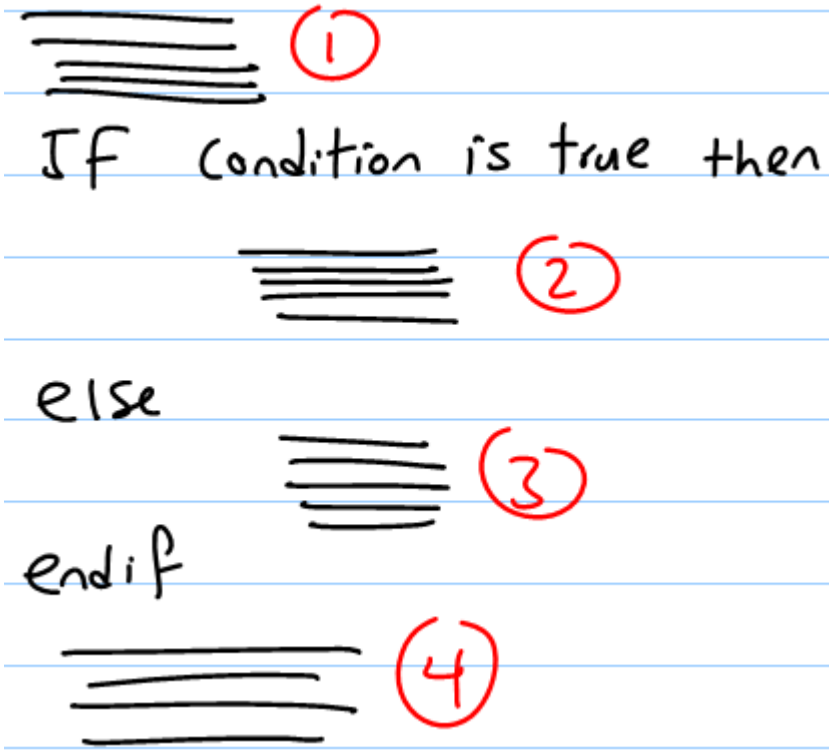
```
          MOV CX, AX
          CMP BX, CX
          JLE  next
          MOV CX,BX
next:     ---------
          ---------
          ---------
```

- **Example**

  Check the number contained in AX register, then replace it with the absolute value; $AX = |AX|$

`CMP AX,0H`
`JG Exit`
`NEG AX`
`Exit: --------`
	If AX < 0 then
	AX = - AX
	End if

Simulate IF…THEN…ELSE…ENDIF



If condition is OK ➔ section 1 then followed by section 2 then section 4 is executed

If condition is not OK ➔ section 1 then followed by section 3 then section 4 is executed

- **Example**

  Assume that AL, BL contain characters as ASCII code, print character with low ASCII code

```
        MOV AH,02H
        MOV DL, AL
        CMP AL, BL
        JA  ELSE_
        JMP print
ELSE_: MOV DL, BL
Print: INT 21H
```

```
If AL < BL
   Display AL
ELSE
   Display BL
END IF
```

Simulate SWITCH …CASE

- **Example**

```
        CMP AX, 0
        JL  Less
        JE  Equal
        MOV BX, 1
        JMP Exit
Less:   MOV BX, -1
        JMP Exit
Equal: MOV BX, 0
Exit:
```

```
CASE AX
        < 0 :  put -1 in BX
        = 0 :  put 0 in BX
        > 0 :  put 1 in BX
END_CASE
```

- **Example**

  Check the content of AL register, if it is 1 or 3 display O, and if it is 2 or 4 display E.

```
        MOV AH, 02H
        CMP AL, 1
        JE  ODD
        CMP AL, 3
        JE  ODD
        CMP AL, 2
        JE  EVEN
        CMP AL, 4
        JE  EVEN
        JMP Exit
ODD:    MOV DL, 'O'
        JMP PRINT
EVEN:   MOV DL, 'E'
PRINT:  INT 21H
Exit:   --------
        --------
        --------
```

- **Example**

  Read a character from keyboard, if it is capital letter, print it

```
        MOV AH, 01H
        INT 21H
        CMP AL,'A'
        JNGE END_IF
        CMP AL, 'Z'
        JNLE END_IF
        MOV DL, AL
        MOV AH, 02
        INT 21H
END_IF:---------
        ---------
        ---------
```

Read a Character into AL
If (‘A’ <= character AND character <= ‘Z’) then
    Display character
End_IF

- **Example**

  Read a character from keyboard, if it is 'y' or 'Y', print it

Read character from keyboard into AL
IF (character = ‘y’ OR character = ‘Y’) then
Display character
Else
Terminate the program
End_IF

- **Example**

  Write a program to display 80 stars '*'

	for 80 times do display “*” End_for