8.1 Jump instructions (Unconditional jump):

The main program control instruction, jump (JMP), allows the programmer to skip sections of a program and branch to any part of the memory for the next instruction. A conditional jump instruction allows the programmer to make decisions based upon numerical tests. The results of numerical tests are held in the flag bits, which are then tested by conditional jump instructions.

Mnem.	Meaning	Format	Operation	Flags
JMP	Unconditional Jump	JMP Operand	Jump to the address specified by the operand.	None

There are two basic kinds of unconditional jump. These kinds are as shown below:

1- Intrasegment
- Short-Label (IP=label)
- Near-Label (IP=label)
- Regptr16 (IP=(reg.))
- Memptr16 (IPcontents of memory location)

2- Intrasegment
- Far-label (IP=first 16-bit, CS=seconds 16-bit)
- Memptr32 (IP=contents of first two bytes.

1- **Intrasegment:** this is a jump within the current segment .This type of jump is achieved by just modifying the value in IP.
- Short Jump: Format → JMP short Label (8 bit)
- Near Jump: Format → JMP near Label (16 bit)

Example: Consider the following example of an unconditional jump instruction: JMP 1234H It means jump to address 1234H. However, the value of the address encoded in the instruction is not 1234H. Instead, it is the difference between the incremented value in IP and 1234H. This offset is encoded as either an 8-bit constant (short label) or a 16-bit constant (near label), depending on the size of the difference.
- Mem.16: Format → JMP Mem.16
- Reg.16:: Format → JMP Reg.16

The jump-to address can also be specified indirectly by the contents of a memory location or the contents of a register, corresponding to the Mem.16 and Reg.16 operand, respectively. Just as for the Near-label operand, they both permit a jump to any address in the current code segment.

Example: JMP BX uses the contents of register BX for the offset in the current code segment that is: the value in BX is copied into IP. To specify an operand as a pointer to memory, the various addressing modes of 8086 can be used, For instance: JMP [BX] uses the contents of BX as the offset address of the memory location that contains the value of IP (Mem.16 operand).
Example: JMP [SI] will replace the IP with the contents of the memory locations pointed by DS:SI and DS:SI+1

2- **Intersegment:** this is a jump out of the current segment. This type of jump requires modification of the contents of both CS and IP.
- Far Jump: Format → JMP far Label (32 bit label)

Example: JMP label (4-byte address) or JMP 1234:5678 Transfers control to another part of the program. 4-byte address may be entered in this form: 1234h:5678H, first value is a segment second value is an offset.
- Mem.32: Format → JMP Mem.32

An indirect way to specify the offset and code-segment address for an intersegment jump is by using the Mem.32 operand. This time the four consecutive memory bytes starting at the specified address contain the offset address and the new code segment address respectively.

Example: JMP DWORD[DI] It uses the contents of DS and DI to calculate the address of the memory location that contains the first word of the pointer that identifies the location to which the jump will take place. The two word pointer starting at this address is read into IP and CS to pass control to the new point in the program.

Short Jump: Short jumps (see Figure 8–1) are called relative jumps because they can be moved, along with their related software, to any location in the current code segment without a change. This is because the jump address is not stored with the opcode. Instead of a jump address, a distance, or displacement, follows the opcode. The short jump displacement is a distance represented by a 1-byte signed number whose value ranges between +127 and -128. When the microprocessor executes a short jump, the displacement is sign extended and added to the instruction pointer (IP) to generate the jump address within the current code segment. The short jump instruction branches to this new address for the next instruction in the program.

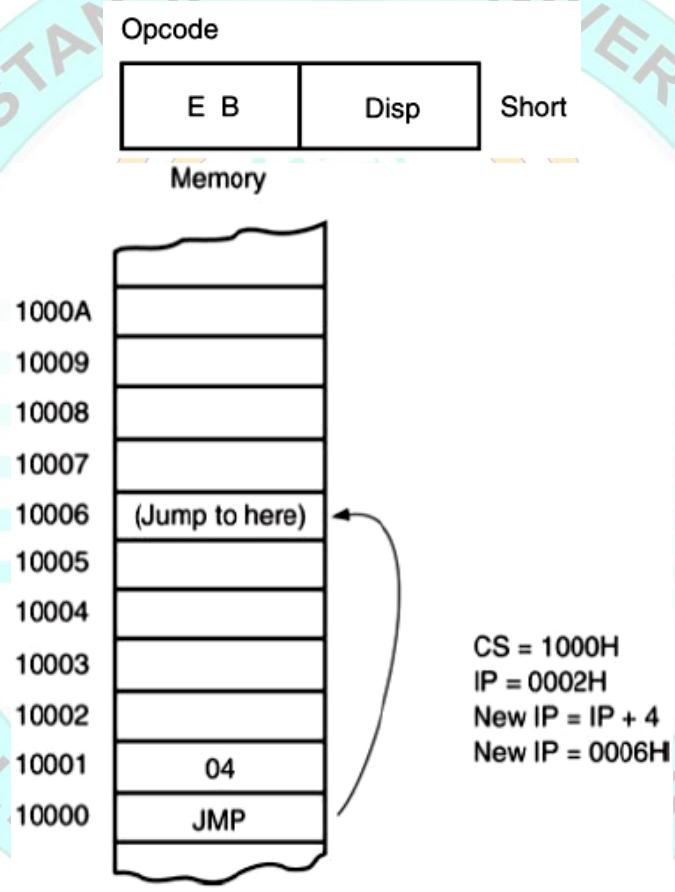


Figure (8–1): A short jump to four memory locations beyond the address of the next instruction.

Near Jump: The near jump is similar to the short jump, except that the distance is farther. A near jump passes control to an instruction in the current code segment located within ±32K bytes from the near jump instruction. The near jump is a 3-byte instruction that contains an opcode followed by a signed 16-bit displacement. The signed displacement adds to the instruction pointer (IP) to generate the jump address. Because the signed displacement is in the range of ±32K, a near jump can jump to any memory location within the current real mode code segment. Figure 8–2

illustrates the operation of the real mode near jump instruction. The near jump is also relocatable (as was the short jump) because it is also a relative jump. If the code segment moves to a new location in the memory, the distance between the jump instruction and the operand address remains the same. This allows a code segment to be relocated by simply moving it.

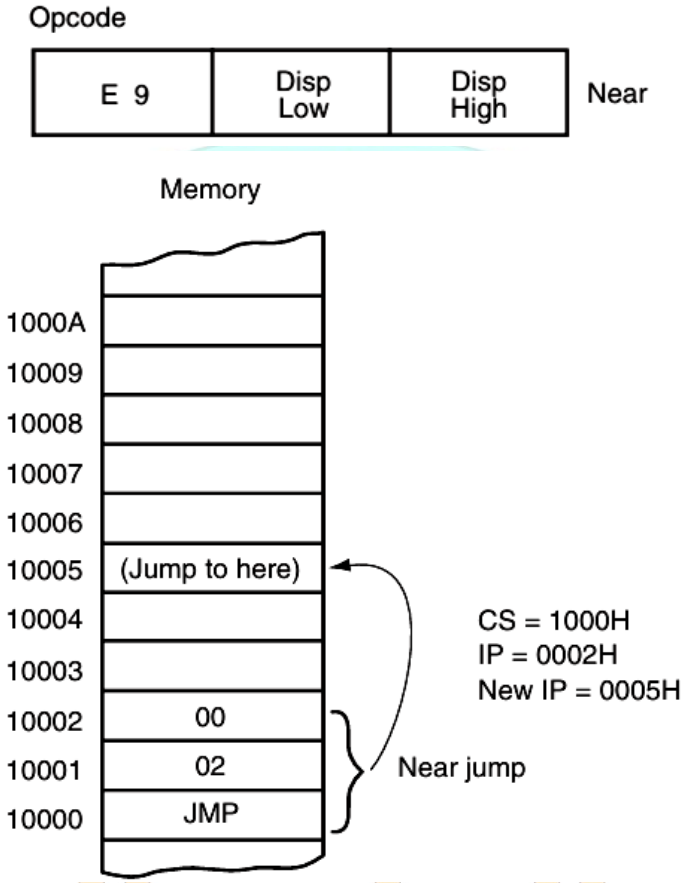
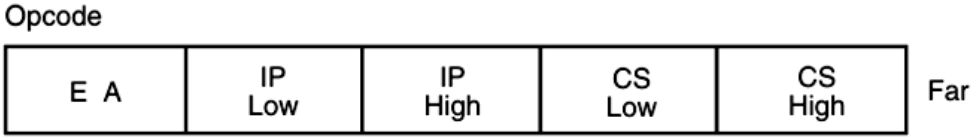


Figure (8–2): A near jump that adds the displacement (0002H) to the contents of IP.

Far Jump: A far jump instruction (see Figure 8–3) obtains a new segment and offset address to accomplish the jump. Bytes 2 and 3 of this 5-byte instruction contain the new offset address; bytes 4 and 5 contain the new segment address. The offset address, which is either 16 or 32 bits, contains the offset address within the new code segment.



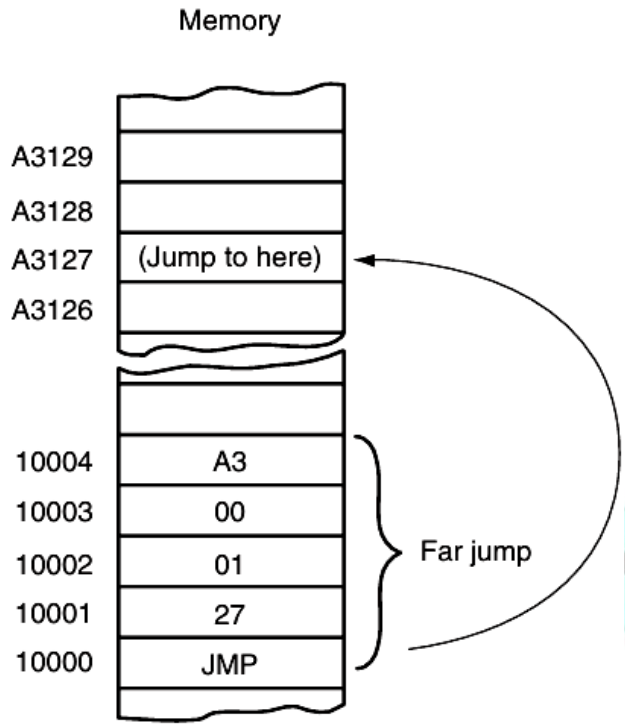


Figure (8–3): A far jump instruction replaces the con tents of both CS and IP with 4 bytes following the opcode.

Example: Assume the following state of 8086: (CS)=1075H, (IP)=0300H, (SI)=A00H, (DS)=400H, & (DS:A00)=10H, (DS:A01)=B3H, (DS:A02)=22H, (DS:A03)=1AH. To what address is program control passed if each of the following JMP instruction is execute? (a) JMP 10. (b) JMP 1000H. (c) JMP [SI]. (d) JMP SI. (e) JMP FAR [SI]. (f) JMP 3000:1000.

Solution:

(a)PA= CS:IP = 1075:10

(b) PA= CS:IP = 1075:1000

(c) EA= (SI) ⇒ PA= DS: SI ⇒PA=DS:A00 & DS:A01 ⇒ IP= B310

∴PA of the step in the program =CS:IP = 1075:B310

(d) IP= (SI) ⇒ PA= CS:IP = 1075:A00

(e) First 16 bit (IP) & second 16bit (CS) ⇒PA=DS:SI⇒32bit from memory

DS:A00=10H & DS:A01=B3H ⇒ IP= B310H

DS:A02=22H & DS:A03=1AH ⇒ CS=1A22H

∴PA of the step in the program = CS:IP = 1A22: B310

(f) PA= CS:IP = 3000:1000

8.2 Jump instructions (Conditional jump):

The conditional jump instructions test the following flag bits: sign (S), zero (Z), carry (C), parity (P), and overflow (0). If the condition under test is true, a branch to the label associated with the jump instruction occurs. If the condition is false, the next sequential step in the program executes. For example, a JC will jump if the carry bit is set. The operation of most conditional jump instructions is straightforward because they often test just one flag bit, although some test more than one. Relative magnitude comparisons require more complicated conditional jump instructions that test more than one flag bit.

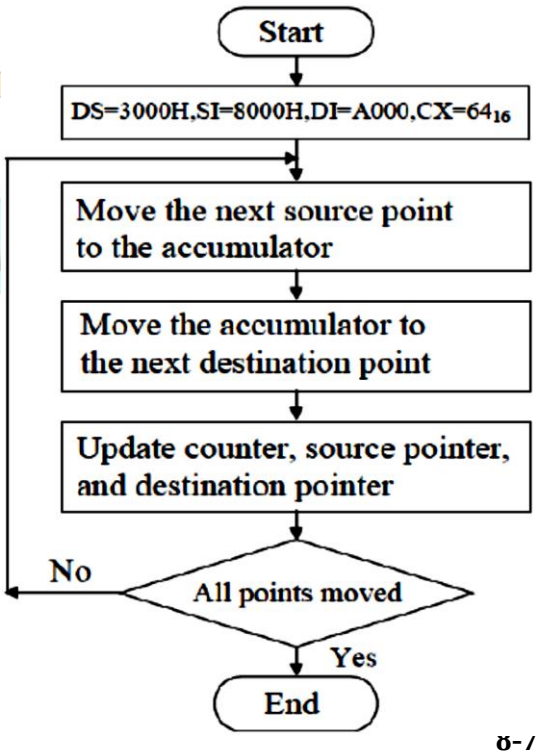
Mnem.	Meaning	Description	Condition
JA	Jump Above	Used to jump if above instruction satisfies.	CF=0 and ZF=0
JAE	Jump Above or Equal	Used to jump if above or equal instruction satisfies.	CF=0
JB	Jump Below	Used to jump if below instruction satisfies.	CF=1
JBE	Jump Below or Equal	Used to jump if below or equal instruction satisfies.	CF=1 and ZF=1
JC	Jump Carry	Used to jump if carry flag	CF=1
JCXZ	CX register is zero	Used to jump to the provided address if CX = 0	(CF or ZF)=0
JE	Jump Equal	Used to jump if equal	ZF=1
JG	Jump Greater	Used to jump if greater instruction satisfies.	ZF=0 and SF=OF
JGE	Jump Greater or Equal	Used to jump if greater or equal instruction satisfies.	SF=OF
JL	Jump Less	Used to jump if less instruction satisfies.	(SF xor OF)=1
JLE	Jump Less or Equal	Used to jump if less or equal instruction satisfies.	(SF xor OF)or ZF=1
JNA	Jump not Above	Used to jump if not above instruction satisfies.	CF=1 and ZF=1
JNAE	Jump not Above nor Equal	Used to jump if not above nor equal instruction satisfies.	CF=1
JNB	Jump not Below	Used to jump if not below instruction satisfies.	CF=0
JNBE	Jump not Below nor Equal	Used to jump if not below nor equal instruction satisfies.	CF=0 and ZF=0
JNC	Jump not Carry	Used to jump if no carry flag	CF=0

JNE	Jump not Equal	Used to jump if not equal	ZF=0
JNG	Jump not Greater	Used to jump if not greater than instruction satisfies.	(SF xor OF)or ZF=1
JNGE	Jump not Greater nor Equal	Used to jump if not greater nor equal than instruction satisfies.	(SF xor OF)=1
JNL	Jump not Less	Used to jump if not less than instruction satisfies.	SF=OF
JNLE	Jump not Less nor Equal	Used to jump if not less nor equal than instruction satisfies.	ZF=0 and SF=OF
JNO	Jump not Overflow	Used to jump if no overflow flag	OF=0
JNP	Jump not Parity	Used to jump if not parity odd	PF=0
JNS	Jump not Sign	Used to jump if not sign	SF=0
JNZ	Jump not Zero	Used to jump if not Zero	ZF=0
JO	Jump Overflow	Used to jump if overflow flag	OF=1
JP	Jump Parity	Used to jump if parity odd	PF=1
JPE	Jump Parity Even	Used to jump if parity even	PF=1
JPO	Jump Parity odd	Used to jump if parity odd	PF=0
JS	Jump Sign	Used to jump if sign	SF=1
JZ	Jump Zero	Used to jump if Zero	ZF=1

Example: Write a program to move a block of 100_{10} consecutive bytes of data string at offset address 8000H in memory to another block of memory location starting at offset address A000H. Assume that both blocks are in the same data segment value 3000H.

Solution:

```
        MOV AX, 3000H
        MOV DS, AX
        MOV SI, 8000H
        MOV DI, A000H
        MOV CX, 64H
NXT:    MOV AH, [SI]
        MOV [DI], AH
        INC SI
        INC DI
        DEC CX
        JNZ NXT
        HLT
```



Because both signed and unsigned numbers are used in programming, and because the order of these numbers is different, there are two sets of conditional jump instructions for magnitude comparisons. Figure below shows the order of both signed and unsigned 8-bit numbers. The 16-bit numbers follow the same order as the 8-bit numbers except that they are larger. Notice that an FFH (255) is above the 00H in the set of unsigned numbers, but an FFH (-1) is less than 00H for signed numbers. Therefore, an unsigned FFH is above 00H, but a signed is less than 00H.

Signed and unsigned numbers follow different orders.