3.2.6 Program Flow Control Instructions

The purpose of a jump instruction is to alter the execution path of instructions in the program. The code segment register and instruction pointer keep track of the next instruction to be fetched for execution.so a jump ivolves altering the contents of either the IP register or CS and IP registers together. In this way an execution continues at an address other than the next sequential instruction.

There two types of jump instructions: unconditional jump and conditional jump.

Unconditional Jump

This type of jump instruction (**JMP**) allows the programmer to skip sections of a program and branch to any part of the memory for the next instruction. In an unconditional jump, no status requirements are imposed for the jump to occur. That is, as the instruction is executed, the jump always takes place to change the execution sequence.

The general format of this instruction is shown below

Mnemonics	**Meaning**	**Format**	**Operation**	**Flags affected**
Jmp	Unconditional jump	Jump operand	Jump to the address specified by operand	none

The destination (target) operand specifies the addres of the instruction being jump to. This operand can be immediate value, a general-purpose register, or a memory location, according to this, the **JMP** instruction can be classified into:

a. **Intrasegment:** is limited with addresses within the current code segment.
b. **Intersegment:** permits jumps from one code segment to another.

Mohammed Ali Saffah

***Important Note*:** Jump instructions specified with a **Short-Label, Near-Label, Memptr 46** or **Regptr16** represent **intrasegment** jumps while jump instructions specified with a **far-label** and **Memptr32** represent intersegment jumps.

According to that, the unconditional jumps can be classified into these types:

i. **Short Jump:** is a 2-byte instruction that allows jumps or branches to memory locations within +127 and -128 bytes from the memory location following the jump.

Ex. **JMP** 32h

$\mathbf{IP}_{new}=\mathbf{IP}_{old}+32h$

Next address= $\mathbf{CS*10}+\mathbf{IP}_{new}$

JMP	Displacement
Byte 1	Byte 2

ii. **Near Jump:** the 3-byte near jump allows a branch or jump within ±32K bytes (anywhere) from the instruction in the current code segment.

JMP 1234h

$\mathbf{IP}_{new}=\mathbf{IP}_{old}+1234h$

Next address= $\mathbf{CS*10}+\mathbf{IP}_{new}$

JMP	Disp. High	Disp. Low
Byte 1	Byte 2	Byte 3

iii. **Far Jump:** the far jump obtains a new segment and offset address to accomplish the jump. Notice that the operand (32-bit) specified with the jump is a new values to the **IP** and **CS** and not a displacement as in short and near jumps.

JMP 1456:F2E6h

After executing the instruction shown above, the following values will be accomplished:

IP=F2E6h CS=1456h

Next address=$\mathbf{CS*10}+\mathbf{IP}_{new}$

Mohammed Ali Saffah

iv. **Reg16-bit** or **Mem16-bit:** the unconditional jump address can also be specified indirectly by the contents of a register or memory location.

JMP BX ; $\mathbf{IP_{new}}$=BX

JMP [BX] ; $\mathbf{IP_{new}}$**(low)=**[BX]

; $\mathbf{IP_{new}}$**(High)**=[BX+1]

v. **Mem32-bit:**as using far jump here an indirect way is used to specify the offset and code segment address and that done by using **Mem32-bit** operand (four consecutive memory bytes starting at specified address).

JMP DWORD [BX] ; IP-low=[BX]

; IP-high=[BX+1]

; CS-low=[BX+2]

; CS-high=[BX+3]

Mohammed Ali Saffah