Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

The process of comparison performed by the CMP instruction is basically a subtraction operation. The source operand is subtracted from the destination operand. However the result of this subtraction is not saved. Instead, based on the result the appropriate flags are set or reset.

EXAMPLE: lets the destination operand equals 100110012 and that the source operand equals 000110112. Subtraction the source from the destination, we get

١٠٠١١٠٠١

\- ٠٠٠١١٠١١

____

Replacing the destination operand with its 2's complement and adding

```
   10011001
+  11100101
-----------
  011111102
```

1. No carry is generated from bit 3 to bit 4, therefore, the auxiliary carry flag AF is at logic 0.
2. There is a carry out from bit 7. Thus carry flag CF is set.
3. Even through a carry out of bit 7 is generated; there is no carry from bit 6 to bit 7. This is an overflow condition and the OF flag is set.

4. There are an even number of 1s, therefore, this makes parity flag PF equal to 1.
5. Bit 7 is zero and therefore sign flag SF is at logic 0.
6. The result that is produced is nonzero, which makes zero flag ZF logic 0.

3.JUMP Instruction

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 executed. Thus a jump instruction involves altering the contents of these registers. In this way, execution continues at an address other than that of the next sequential instruction. That is, a jump occurs to another part of the program.

There two type of jump instructions:

a. Unconditional jump.
b. Conditional jump.

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. See Figure 16.

Instruction	Meaning	Format	Operation	Flag affected
JMP	Unconditional jump	JMP operand	Jump is to the address specified by operand	None

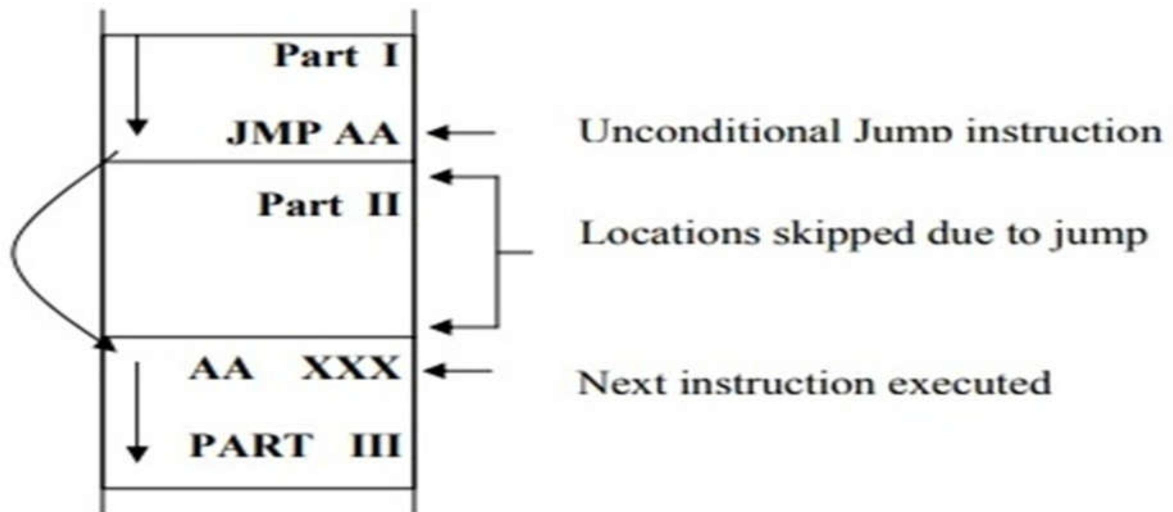

Figure 16: Unconditional jump program sequence.

On the other hand, for a **conditional jump** instruction, status conditions that exist at the moment the jump instruction is executed decide whether or not the jump will occur. If this condition or conditions are met, the jump takes place, otherwise execution continues with the next sequential instruction of the program. The conditions that can be referenced by a conditional jump instruction are status flags such as carry (CF), parity (PF), and overflow (OF). See Figure 17

Instruction	Meaning	Format	Operation	Flags affected
JCC	Conditional jump	Jcc operand	If the specific condition cc is true, the jump to the address specified by the operand is initiated, otherwise the next instruction is executed	None

The following table lists some of the conditional jump instructions

Instruction	Meaning
JAE/JNB	Jump if above or equal jump if not below
JB/JNAE	Jump if below/jump if not above or equal
JC	Jump if carry
JCXZ	Jump if CX is zero
JE/JZ	Jump if equal/jump if zero
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/ jump if not zero
JNO	Jump if not overflow
JNP/JPO	Jump if parity/jump if parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/jump if parity Even
JS	Jump if sign

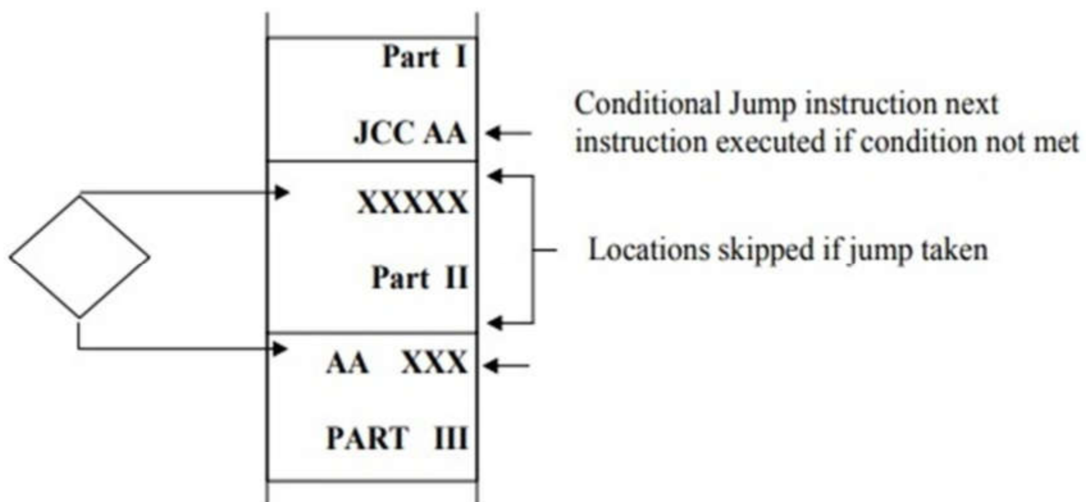

Figure 17: Conditional jump program sequence.

EXAMPLE: write a program to move a block of N bytes of data starting at offset address BLK1ADDR to another block starting at offset address BLK2ADDR. Assume that both blocks are in the same data segment, whose starting point is defined by the data segment address DATASEGMADDR.

```
MOV AX, DATASEGADDR
MOV DS, AX
MOV SI, BLK1ADDR
MOV DI, BLK2ADDR
MOV CX, N
NXTPT: MOV AH, [SI]
MOV [DI], AH
INC SI
INC DI
DEC CX
JNZ NXTPT
HLT
```

4.Push and POP Instruction

It is necessary to save the contents of certain registers or some other main program parameters. These values are saved by pushing them onto the stack. Typically, these data correspond to registers and memory locations that are used by the subroutine.

The instruction that is used to save parameters on the stack is the push (PUSH) instruction and that used to retrieve them back is the pop (POP) instruction. Notice a general-purpose register, a segment register (excluding CS), or a storage location in memory as their operand.

- Execution of a PUSH instruction causes the data corresponding to the operand to be pushed onto the top of the stack. For instance, if the instruction is **PUSH AX** the result is as follows:

((SP)-1) ← (AH)

((SP)-2) ← (AL)

(SP) ← (SP)-2

This shows that the two bytes of the AX are saved in the stack part of memory and the stack pointer is decrement by 2 such that it points to the new top of the stack.

- On the other hand, if the instruction is POP AX Its execution results in

(AL) ((SP))

(AH) ((SP) + 1)

(SP) (SP)+2

The saved contents of AX are restored back into the register.

- We also can save the contents of the flag register and if saved we will later have to restore them. These operations can be accomplished with the push flags **(PUSHF)** and pop flags **(POPF)** instructions, respectively. Notice the **PUSHF** save the contents of the flag register on the top of the stack. On the other hand, **POPF** returns the flags from the top of the stack to the flag register.

Instruction	Meaning	Operation	Flags affected
PUSHF	Push flags onto stack	((SP)) ← (flag)	None
POPF	Pop flags from stack	(flag) ← ((SP))	OF, DF, IF, TF, SF, ZF, AF, PF, CF

5.String Instructions

The microprocessor is equipped with special instructions to handle string operations. By "string" we mean a series of data words or bytes that reside in consecutive memory locations. There are five basic string instructions in the instruction set of the 8086, these instruction are:

a. Move byte or work string (MOVS, MOVSB, and MOVSW).
b. Compare string (CMPS).
c. Scan string (SCAS).
d. Load string (LODS)
e. Store string (STOS).

They are called the basic string instructions because each defines and operations for one element of a string.