## 7.Control Transfer or Branching Instruction:

The control transfer instructions transfer the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location where the flow of execution is going to be transferred.

**This type of instructions are classified in two types:**

i. **Unconditional control Transfer (Branch) Instructions:**
In case of unconditional control transfer instructions; the execution control is transferred to the specified location independent of any status or condition. The CS and IP are unconditionally modified to the new CS and IP.

ii. **Conditional Control Transfer (Branch) Instructions:**
In the conditional control transfer instructions, the control is transferred to the specified location provided the result of the previous operation satisfies a particular condition, otherwise, the execution continues in normal flow sequence. The results of the previous operations are replicated by condition code flags. In other words, using type of instruction the control will be transferred to a particular specified location, if a particular flag satisfies the condition.

**Unconditional Branch Instructions:**

| | | | |
|---|---|---|---|
| **CALL** | **RET** | **JUMP** | **IRET** |
| **INT N** | **INT O** | **LOOP** | |

**CALL: Unconditional Call:** This instruction is used to call a subroutine procedure from a main program. The address of the procedure may specify directly or indirectly depending on the address mode.
There are again two types of procedures depending on whether it is available in the same segment (Near CALL, i.e. + 2K displacement) or in another segment (Far CALL, i.e. anywhere outside the segment). The modes for them are respectively called as intrasegment and intersegment addressing (i.e. address of the next instruction) and CS onto the stack along with the flags and loads the CS and IP registers, respectively, with the segment and offset addresses of the procedure to be called.

**RET: Return from the Procedure**: At each CALL instruction, the IP and CS of the next instruction is pushed onto stack, before the control is transferred to the procedure. At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with flags are retrieved into the CS, IP and flag registers from the stack and the execution of the main program continues further. In case of a FAR procedure the current contents of SP points to IP and CS at the time of return. While in case of a NEAR procedure, it points to only IP. Depending on the byte of procedure and the SP contents, the RET instruction is of four types:

i. Return within a segment.
ii. Return within a segment adding 16-bit immediate displacement to the SP contents.
iii. Return intersegment.
iv. Return intersegment adding 16-bit immediate displacement to the SP contents.

**INT N: Interrupt Type N**: In the interrupt structure of 8086/8088, 256 interrupts are defined corresponding to the types from 00H to FFH. When an INT N instruction is executed, the TYPE byte N is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from the hexadecimal multiplication (N * 4) as offset address and 0000 as segment address. In other words, the multiplication of type N by 4 (offset) points to a memory block in 0000 segment, which contains the IP and CS values of the interrupt service routine. For the execution of this instruction, the IP must be enabled.

**Ex:** The INT 20H will find out the address of the interrupt service routine follows:

INT 20H

Type * 4 = 20 X 4 = 80H

Pointer to IP and CS of the ISR is 0000:0080H
The arrangement of CS and IP addresses of the ISR in the interrupt rector table is as follows.

**INTO: Interrupt on overflow**: This is executed, when the overflow flag OF is set. The new contents of IP an CS are taken from the address 0000:0000 as explained in INT type instruction. This is equivalent to a type 4 instruction.

**JMP:** Unconditional Jump: This instruction unconditionally transfer the control of execution to the specified address using an 8-bit or 16-bit displacement (intrasegment relative, short or long) or CS:IP (intersegment direct for) No flags are affected by this instruction. Corresponding to the three methods of specifying jump address, the JUMP instruction has the following three formats.

| | | | |
|---|---|---|---|
| JUMP | DISP 8-bit | | Intrasegment, relative, near jump |
| JUMP | DISP 16-bit | DISP 16-bit | Intrasegment, relative, For jump |
| JUMP | IP(LB) IP(UB) | CS(LB) CS(UB) | Intersegment, direct, jump |

**IRET:** Return from ISR: When interrupt service routine is to be called, before transferring control to it, the IP, CS and flag register are stored on to the stack to indicate the location from where the execution is to be continued, after the ISR is executed. So, at the end of each ISR, when IRET is executed, the valuesof IP, CS and flags are retrieved from the stack to continue the execution of the main program. The stack is modified accordingly.

**LOOP:** Loop unconditionally: This instruction executes the part of the program from the label or address specified in the instruction up to the loop instruction, CX number of times. At each iteration, CX is decremented automatically, in other words, this instruction implements DECREMENT counter and JUMP IF NOT ZERO structure.

**Ex**:

```
        MOV CX,0005H ; Number of times in CX
        MOV BX, 0FF7H ; Data to BX
Label   MOV AX, CODE1
        OR BX,AX
        AND DX,AX
        LOOP Label
```

The execution proceeds in sequence, after the loop is executed, CX number of times. IF CX is already 00H, the execution continues sequentially. No flags are affected by this instruction.

**Conditional Branch Instructions:**

| LOOPE/LOOPZ | LOOPNE/LOOPNZ |
|---|---|

When these instructions are executed, they transfer execution control to the address specified relatively in the instruction, provided the condition in the opcode is satisfied, otherwise, the execution continues sequentially. The conditions, here means the status of the condition code flags. These type of instructions don't affect any flags. The address has to be specified in the instruction relatively in terms of displacement, which must lie within – 80H to 7FH (or –128 to 127) bytes from the address of the branch instruction. In other words, only short jumps can be implemented using conditional branch instructions. A label may represent the displacement, if it has within the above-specified range.

The different 8086/8088 conditional branch instructions and their operations are listed in Table1

| SL.No | Mnemonic | Displacement | Operation |
|---|---|---|---|
| 1 | JZ/JE | Label | Transfer execution control to address 'Label', if ZF=1 |
| 2 | JNZ/JNE | Label | Transfer execution control to address 'Label', if ZF=0 |
| 3 | JS | Label | Transfer execution control to address 'Label', if SF=1 |
| 4 | JNS | Label | Transfer execution control to address 'Label', if SF=0 |
| 5 | JO | Label | Transfer execution control to address 'Label', if OF=1 |
| 6 | JNO | Label | Transfer execution control to address 'Label', if OF=0 |
| 7 | JP/JPE | Label | Transfer execution control to address 'Label', if PF=1 |
| 8 | JNP | Label | Transfer execution control to address 'Label', if PF=0 |
| 9 | JB/JNAE/JC | Label | Transfer execution control to address 'Label', if CF=1 |
| 10 | JNB/JNE/JNC | Label | Transfer execution control to address 'Label', if CF=0 |
| 11 | JBE/JNA | Label | Transfer execution control to address 'Label', if CF=1 or ZF=1 |
| 12 | JNBE/JA | Label | Transfer execution control to address 'Label', if CF=0 or ZF=0 |
| 13 | JL/JNGE | Label | Transfer execution control to address 'Label', if neither SF=1 nor OF=1 |
| 14 | JNL/JGE | Label | Transfer execution control to address 'Label', if neither SF=0 nor OF=0 |
| 15 | JNE/JNC | Label | Transfer execution control to address |

| | | | |
|---|---|---|---|
| | | | 'Label', if ZF=1or neither SF nor OF is 1 |
| 16 | JNLE/JE | Label | Transfer execution control to address 'Label', if ZF=0 or at least any are of SF & OF is 1 |

Table:1 Conditional branch instructions.

**8. Flag Manipulation and Processor Control Instructions:**

These instructions control the functioning of the available hardware inside the processor chip.

These are categorized into 2 types:

a) flag manipulation instructions

b) Machine control instructions.

The flag manipulation instructions directly modify same of the flags of 8086.

**The flag manipulation instructions** and their functions are as follows:

| |
|---|
| **CLC** – clear carry flag |
| **CMC** – Complement carry flag |
| **STC** – Set carry flag |
| **CLD** – clear direction flag |
| **STD** - Set direction flag |
| **CLI** – clear interrupt flag |
| **STI** – Set interrupt flag |

These instructions modify the carry (CF), Direction (DF) and interrupt (IF) flags directly. The DF and IF, which may be the processor operation; like interrupt responses and auto increment or auto-decrement modes. Thus the respective instructions may also be called as machine or processor control instructions. The other flags can be modified using POPF and SAHF instructions, which are termed as data transfer instructions. No direct instructions are available for modifying the status flags except carry flags. The machine control instructions don't require any operational.

The **machine control instructions** supported by 8086/8088 are listed as follows along with their functions:

| | |
|---|---|
| **WAIT** | – Wait for Test input pin to go low |
| **HLT** | – Halt the processor |
| **NOP** | – No operation |
| **ESC** | – Escape to external device like NDP |
| **LOCK** | – Bus lock instruction prefix. |

## ASSEMBLER DIRECTIVES

- Assembler directives are the commands to the assembler that direct the assembly process.
- They indicate how an operand is treated by the assembler and how assembler handles the program.
- They also direct the assembler how program and data should arrange in the memory.
- ALP's are composed of two type of statements.

(i) The instructions which are translated to machine codes by assembler.

(ii) The directives that direct the assembler during assembly process, for which no machine code is generated.

**1. ASSUME:** Assume logical segment name.

The ASSUME directive is used to inform the assembler the names of the logical segments to be assumed for different segments used in the program .In the ALP each segment is given name.

Syntax: ASSUME segreg:segname,…segreg:segname

Ex: ASSUME CS:CODE

ASSUME CS:CODE,DS:DATA,SS:STACK