**Assembly - Conditions**

Conditional execution in assembly language is accomplished by several looping and branching instructions. These instructions can change the flow of control in a program. Conditional execution is observed in two scenarios −

|  |  |
| --- | --- |
| **SN** | **Conditional Instructions** |
| 1 | **Unconditional jump**  This is performed by the JMP instruction. Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps. |
| 2 | **Conditional jump**  This is performed by a set of jump instructions j<condition> depending upon the condition. The conditional instructions transfer the control by breaking the sequential flow and they do it by changing the offset value in IP. |

Let us discuss the CMP instruction before discussing the conditional instructions.

**CMP Instruction**

The CMP instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not. It does not disturb the destination or source operands. It is used along with the conditional jump instruction for decision making.

**Syntax**

CMP destination, source

CMP compares two numeric data fields. The destination operand could be either in register or in memory. The source operand could be a constant (immediate) data, register or memory.

**Example**

CMP DX, 00 ; Compare the DX value with zero

JE L7 ; If yes, then jump to label L7

.

.

L7: ...

CMP is often used for comparing whether a counter value has reached the number of times a loop needs to be run. Consider the following typical condition −

INC EDX

CMP EDX, 10 ; Compares whether the counter has reached 10

JLE LP1 ; If it is less than or equal to 10, then jump to LP1

**Unconditional Jump**

As mentioned earlier, this is performed by the JMP instruction. Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps.

**Syntax**

The JMP instruction provides a label name where the flow of control is transferred immediately. The syntax of the JMP instruction is −

JMP label

**Example**

The following code snippet illustrates the JMP instruction −

MOV AX, 00 ; Initializing AX to 0

MOV BX, 00 ; Initializing BX to 0

MOV CX, 01 ; Initializing CX to 1

L20:

ADD AX, 01 ; Increment AX

ADD BX, AX ; Add AX to BX

SHL CX, 1 ; shift left CX, this in turn doubles the CX value

JMP L20 ; repeats the statements

**Conditional Jump**

If some specified condition is satisfied in conditional jump, the control flow is transferred to a target instruction. There are numerous conditional jump instructions depending upon the condition and data.

Following are the conditional jump instructions used on signed data used for arithmetic operations −

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Flags tested** |
| JE/JZ | Jump Equal or Jump Zero | ZF |
| JNE/JNZ | Jump not Equal or Jump Not Zero | ZF |
| JG/JNLE | Jump Greater or Jump Not Less/Equal | OF, SF, ZF |
| JGE/JNL | Jump Greater/Equal or Jump Not Less | OF, SF |
| JL/JNGE | Jump Less or Jump Not Greater/Equal | OF, SF |
| JLE/JNG | Jump Less/Equal or Jump Not Greater | OF, SF, ZF |

Following are the conditional jump instructions used on unsigned data used for logical operations −

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Flags tested** |
| JE/JZ | Jump Equal or Jump Zero | ZF |
| JNE/JNZ | Jump not Equal or Jump Not Zero | ZF |
| JA/JNBE | Jump Above or Jump Not Below/Equal | CF, ZF |
| JAE/JNB | Jump Above/Equal or Jump Not Below | CF |
| JB/JNAE | Jump Below or Jump Not Above/Equal | CF |
| JBE/JNA | Jump Below/Equal or Jump Not Above | AF, CF |

The following conditional jump instructions have special uses and check the value of flags −

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Flags tested** |
| JXCZ | Jump if CX is Zero | none |
| JC | Jump If Carry | CF |
| JNC | Jump If No Carry | CF |
| JO | Jump If Overflow | OF |
| JNO | Jump If No Overflow | OF |
| JP/JPE | Jump Parity or Jump Parity Even | PF |
| JNP/JPO | Jump No Parity or Jump Parity Odd | PF |
| JS | Jump Sign (negative value) | SF |
| JNS | Jump No Sign (positive value) | SF |

The syntax for the J<condition> set of instructions −

Example,

CMP AL, BL

JE EQUAL

CMP AL, BH

JE EQUAL

CMP AL, CL

JE EQUAL

NON\_EQUAL: ...

EQUAL: ...

**Example**

The following program displays the largest of three variables. The variables are double-digit variables. The three variables num1, num2 and num3 have values 47, 72 and 31, respectively −

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov ecx, [num1]

cmp ecx, [num2]

jg check\_third\_num

mov ecx, [num3]

check\_third\_num:

cmp ecx, [num3]

jg \_exit

mov ecx, [num3]

\_exit:

mov [largest], ecx

mov ecx,msg

mov edx, len

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov ecx,largest

mov edx, 2

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov eax, 1

int 80h

section .data

msg db "The largest digit is: ", 0xA,0xD

len equ $- msg

num1 dd '47'

num2 dd '22'

num3 dd '31'

segment .bss

largest resb 2

When the above code is compiled and executed, it produces the following result −

The largest digit is:

47