

# Chapter #3

## Assembly:

### Loops and Control structures

# Control structures

## CONDITIONS and LOOPS

- Conditional execution in assembly language is accomplished by several looping and branching instructions.
- These instructions can change the flow of control in a program.
- They can be
  - Unconditional
  - Conditional
- 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.

# CONDITIONS and LOOPS

- **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.

# 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 Instruction

- 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

- 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:

— **JMP** *destination\_label*

- Purpose: To Transfer control to another part of the program.
- Example:

**MOV AX, 2**

**MOV BX, 2**

**JMP LABEL\_SUB**

**ADD AX, BX** ;this instruction will never execute

**LABEL\_SUB:**

**SUB AX, BX**

# Conditional Jump

- If some specified **condition is satisfied** in conditional jump, the control flow is transferred to a target instruction.
- Syntax:
  - **Jxxx destination\_label**
  - where **xxx** represents the **condition**
- If condition is **true**, the next instruction to be executed is the one at **destination\_label**.
- If condition is **false**, the instruction immediately following the jump is done next.

- There are numerous conditional jump instructions depending upon the condition and data.

Instruction	Description
JE/JZ	Jump Equal or Jump Zero
JNE/JNZ	Jump not Equal or Jump Not Zero
JG/JNLE	Jump Greater or Jump Not Less/Equal
JGE/JNL	Jump Greater or Jump Not Less
JL/JNGE	Jump Less or Jump Not Greater/Equal
JLE/JNG	Jump Less/Equal or Jump Not Greater



# Example

- The following code fragment compares two nos and display the largest

Section **.data**

```
num1 dd '47'
```

```
num2 dd '22'
```

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

Section **.bss**

```
largest resb 2
```

section **.text:**

```
mov ax,num1
```

```
cmp ax,num2
```

```
jg msg
```

```
jg _exit
```

```
.....
```

# Example

- The following code fragment compares two nos and display a message

section **.text:**

**mov** al, 25 ; set al to 25.

**mov** bl, 10 ; set bl to 10.

**cmp** al, bl ; compare al - bl.

**je** equal ; jump if al = bl (zf = 1).

; if it gets here, then al <> bl,

**mov** eax, 4

**mov** ebx, 1

**mov** ecx, 'N'

**mov** edx, 2

**int** 80h

# Example

**jmp exit** ; so print 'n', and jump to exit.

**equal:** ; if gets here, then al = bl, so print 'y'.

**mov eax, 4**

**mov ebx, 1**

**mov ecx, 'Y'**

**mov edx, 2**

**int 80h**

**exit:**

**mov eax, 1**

**mov ebx, 0**

**int 80h**

# Loops

- These instructions are used to execute the given instructions for number of times.
- The processor instruction set, however, includes a group of loop instructions for implementing iteration.
- The basic LOOP instruction syntax:
  - *LOOP label*
- Where, label is the target label that identifies the target instruction as in the jump instructions.
- The LOOP instruction assumes that the AX register contains the loop count.
- The **JMP** instruction can be used for implementing loops.

# Loops

- For example, the following code snippet can be used for executing the loop-body 10 times.

**MOV CL, 10**

**L1:**

**<LOOP-BODY>**

**DEC CL**

**JNZ L1 ;it will jump to L1 until the value of CL becomes Zero**

- When the loop instruction is executed, the **AX register** is decremented and the control jumps to the target label, **until the AX register value, i.e., the counter reaches the value zero.**
- For example the above code snippet could be written as:

***mov AX,10***

***l1:***

***<loop body>***

***loop l1***

# Arrays

- When a variable is declared it could be initialized with some specific value. The initialized value could be specified in hexadecimal, decimal or binary form.
- For example, we can define a word variable months in either of the following way:
  - MONTHS DW 12
  - MONTHS DW 0CH
  - MONTHS DW 0110B
- The data definition directives can also be used for defining a one dimensional array. Let us define a one dimensional array of numbers.
  - NUMBERS DW 34, 45, 56, 67, 75, 89
- The above definition declares an array of six words each initialized with the numbers 34, 45, 56, 67, 75, 89. This allocates  $2 \times 6 = 12$  bytes of consecutive memory space.
- The symbolic address of the first number will be NUMBERS and that of the second number will be NUMBERS + 2 and so on.

# Arrays

- Let us take up another example. You can define an array named inventory of size 8, and initialize all the values with zero, as:
- ```
INVENTORY      DW 0
                DW 0
                DW 0
                DW 0
                DW 0
                DW 0
                DW 0
                DW 0
```
- Which, can be abbreviated as:
- ```
INVENTORY DW 0, 0 , 0 , 0 , 0 , 0 , 0 , 0
```
- The **TIMES** directive can also be used for multiple initializations to the same value. Using TIMES, the INVENTORY array can be defined as
  - ```
INVENTORY TIMES 8 DW 0
```