

# Flow Control Instructions

Course Code: CSC 2106

Course Title: Computer Organization and Architecture



**Dept. of Computer Science**  
**Faculty of Science and Technology**

<b>Lecturer No:</b>	<b>6.2</b>	<b>Week No:</b>	<b>8</b>	<b>Semester:</b>	<b>Summer 21-22</b>
<b>Lecturer:</b>	<i>Noboranjana Dey; noboranjana@aiub.edu</i>				

# Lecture Outline



Decision making and repeating statement

Jump and loop instructions

Algorithm conversion to assembly language

High-Level Language Structures

# Branches with Compound Conditions



Sometimes the branching condition in an IF or CASE takes the form

**condition\_1' AND condition\_2'**

**or**

**condition\_1 OR condition\_2**

Where condition 1 and condition:2 are either true or false. We will refer to the

First of these as an AND condition and to the second as an OR condition.

# Example : AND



An AND condition is true if and only if Condition\_1 and Condition\_2 are both true. Likewise, if either condition is false, then the whole thing is false.

**Read a character, and if it's an uppercase letter, display it.**

**Read a character (into AL)**

**IF (character >= 'A') and (character <= 'Z')**

**THEN**

**display character**

**END IF**

# Converting to Assembly



;read a character

MOV AH,1

INT 21H

; if ('A' <= char> and (char <= 'Z')

CMP AL, 'A' ;char >'A'

JL END\_IF ;no exit

CMP AL, 'Z'

JG END\_IF ;no exit

MOV DL, AL.

MOV AH, 2

INT 21H

END\_IF:

# OR Conditions



Condition\_1 OR condition\_2 is true if at least one of the conditions is true; it is only false when both conditions are false.

**Read a character. If it's "y" or "Y", display it; otherwise, terminate the program.**

**Read a character (into AL)**

**IF (character = 'y') OR (character = 'Y')**

**THEN**

**display it**

**ELSE**

**terminate the program**

**END IF**

# Assembly Conversion



**MOV AH,1**

**INT 21H**

**CMP AL,'y' ;AL=='y'**

**JE THEN**

**CMP AL, 'Y';char ~ 'Y'?**

**JE THEN**

**;yes, go to display it**

**JMP ELSE\_**

**;no - Terminate**

**THEN:**

**MOV AH,2 ;prepare to  
display**

**MOV DL,AL ;get char**

**INT 21H ;display it**

**JMP END IF ;and exit –**

**ELSE\_:**

**MOV AH, 4CH**

**INT 21H ;DOS exit**

**END\_IF:**

# Looping Structure



A loop is a sequence of instructions that is repeated.

The number of times to repeat may be known in advance, or  
It may depend on conditions

**1. FOR LOOP**

**2. WHILE LOOP**

**3. REPEAT LOOP**



# FOR LOOP



FOR LOOP is a loop structure in which the loop statements are repeated a **known number of times (a count-controlled loop)**. In pseudo code,

**FOR** loop\_count times **DO**

**Statements**

**END\_FOR**

The **LOOP** instruction can be used to implement a FOR loop. i.e.

**LOOP** destination\_label

The **counter** for the loop is the **register CX** which is initialized to loop\_count.

Execution of the **LOOP** Instruction causes **CX to be decremented** automatically,

# FOR LOOP



The control is transferred to destination\_label until CX becomes 0.

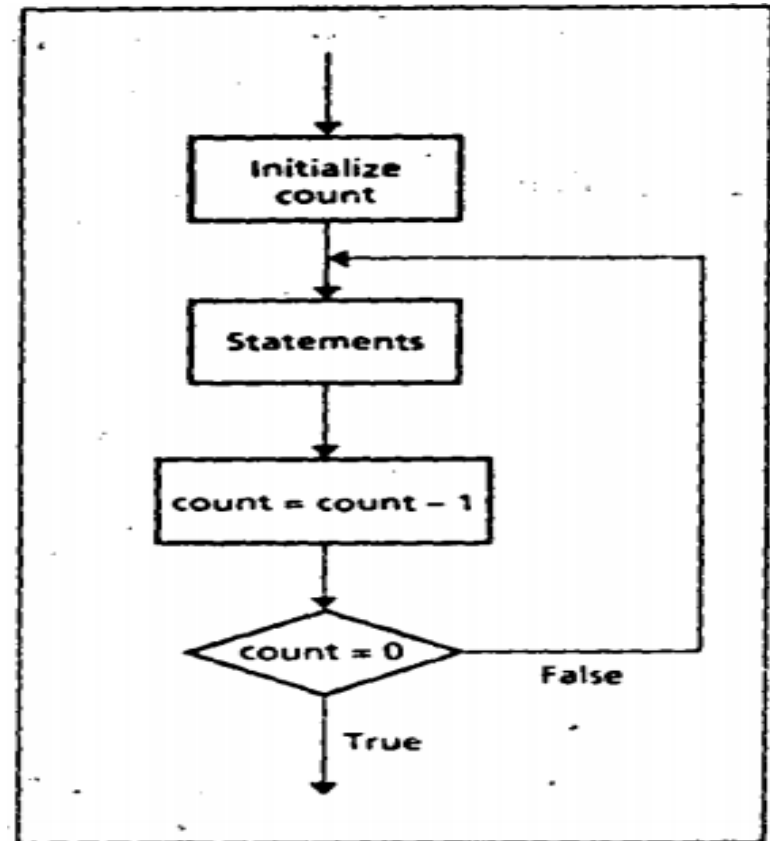
A FOR LOOP can be implemented using the LOOP instruction:

**TOP:**

;initialize CX to loop\_count

;body of the loop

**LOOP TOP**



# Example:



Write a count-controlled loop to display a row of 80 stars:

FOR 80 times DO

display '\*'

END\_FOR

MOV CX,80

MOV AH,2

MOV DL, '\*'

TOP:

INT 21H

LOOP TOP

# JCXZ and The LOOP



FOR LOOP executes at least once.

if CX contains 0 when the loop is entered, the LOOP instruction causes CX to be decremented to FFFFh

The loop is then executed FFFFh=65535 times more!

To Prevent this, the instruction **JCXZ (jump if CX is zero)** may be used before the loop.  
Its syntax

**JCXZ destination\_label**

# Use of JCXZ



If CX contains 0, control transferred to the destination label. So a loop implemented as follows is bypassed if CX is 0:

**JCXZ SKIP**

**TOP:**

    ;body of the loop

    LOOP TOP

**SKIP:**

# WHILE LOOP

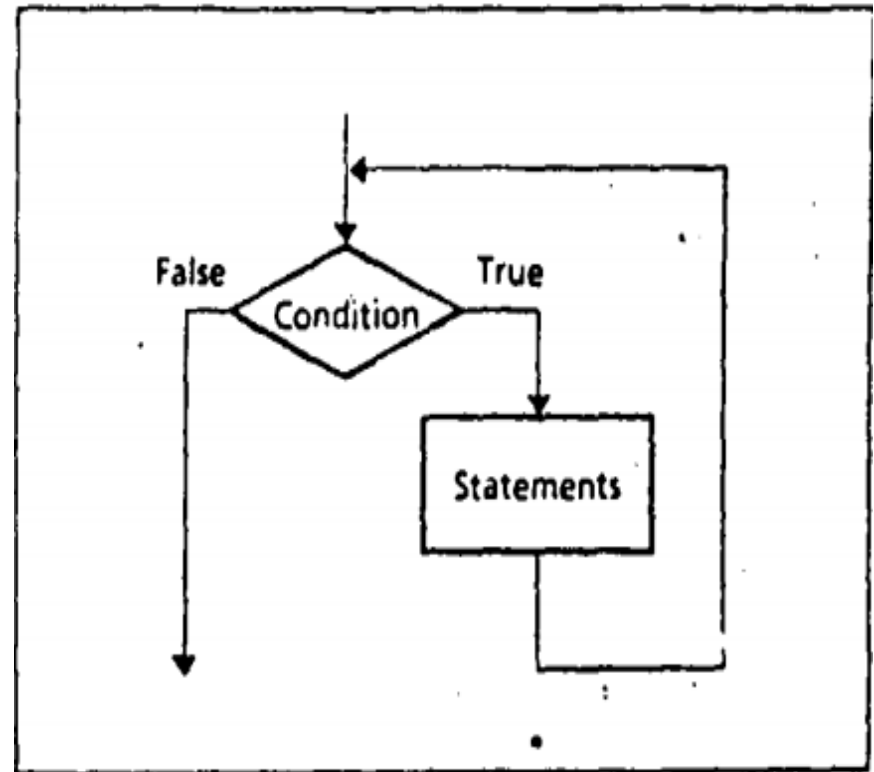


This WHILE LOOP depends on a condition.

**WHILE** *condition* DO

statements

**END\_WHILE**



# WHILE LOOP



The condition is **checked** at the **top of the loop**.

If **true**, the statements are executed;

If **false**, the program goes on to whatever follows.

It is possible the condition will be **false initially**, in which case the loop body is **not executed at all**.

The loop executes as long as the condition is true

# Example: WHILE LOOP



Write some code to count the number of characters in an input line.

Initialize count to 0

Read a character

WHILE character <> carriage\_return DO

count =count + 1

read a character

END\_WHILE

```
MOV DX,0 ; char count
MOV AH,1
INT 21H

WHILE_:
    CMP AL,0DH      ; CR ?
    JE END_WHILE    ;yes, exit
    INC DX          ; not CR so inc

    INT 21H ; read next char

    JMP WHILE_ ; loop again

END_WHILE:
```



# WHILE LOOP Insights



A WHILE loop **checks** the terminating condition at the **top of the loop**,

So, you must make sure that **any variables involved** in the condition are **initialized before the loop is entered**.

So you read a character before entering the loop, and **read another one at the bottom**.

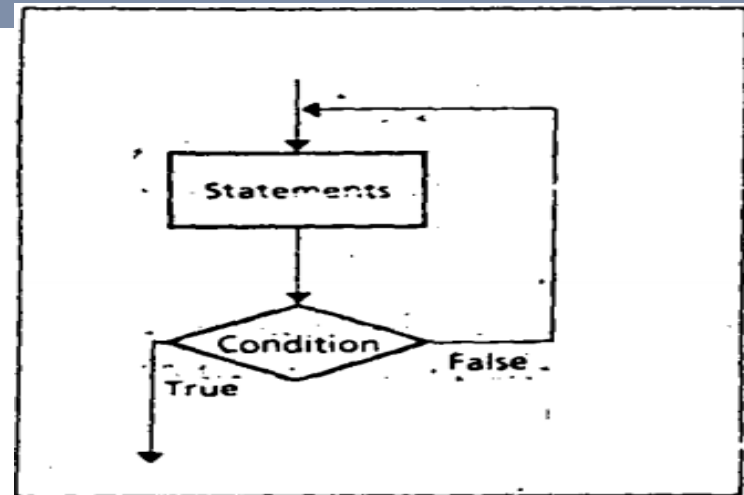
The label **WHILE\_:** is used because **WHILE** is a reserved word

# REPEAT LOOP



**REPEAT**  
statements

**UNTIL** condition



In a REPEAT...UNTIL loop, the statements are executed, and then the condition is checked.

If true, the loop terminates;

If false, control branches to the top of the loop.

# Example: REPEAT LOOP



- Write code to read characters until a blank is read.

**MOV AH,1**

**REPEAT**

**REPEAT:**

**read a character**

**INT 21H**

**UNTIL character is a BLANK**

**CMP AL,' '**

**JNE REPEAT**

# Difference between WHILE and REPEAT



Use of a WHILE loop or a REPEAT loop is a matter of **personal preference**.

The advantage of a **WHILE** is that the loop **can be bypassed** if the terminating condition is **initially false**.

Whereas the statements in a **REPEAT** **must be done at least once**.

However, the code for a REPEAT loop is likely to be a **little shorter** because there is **only a conditional jump** at the end,

But a WHILE loop has **two jumps**: a **conditional jump** at the **top** and a JMP at the **bottom**.



# References

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).
- <https://www.slideshare.net/prodipghoshjoy/flow-control-instructions-60602372>



# Books

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).
- Essentials of Computer Organization and Architecture, (Third Edition), Linda Null and Julia Lobur
- W. Stallings, “Computer Organization and Architecture: Designing for performance”, 6th Edition, Prentice Hall of India, 2003, ISBN 81 – 203 – 2962 – 7
- Computer Organization and Architecture by John P. Haynes.