Computer Organizations and Architecture

Flow Control Instructions – Part 2

Spring 24-25, CS 3205, Section D

Dr. Nazib Abdun Nasir

Assistant Professor, CS, AIUB

nazib.nasir@aiub.edu

April 30 & May 05, 2025

Outline

- Branching Structures
  - → IF-THEN
  - → IF-THEN-ELSE
  - → CASE
- Branches with Compound Conditions
  - → AND
  - → OR
- Looping Structure
  - → FOR LOOP
  - → WHILE LOOP
  - → REPEAT LOOP

Branching Structures

Branching structures enable a program to take different paths, depending on conditions.

Here, We will look at three structures.

1. IF-THEN

2. IF-THEN-ELSE

3. CASE

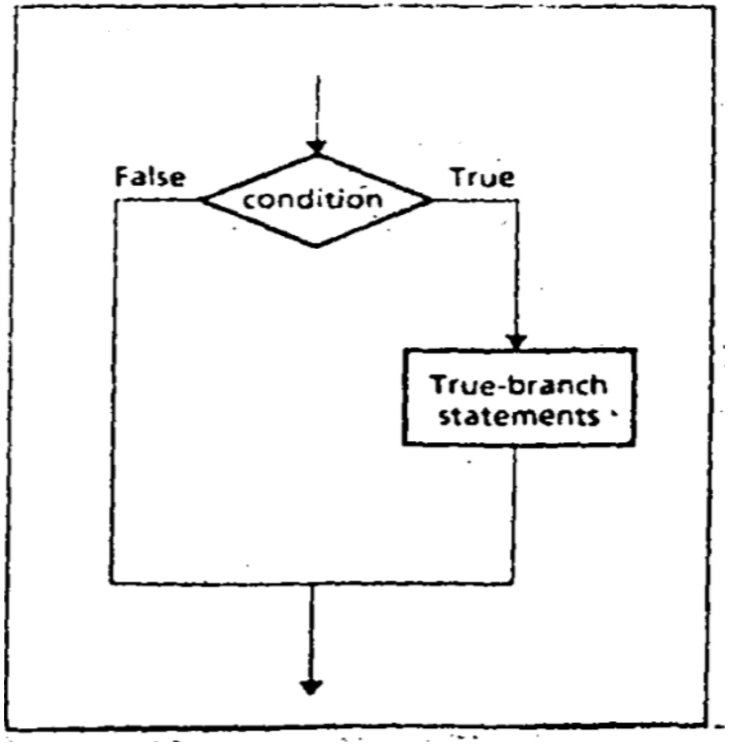
IF-THEN

IF condition is true

THEN

execute true-branch statements

END_IF



A Pseudo Code, Algorithm, and Code for IF-THEN

The condition is an expression that is either true or false.

lf It is true, the true-branch statements are executed.

lf It is false, nothing is done, and the program goes on to whatever follows.

Example: Replace a number in AX by its absolute value.

```
IF AX < 0
THEN
        replace AX by -AX
END_IF
```

```
        CMP AX, 0
        JNL END_IF
        NEG AX
END_IF:
```

IF-THEN-ELSE

IF condition is true

THEN

execute true-branch statements

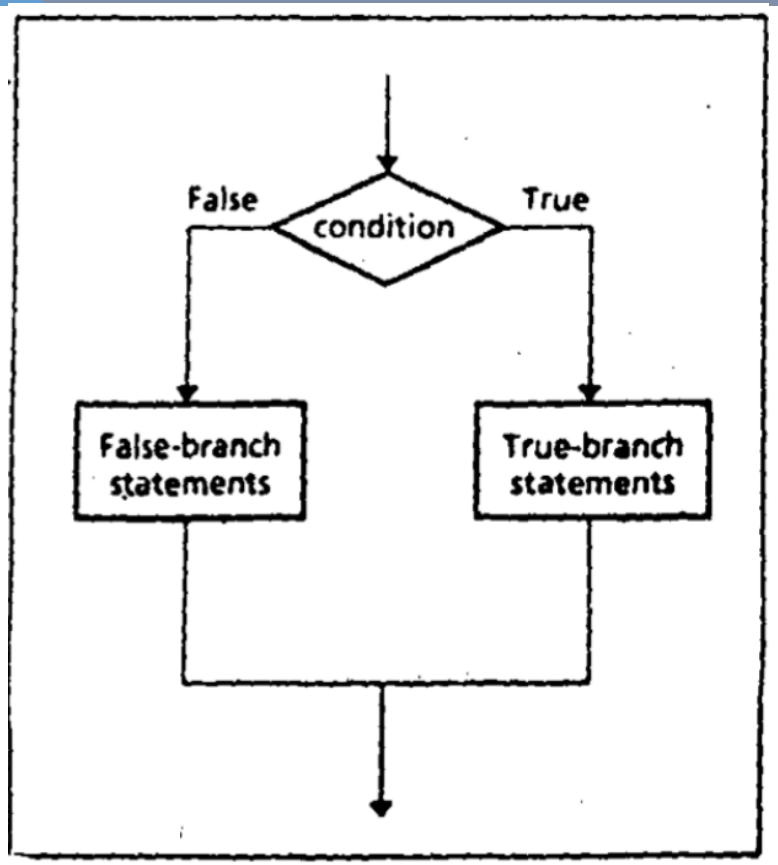
ELSE

execute false-branch statements

END_IF

You need to create statements for both conditions.

And then jump to the END from both labels as well.



A Pseudo Code, Algorithm, and Code for IF-THEN-ELSE

The condition is an expression that is either true or false.

If It is true, the true-branch statements are executed.

If It is false, then false-branch statements are executed.

Example: Suppose AL and BL contain extended ASCII characters. Display the one that comes first in the character sequence.

```
IF AL <= BL
        THEN
        Display the character in AL
        ELSE
        Display the character in BL
END IF
```

```
        MOV AH, 2
        CMP AL, BL    ; AL<=BL ?
        JNBE ELSE_IF
        MOV DL, AL
        JMP DISPLAY
ELSE_IF:
        MOV DL, BL
DISPLAY:
        INT 21H
```

A Pseudo Code, Algorithm, and Code for IF-THEN-ELSE

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE

MAIN PROC

    MOV AL,'A'
    MOV BL,'B'

    CMP AL,BL
    JBE DISPLAY_AL

    MOV AH,2
    MOV DL,BL
    INT 21H
    JMP END_PRO


  DISPLAY_AL:

    MOV AH,2
    MOV DL,AL
    INT 21H
    JMP END_PRO

  END_PRO:

    MOV AH,4CH
    INT 21H

  MAIN ENDP

END MAIN
```

A Pseudo Code, Algorithm, and Code for CASE

- A CASE is a **multi-way branch structure** that tests a register, variable, or expression for particular values or a range of values.

CASE Expression

Values_1: Statement_1

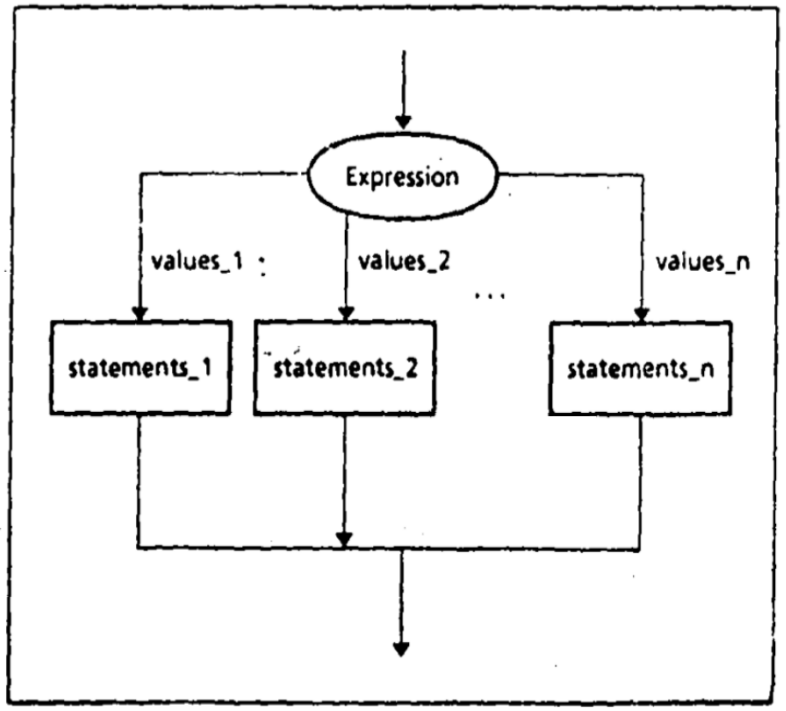
Values_2: Statement_2

…

Values_n: Statement_n

END_CASE

You MUST create labels for ALL the possible CASEs.



A Pseudo Code, Algorithm, and Code for CASE

Example: If AX contains a negative number, put -1 in BX; if AX contains 0, put 0 in BX; and if AX contains a positive number, put 1 in BX.

```
CASE AX
      <0 : put -1 in BX
      =0 : put  0 in BX
      >0 : put +1 in BX
END_CASE
```

```
CMP AX, 0
            JL NEGATIVE
            JE ZERO
            JG POSITIVE
NEGATIVE:
            MOV BX, -1
            JMP END_CASE
ZERO:
            MOV BX, 0
            JMP END_CASE
POSITIVE:
            MOV BX, 1
END_CASE:
```

Solve the Following

If AL contains 1 or 3, display “odd”.
If AL contains 2 or 4, display “even”.

CASE AL

1,3 : display ‘odd’

2,4 : display ‘even’

END_CASE

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

AND Condition

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 ('A'<= character) and (character <= 'Z')

THEN
        display character

END IF
```

Assembly Conversion

```
; read a character

MOV AH, 1

INT 21H

; IF ('A' <= char AND char <= 'Z')

CMP AL, ‘A’      ; char > ’A’

JNGE END_IF  ; exit

CMP AL, 'Z'

JNGE END_IF  ; 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 DISP
; yes, go to display it

CMP AL, 'Y' ; AL == 'Y'

JE DISP
; yes, go to display it

JMP NOPE
; diff char, terminate
```

```
DISP:

MOV AH, 2
; prepare to display

MOV CL, AL     ; get char

INT 21H        ; display it

JMP END_IF     ; and exit

NOPE:

    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.

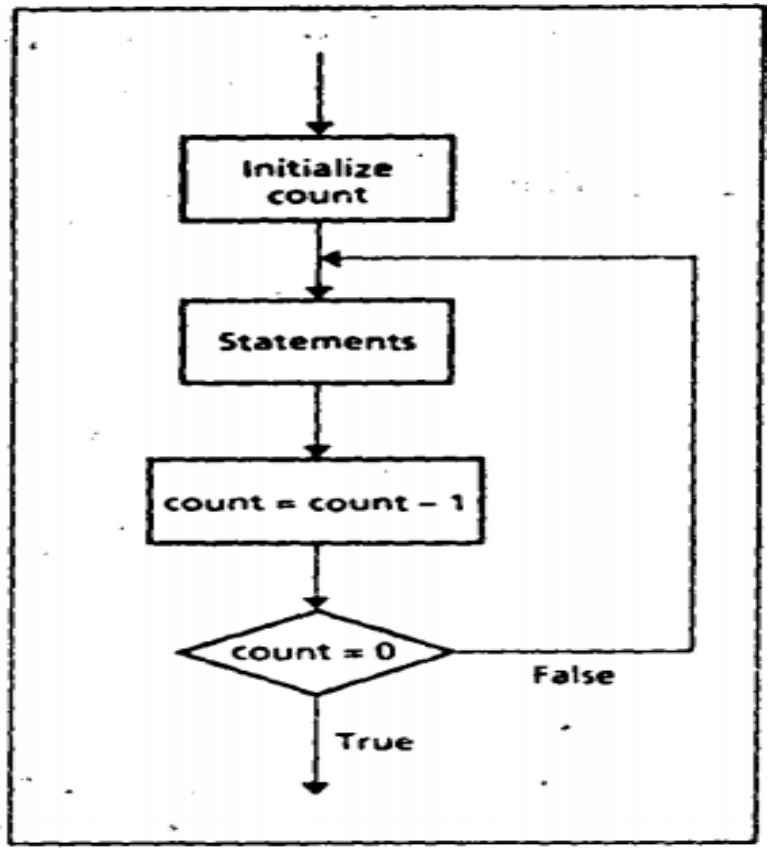
A FOR LOOP can be implemented using the LOOP instruction.

TOP:

; initialize CX to loop_count

; body of the loop

LOOP TOP



Example: FOR LOOP

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:
```

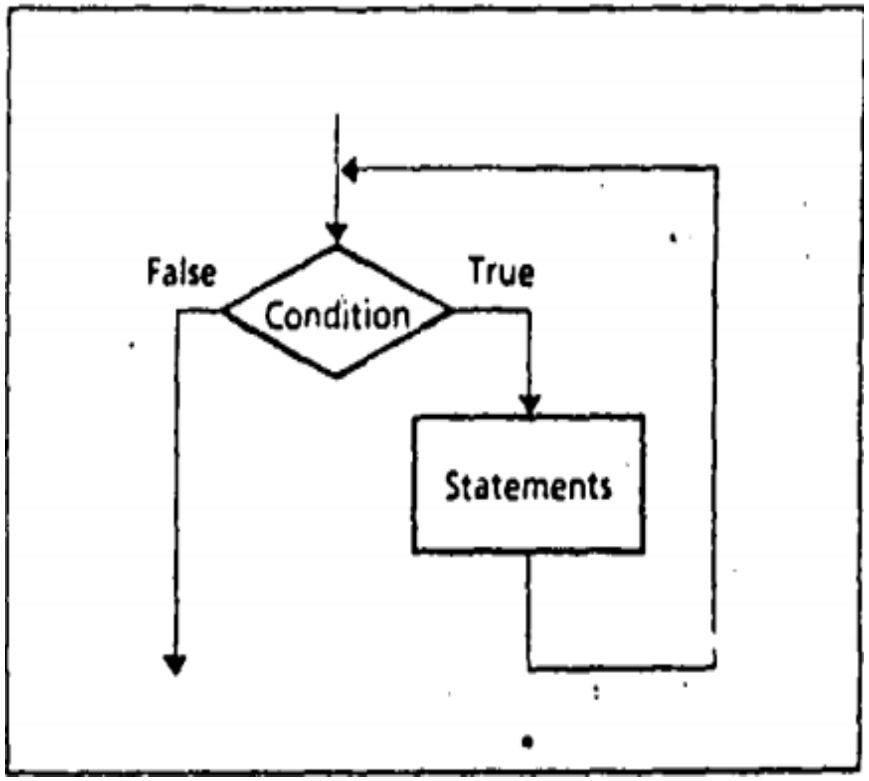
WHILE LOOP

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 ls **not executed at all**.

The loop executes as long as the condition is true.

Example: WHILE LOOP

Write a 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 CX, 0 ; char count
MOV AH, 1
INT 21H
WHILE_:
        CMP AL, 0DH       ; CR ?
        JE END_WHILE      ; yes, exit
        INC CX            ; 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, we must make sure that **any variables involved** in the condition are **initialized before the loop is entered**.

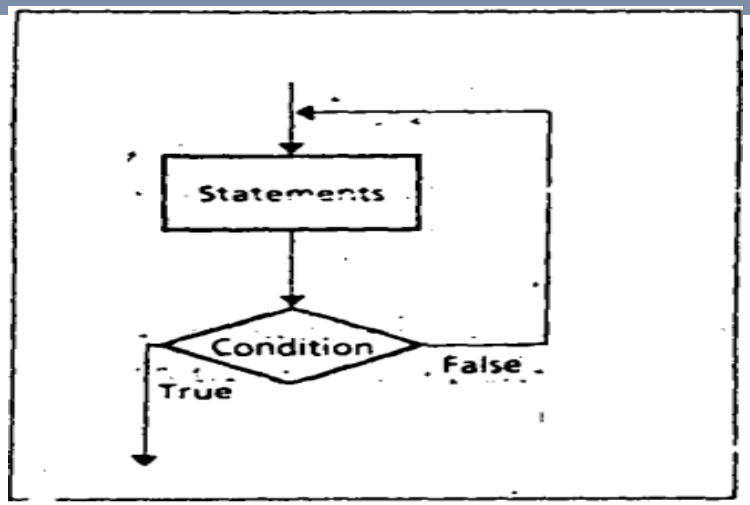
So, we read a character before entering the loop and **read another one at the bottom**.

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 a code to read characters until a blank is read.**

REPEAT

read a character

UNTIL character is a BLANK

```
MOV AH, 1

REPEAT:

INT 21H

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”, 67h Edition, Prentice Hall of India, 2003, ISBN 81 – 203 – 2962 – 7
- Computer Organization and Architecture by John P. Haynes.

References

- This is the Provided Material, modified by me.
- Chapter 6 of the Text-Book.