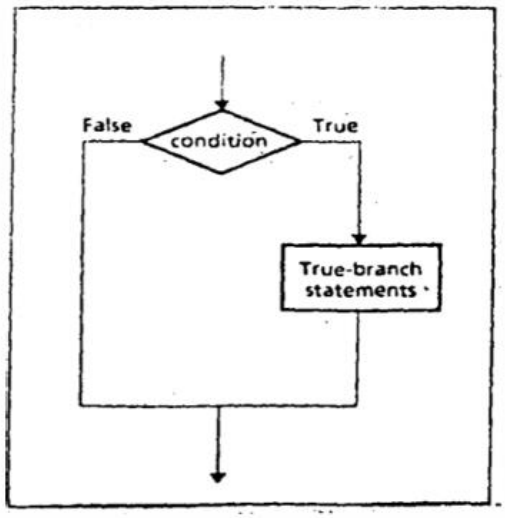
IF condition is true

THEN

execute true-branch statements

END_IF



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, 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 condition is true

THEN
execute true-branch statements

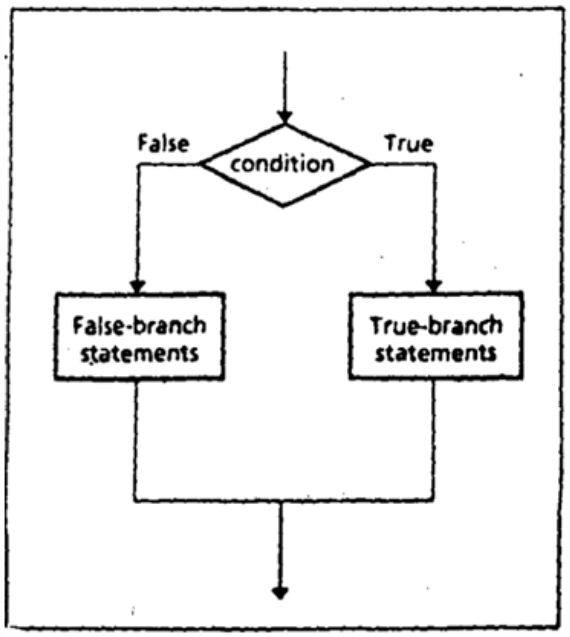
You need to create statements for both conditions.

ELSE

execute false-branch statements

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

END_IF



that comes first in the character sequence.

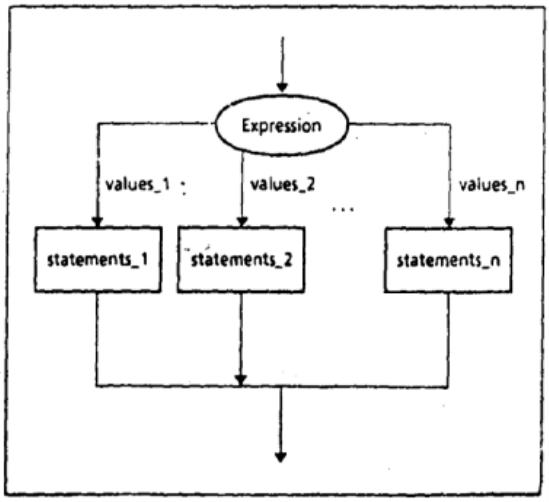
```
IF AL <= BL
        THEN
        Display the character in AL
        ELSE
        Display the character in BL
END IF
```

```
        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 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.



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:
```

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.

The control is transferred to destination_label until CX becomes 0.

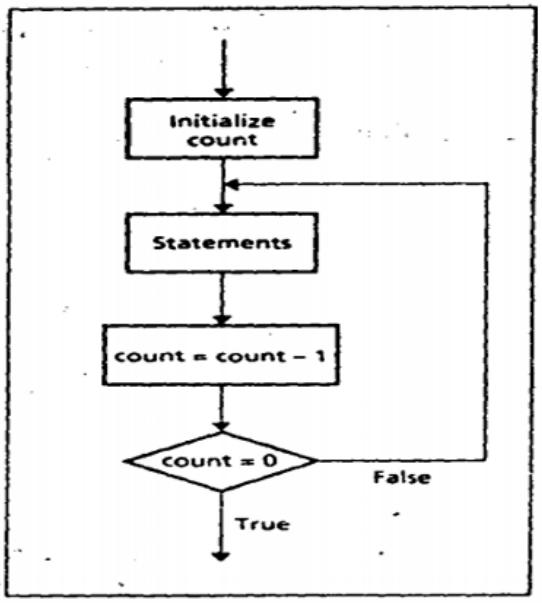
A FOR LOOP can be implemented using the LOOP instruction.

TOP:

; initialize CX to loop_count

; body of the loop

LOOP TOP

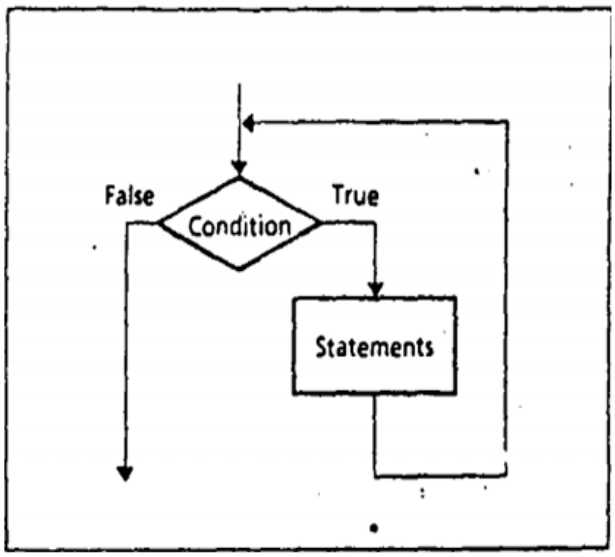


This WHILE LOOP depends on a condition.

WHILE *condition* DO

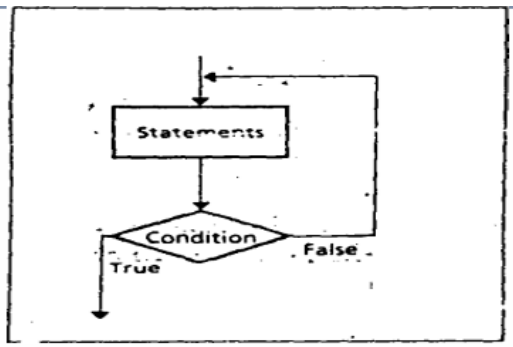
statements

END_WHILE



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.

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
```

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**.