Flow Control Instructions

Course Code: CSC 2106

Course Title: Computer Organization and Architecture

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**6.1**	**Week No:**		**Semester:**	**Fall’23**
Lecturer:	***Saeeda Sharmeen Rahman, sharmeen@aiub.edu***				

9 for Msg print

1 for input 2 for output

```
msg1 db "Enter the first capital letter: $"
msg2 db "Enter the second capital letter: $"
result db "Result: $"
newline db 10, 13, "$"
letter1 db 0
letter2 db 0

de
n proc
mov ax, @data
mov ds, ax

; Print a newline character
mov ah, 09h
lea dx, newline
int 21h

; Print the first prompt
mov ah, 09h
lea dx, msg1
int 21h
```

```
MAIN PROC
    ; INPUT
    MOV AH,1
    INT 21H
    MOV BL,AL        ; HERE BL REGISTER TAKE THE INPUT.

    ; OUTPUT
    MOV AH,2
    MOV DL,BL        ; NOW MOVE TO THE DL REGISTER.
    INT 21H
```

Lecture Outline

Decision making and repeating statement

Jump and loop instructions

Algorithm conversion to assembly language

High-Level Language Structures

Jump and Loop

Jump and Loop instructions transfers control to another program

The transfers can be unconditional or

Depends on a particular combination of status flags settings

Conversion of algorithm is easier in assembly

Example of Jump

```
.MODEL SMALL
.STACK 100
.DATA
.CODE
MAIN PROC
MOV AH,2
MOV CX,256
MOV DL,0 ;ASCII of null
PRINT_LOOP:
INT 21H
INC DL
DEC CX
JNZ PRINT_LOOP
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

JNZ (Jump if Not Zero)

JNZ is the instruction that controls loop.

If result of preceding instruction is not Zero Then the JNZ transfers the control to the instruction at label PRINT_LOOP.

If the preceding instruction contains zero (i.e. CX=0) then the program goes to execute DOS return instructions.

PRINT_LOOP is the first statement label

Labels are needed to refer another instruction

Labels end with colon (:)

Labels are placed on a line by themselves to make it stand out.

Conditional Jumps

JNZ is an example of Conditional Jump Instruction:

JXXX destination_label

if the condition for the jump is true, the next instruction to be executed is at **destination_label**

If condition is false, the instruction following the jump is done next.

i.e. for **JNZ if the preceding instruction is non-zero**

Implementation of Conditional JUMP by CPU

The CPU looks at the FLAGS register (it reflects the result of last thing that processor did)

If the conditions for the jump (combination of status flag settings) are true, the CPU adjusts the IP to point to the destination label.

The instruction at this label will be done next.

If the jump condition is false, then IP is not altered and naturally the next instruction is performed.

Example(cont'd…)

In the previous example:

The JNZ PRINT_LOOP is executed by inspecting ZF

If ZF=0 then control is transferred to PRINT_LOOP and continues

If the ZF=1 then the program goes on to execute next (i.e. MOV AH,4CH)

CMP Instruction

CMP destination , source

Compares the destination with source by computing contents

It computes by…

destination contents - source contents

The result is not stored but the FLAGS are affected

The OPERANDS of CMP may not both be Memory Locations

Destination may not be **CONSTANT**

CMP is just like SUB. However result is not stored in destination.

Signed Conditional Jumps

JG or JNLE	• Jump if Greater than • Jump if Not Less than or Equal to	ZF = 0 and SF = OF
JGE or JNL	• Jump if Greater than or Equal to • Jump if Not less than or Equal to	SF = OF
JL or JNGE	• Jump if less than • Jump if not greater than or equal	SF<>OF
JLE or JNG	• Jump if less than or Equal • Jump if not greater than	ZF = 1 or SF<> OF

Unsigned Conditional Jumps

JA or JNBE	• Jump if Above • Jump if Not Below or Equal to	ZF = 0 and CF = 0
JAE or JNB	• Jump if Above or Equal to • Jump if Not Below	CF = 0
JB or JNAE	• Jump if Below • Jump if not Above or Equal	CF = 1
JBE or JNA	• Jump if Below or Equal • Jump if Not Above	CF=1 or ZF = 1

Single-Flag Jumps

JE or JZ	• Jump if Equal • Jump if equal to Zero	ZF = 1
JNE or JNZ	• Jump if Not Equal • Jump if Not Zero	ZF = 0
JC	• Jump if Carry	CF = 1 CF = 0
JNC	• Jump if no Carry	CF=0
JO	• Jump if Overflow	CF=1 or ZF = 1
JNO	• Jump if No Overflow	OF=1
JS	Jump if Sign Negative	SF = 1
JNS	Jump if Non-Negative Sign	SF =0
JP/JPE	Jump if Parity Even	PF=1
JNP/JPO	Jump if parity Odd	PF=1

Conditional Jumps Interpretation

Signed JUMPs correspond to an analogous unsigned JUMPs (i.e. JG is equivalent to JA)

Signed jump's operates on ZF, SF and OF

Unsigned JUMP's operates on ZF and CF

Using wrong kind of JUMP can lead to wrong results.
For example: for AX= 7FFFh and BX= 8000h

CMP AX,BX

JA BELOW

Even though 7FFFh>8000h in a signed sense, the program does not jump to label BELOW. Because

In unsigned sense (JA) 7FFFh < 8000h

Working with Characters

To deal with ASCII character set, either Signed or Unsigned jumps may be used.

i.e. sign bit of a byte in a character is always zero.

However, while comparing extended ASCII characters (80h to FFh), UNSIGNED jumps should be used

JMP Instruction

JMP instruction causes an unconditional transfer of control.

JMP destination

Destination is usually a label in the same segment as the JMP itself. [ref: appendix - F]

To get around the range of restriction of a conditional jump, JMP can be used.

```
MAIN PROC

    ;Suppose AL and BL contain extended ASCII
    ;Display the one that comes first in the c

    ;unsigned jumps should be used when compar
    ;extended ASCII character  codes (80H to F


    MOV AL, 90H
    MOV BL, 82H

    CMP AL,BL
    JB PRINTAL     ; AL<BL

    MOV AH, 2
    MOV DL, BL
    INT 21H

    JMP RETURN


    PRINTAL:
    MOV AH, 2
    MOV DL, AL
    INT 21H

    RETURN:
    MOV AH, 4CH
    INT 21H
```

edit: E:\SayaStation\Assembly Language Programming\Flow Control Instruction\1.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor op

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

    ;Suppose AX and BX contain signed numbers.
    ;Write  some code to put the biggest one in CX

    MOV AX, 3
    MOV BX, 5

    CMP AX,BX
    JG Label1           ; AX < BX
    MOV CX,BX
    JMP RETURN ;unconditional jump

    Label1:
    MOV CX,AX


    RETURN:
    MOV AH, 4CH
    INT 21H








MAIN ENDP
```

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

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

    MOV AX, -1

    CMP AX,0
    JL NEGATIVE    ; AX<0
    JE ZERO        ; AX=0
    JG POSITIVE    ; AX>0

    NEGATIVE:
    MOV BX,-1
    JMP RETURN

    ZERO:
    MOV BX,0
    JMP RETURN

    POSITIVE:
    MOV BX,1


    RETURN:
    MOV AH,4CH
    INT 21H
```

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

.CODE
MAIN PROC

    ;Read a character and if it is an uppercase letter
    ;display it.

    MOV AH,1
    INT 21H

    CMP AL,'A'
    JNGE RETURN

    CMP AL,'Z'
    JNLE RETURN

    MOV AH, 2
    MOV DL, AL
    INT 21H

    RETURN:
    MOV AH,4CH
    INT 21H





MAIN ENDP
```

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

    ;Read a character and if it is 'y' or 'Y' display it
    ;otherwise terminate the program.

    MOV AH,1
    INT 21H

    CMP AL, 'Y'
    JE DISPLAY

    CMP AL, 'y'
    JE DISPLAY

    JMP RETURN


    DISPLAY:
    MOV AH,2
    MOV DL,AL
    INT 21H


    RETURN:
    MOV AH,4CH
```

JMP Vs JNZ

```
TOP:
        DEC CX
        JNZ TOP
        MOV AX,BX
```

```
TOP:
        DEC CX
        JNZ BOTTOM  ; keep looping till CX>0
        JMP EXIT
BOTTOM:
        JMP TOP
EXIT:
        MOV AX,BX
```

High-Level Language Structures

Jump can be used to implement branches and loops

As the Jump is so primitive, it is difficult to code an algorithm with jumps without some guidelines.

The High-level languages (conditional IF-ELSE and While loops) can be simulated in assembly.

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

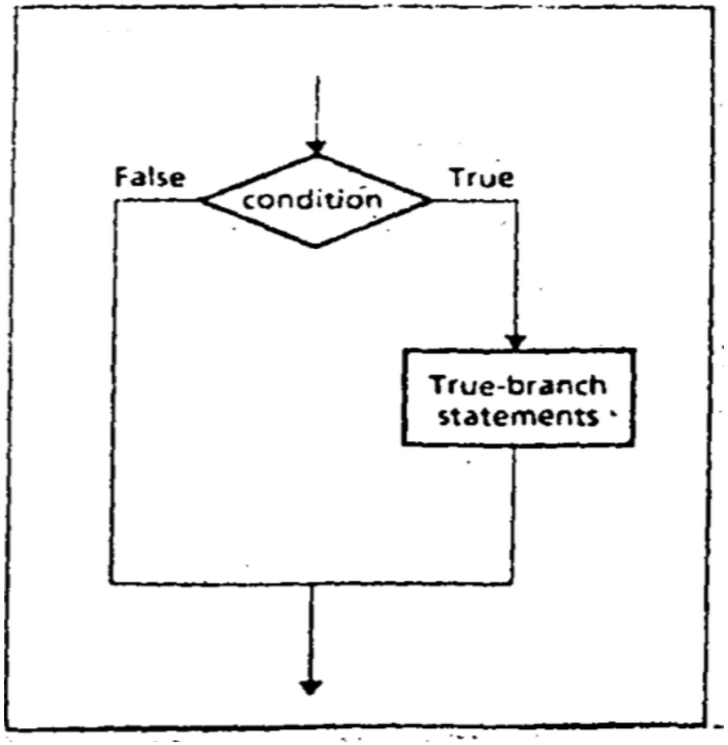
IF-THEN

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.

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: to 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

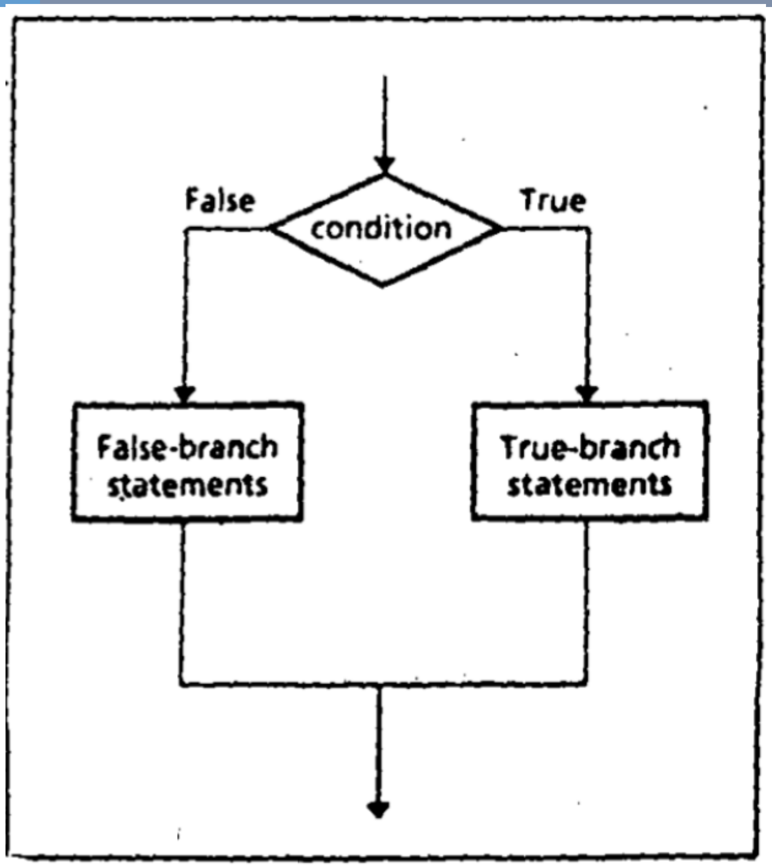
IF condition is true

THEN
execute true-branch statements

ELSE

execute false-branch statements

END_IF



A Pseudo Code and 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_
        MOV DL,AL
        JMP DISPLAY
ELSE_:
        MOV DL,BL
DISPLAY:
        INT 2lh
```

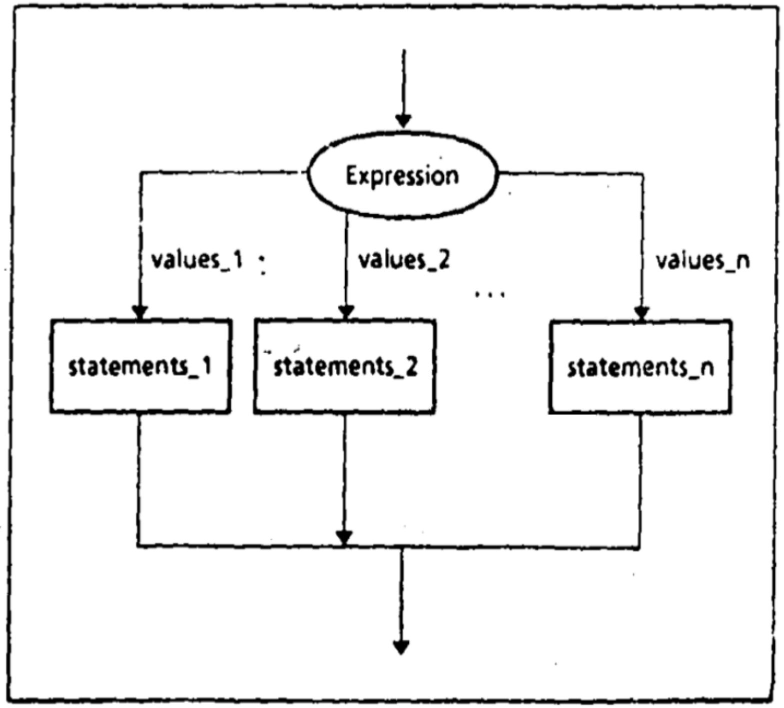
CASE

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



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 +l in BX
END_CASE
```

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

Solve the Following

If AL contains 1 or 3, display "o"; if AL contains 2 or 4, display "e".

- CASE AL

  1,3: display ‘o’

  2,4: display ‘e’

END_CASE

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.