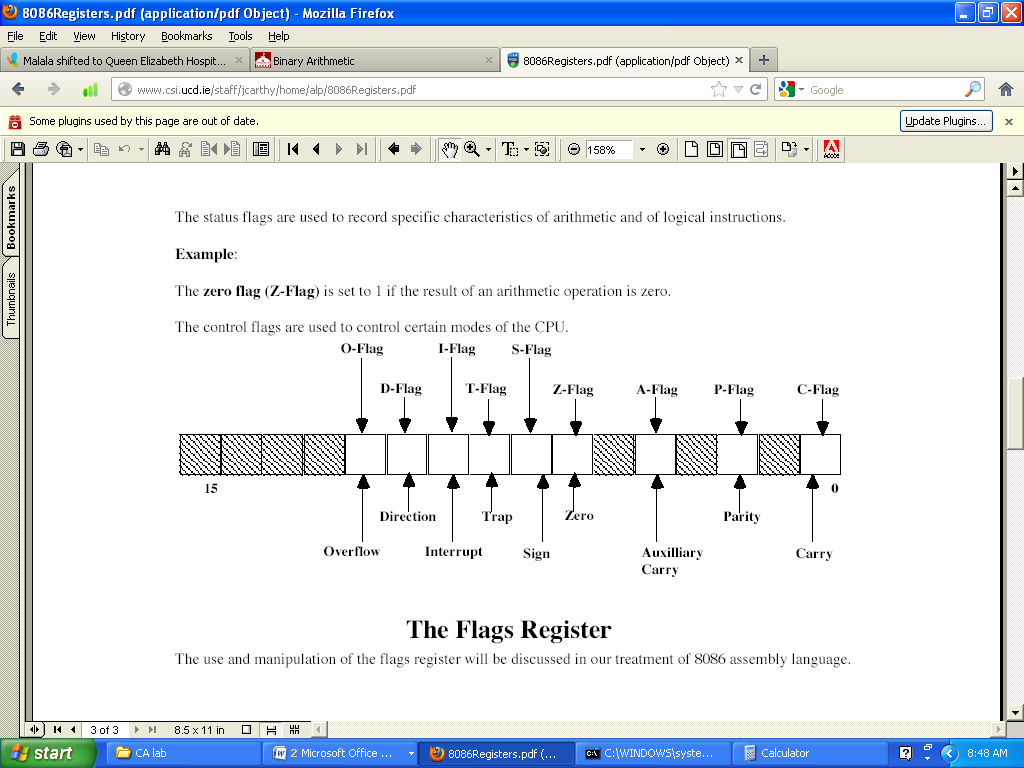
**Lab 03: Program Flow Control Instructions**

**OBJECTIVE**

To learn to change the sequence of execution of a program by using program flow control instructions.

**A. Flag Registers:**

The flag register is a special 16-bit register with individual bit positions assigned to show the status of CPU or the result of arithmetic operations. Each relevant bit position is given a name other positions are undefined:



**Note:** The Status of these flags can be used to make a decision to change the program flow control. The decision can be either TRUE of FALSE. Based on this decision the program executes/jumps to a certain line of code.

**B. Program Flow Control**

Controlling the program flow is a very important thing, this is where your program can make decisions according to certain conditions.

1. Unconditional Jumps
2. Conditional Jumps

**1. Unconditional Jumps:**

The basic instruction that transfers control to another point in the program is JMP.

The basic syntax of JMP instruction:

Syntax: **JMP** label

To declare a label in your program, just type its name and add ":" to the end, label can be any character combination but it cannot start with a number, for example here are 3 legal label definitions:

label1:

label2:

a:

Label can be declared on a separate line or before any other instruction, for example:

x1:

MOV AX, 1

x2: MOV AX, 2

**Here is an example of JMP instruction:**

main proc

mov ax,5 ;set AX to 5

mov bx,2 ;set BX to 2

jmp calc ;go to ‘Calc’

back:

jmp stop ;go to ‘Stop’

calc:

add ax,bx ;Add BX to AX

jmp back ;go to ‘Back’

stop:

main endp

**Another example:**

main proc

jmp label1 ;go to 'label1'

mov ah,2

mov dl,'k'

int 21h

label1:

mov ah,2

mov dl,'p'

int 21h

main endp

**2. Short Conditional Jumps**

Unlike JMP instruction that does an unconditional jump, there are instructions that do a conditional jump (jump only when some conditions are in act). These instructions are divided in three groups, first group just test single flag, second compares numbers as signed, and third compares numbers as unsigned.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Operands** | **Description** |
| **CMP** | REG, REG  REG, immediate  REG, memory memory, REG memory, immediate | Compare.  Algorithm: operand1 - operand2  result is not stored anywhere, only flags affected.  If  operand1 - operand2 > 0 ZF=0, Sign=0  operand1 - operand2 < 0 ZF=0, Sign=1 (**JS**)  operand1 - operand2 = 0 ZF=1 (**JZ**) **Example:**  MOV AL, 5  MOV BL, 5  CMP AL, BL ; ZF = 1 (so equal)  JZ label   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| **JE** | Label | Short Jump if first operand is Equal to second operand (as set by CMP instruction). Signed/Unsigned.  Algorithm:  if ZF = 1 then jump  **Example:**  include 'emu8086.inc'  main proc  MOV AL, 5  CMP AL, 5  JE label1  PRINT 'AL is not equal to 5.'  JMP exit  label1:  PRINT 'AL is equal to 5.'  exit:     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |
| **JG**  (same as **JA**) | Label | Short Jump if first operand is Greater than second operand (as set by CMP instruction). Signed.  Algorithm:  if (ZF = 0) and (SF = OF) then jump  **Example:**  include 'emu8086.inc'  main proc  MOV AL, 5  CMP AL, -5  JG label1  PRINT 'AL is not greater -5.'  JMP exit  label1:  PRINT 'AL is greater -5.'  exit:     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |
| **JL**  (same as **JB**) | Label | Short Jump if first operand is Less than second operand (as set by CMP instruction). Signed.  Algorithm:  if SF = 1 then jump  **Example:**  include 'emu8086.inc'    MOV AL, 2  CMP AL, 5  JL label1  PRINT 'AL > 5'  JMP exit  label1:  PRINT 'AL < 5'  exit:     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |

**More Conditional Jumps:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| JA | JAE | JB | JBE | JC | JL | JLE | JNA | JNAE |
| JNBE | JNC | JNG | JNGE | JNL | JNLE | JNO | JNP | JNS |
| JO | JP | JPE | JPO | JS | JNB | JNZ |  |  |

**Code-01:** Write a program to find if the number is Even or Odd? (**JE**)

**Hint:** Divide the number by 2, if the remainder is 0, its Even, else Odd.

include ‘emu8086.inc’

.model small

.code

main proc

mov ah,1 ;input the number to be checked

int 21h

mov bl,2 ; divisor/source moved to bl

div bl

cmp ah,0 ; remainder goes to AH, check it

**JE** even ; remainder zero is even

JNE odd ; remainder zero is odd

even:

print ‘Even No’

jmp exit

odd:

print ‘Odd No’

exit:

main endp

end main

**Code-02:** Count the entered characters before user presses space bar. (**JE**)

**Hint:** We use ‘cmp’ command and ‘increment’ counter if no space-bar.

.model small

.stack 100h

.data

.code

main proc

mov cl,0 ; set counter to zero

label1:

mov ah,1

int 21h

cmp al,' ' ; this is single space showing ASCII code for space bar

**JE** label2 ; if spacebar is found jump to label2

inc cl ; otherwise increment counter

jmp label1 ; again another input is taken

label2:

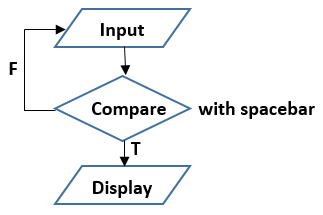
mov ah,2 ;display the count

mov dl,cl

**Program Flow Control:**

add dl,30h ; ASCII correction

int 21h

****

main endp

end main

**Code-03:** Compare two numbers and tell which one is bigger. (**JG** or **JA**)

**Hint:** Jump to a label if JG is TRUE and display the bigger number.

include ‘emu8086.inc’

.model small

.code

main proc

mov ah,1 ; input first number

int 21h

mov bl,al ; this number is saved to ‘bl’

mov ah,1 ; input second number

int 21h

CMP bl,al ; compare the first number with second

**JG** label1 ; if bl > al, goto label1

Print ‘Second no is bigger’

JMP exit

Label1:

Print ‘First no is bigger’

Exit:

Main endp

End main

**Code-04:** Write a code to find the smallest of the three numbers. (**JL** / **JB**)

**Hint:** First comparison: Value1 (V1) with Value2 (V2)

Second comparison: Any one of the smaller compared with (V3)

Include ‘emu8086.inc’

.model small

.data

v1 db ?

v2 db ?

v3 db ?

.code

main proc

mov ah,1 ; input first number v1

int 21h

mov v1,al

mov ah,1 ; input second number v2

int 21h

mov v2,al

mov ah,1 ; input third number v3

int 21h

mov v3,al

; finding the smallest number

Mov al,v1

Cmp al,v2 ; compare v1 with v2

**JL** label1 ; goto label1 if v1 < v2

Mov al,v2 ; else move v2 to AL

CMP al,v3 ; compare v2 with v3

**JL** label2 ; goto label2 if v2 < v3

Mov al,v3 ; else move v3 to AL (v3 = smallest)

JMP end

Label1:

CMP al,v3 ; compare v1 with v3

**JL** end ; goto end if v1 < v3 (v1 = smallest)

Mov al,v3 ; else (v3 = smallest)

Jmp end

Label2:

Mov al,v2 ; (v2 = smallest)

; display the smallest number

End:

Mov ah,2

Mov dl,al

Int 21h

Main endp

End main

**C. LOOPS:**

The loop instruction is the easiest way to repeat a block of statements a specific number of times. CX is automatically used as a counter and is decremented each time the loop repeats.

**Syntax** is: **loop** *destination*.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Operands** | **Description** |
| **LOOP** | Label | Decrease CX, jump to label if CX not zero. If CX = 0, loop ends. Algorithm:   * CX = CX - 1 * if CX <> 0 then   + jump   else   * + no jump, continue   **Example:**  include 'emu8086.inc'  main proc  MOV CX, 5  label1:  PRINT 'loop!'  LOOP label1  Main endp   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | unchanged | | | | | | |

**More Loops:**

|  |  |  |  |
| --- | --- | --- | --- |
| LOOPE | LOOPNE | LOOPNZ | LOOPZ |

**LOOPE:** Loop if Equal.

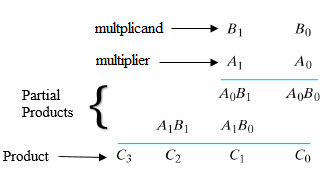
**LOOPNE:** Loop if Not Equal.

**LOOPZ:** Loop if Zero.

**LOOPNZ:** Loop if Not Zero.

**Code-05:** Write a program to multiply two positive numbers by a ‘repeated addition’ method. For example, to multiply 5**ⅹ**3, the program evaluates the product by adding 5 three times, or 5 + 5 + 5.

**Hint:** We move ‘multiplier’ to CX register to **loop** ‘Addition’ CX times.



.model small

.stack 100h

.data

.code

main proc

mov bl,5 ;multiplicand value moved to BL

mov cx,3 ;multiplier value moved to CX to loop CX times

mov al,0 ;initialize AL register to zero to perform addition

label1:

add al,bl ;BL will add to AL, CX times to produce multiplier result

**LOOP** label1

;Now we divie AL by 10 to display two digit No's separately

mov bh,10

div bh

mov bl,ah ;remainder moved to BL

mov ah,2

mov dl,al ;display quotient

add dl,30h

int 21h

mov ah,2

mov dl,bl ;display remainder

add dl,30h

int 21h

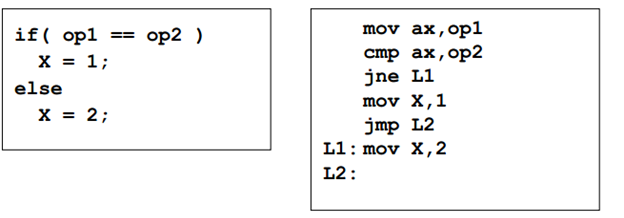
main endp

end main

**THE END**

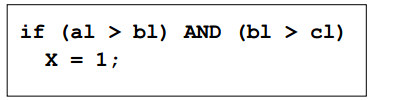
**Block-Structured IF Statements:**

* Assembly language programmers can easily translate logical statements written in C++/Java into assembly language. For example:

****

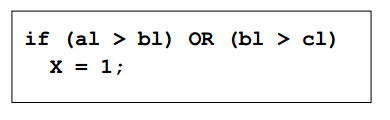
**Compound Expression with AND:**

* When implementing the logical AND operator, consider that HLLs use short-circuit evaluation
* In the following example, if the first expression is false, the second expression is skipped:

****

**Compound Expression with OR :**

* When implementing the logical OR operator, consider that HLLs use short-circuit evaluation
* In the following example, if the first expression is true, the second expression is skipped:

****

**For Loop Instruction:**

* Cx used as Counter
* Automatically Decremented after every iteration
* Label is used to jump
* Ends when counter reaches zero
* Example:

**Mov cx,10 ;loop runs 10 times**

**Loop1:**

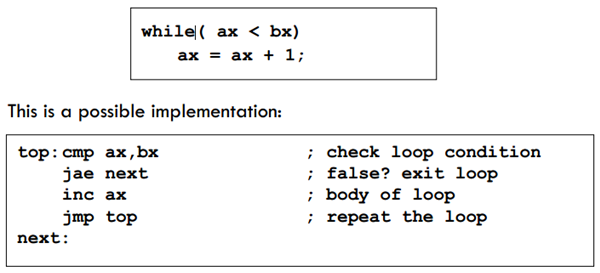
**…**

**Loop Loop1**

**WHILE Loops:**

A WHILE loop is really an IF statement followed by the body of the loop, followed by an unconditional jump to the top of the loop. Consider the following example:

**Example:**

****

**Activty:**

* Write an Assembly program to print all the Odd numbers from 1 to 100 using loops and Compare instructions.
* Take two inputs from user and display the largest one.
* Take two inputs from user lets say x and y and implement the given c++ code.

if((x <= 4) && (y<=9))

{

For(int i=0; I < x; i++)

{

Cout<< x;

}

}

Else

{

Cout<<“0”;

}