ASSIGNMENT 10 LAB 10

LOOPZ and LOOPE Instruction:

It is Useful when scanning an array for the first element that does not match a given value.

Syntax:

LOOPE destination
LOOPZ destination

Logic:

- ECX ECX – 1
- if ECX > 0 and ZF=1, jump to destination

LOOPNZ and LOOPNE Instruction:

It is Useful when scanning an array for the first element that matches a given value.

Syntax:

LOOPNZ destination
LOOPNE destination

Logic:

- ECX ECX – 1
- if ECX > 0 and ZF=0, jump to destination

Example LOOPNZ:

The following code finds the first positive value in an array:

CODE:

```
INCLUDE Irvine32.inc

.data
array SWORD -3, -6, -1, -10, 10, 30, 40, 4 ; Array of signed words

.code
main PROC
   mov esi, OFFSET array      ; Point ESI to the start of the array     mov
ecx, LENGTHOF array     ; Set ECX to the number of elements

next:
   mov ax, WORD PTR [esi]      ; Load the current array value into AX
test ax, 8000h            ; Check if the sign bit is set (negative value)     jz
found                ; If ZF = 1 (positive value), jump to "found"     add esi,
TYPE array        ; Move to the next element (16-bit step)
   loopnz next              ; Continue if ECX > 0     jmp quit
; Exit if no positive value is found found:
   movsx eax, ax             ; Sign-extend AX to EAX (32-bit) for WriteInt
```

NAME: WALEED GHANI **ID: 012-0082**

```
   call WriteInt            ; Display the positive value     call
Crlf              ; Print a new line

quit:
   exit                  ; Exit the program
main ENDP

END main
```

3. Block -Structured IF Statements Structured IF Statements:

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

C++/Java CODE:

```
if( op1 == op2 )
X = 1; else

X = 2;
```

Assembly CODE:

```
INCLUDE Irvine32.inc
.data
op1 DWORD 5        ; Example value for op1 op2
DWORD 10       ; Example value for op2
X   DWORD ?        ; Variable X to store the result (initialized later)
.code
main PROC
   mov eax, op1    ; Load op1 into eax
cmp eax, op2    ; Compare eax with op2
 jne L1         ; Jump to L1 if op1 != op2
mov X, 1       ; If op1 == op2, set X to 1
jmp L2         ; Jump to L2 to skip setting X to 2
L1:
   mov X, 2       ; If op1 != op2, set X to 2
L2:
   ; Exit the program
   exit
main ENDP END

main
```

ASSIGNMENT 10 LAB 10

Task Example: Implement the following pseudo code in assembly language. All values are unsigned:

CODE:

```
if( ebx <= ecx )
{ eax =
5; edx =
6; }
```

Solution:

```
INCLUDE Irvine32.inc  ; Include the Irvine library for input/output functions

.data
X DWORD 0  ; Variable X, initialized to 0 (for testing)

.code
main PROC
   ; Compare EBX with ECX
   cmp ebx, ecx       ; Compare EBX with ECX

   ; Jump if EBX <= ECX (JBE = Jump if Below or Equal)     jbe
condition_true  ; If EBX <= ECX, jump to condition_true

   ; Skip the code block (if EBX > ECX, do nothing)     jmp
done

condition_true:
   mov eax, 5         ; Set EAX to 5

   mov edx, 6         ; Set EDX to 6

done:
   ; Exit the program
   exit
main ENDP

END main
```

Implement the following pseudo code in assembly language. All values are 32-bit signed integers:

```
if( var1 <= var2 ) var3 = 10; else
 { var3 =
6;
var4 = 7;
```

NAME: WALEED GHANI ID: 012-0082

ASSIGNMENT 10 LAB 10

}

4. Compound Expression with AND [1/3] Compound Expression with AND [1/3]:

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:

Expression:

```
if (al > bl) AND (bl > cl)

X = 1;
```

5. Compound Expression with AND [2/3] Compound Expression with AND [2/3]:

Expression:

```
if (al > bl) AND (bl > cl) X
= 1;
```

This is one possible implementation . . .

```
cmp al,bl                ; first expression...

ja L1

jmp next

L1:

cmp bl,cl                ; second expression...

ja L2

jmp next

L2:                      ; both are true

mov X,1                  ; set X to 1

next:
```

6. Compound Expression with AND [3/3]:

Expression:

```
if (al > bl) AND (bl > cl) X
= 1;
```

ASSIGNMENT 10 LAB 10

But the following implementation uses 29% less code by reversing the first relational operator.

```
cmp al,bl                ; first expression...
jbe next                 ; quit if false
cmp bl,cl                ;                second
                         expression...
jbe next                 ; quit if false
mov X,1                  ; both are true
next:
```

Implement the following pseudo code in assembly language. All values are unsigned:

```
if(ebx <= ecx) AND (ecx > edx )
{ eax =
5; edx
= 6; }
```

Solution:

```
INCLUDE Irvine32.inc  ; Include the Irvine library for input/output

.data    prompt1 BYTE "EAX
= ", 0    prompt2 BYTE "EDX
= ", 0

.code
main PROC
   ; Set initial values    mov ebx,
10        ; Set EBX to 10    mov ecx,
20        ; Set ECX to 20    mov edx,
15        ; Set EDX to 15

   ; Compare EBX and ECX (first condition: EBX <= ECX)
cmp ebx, ecx        ; Compare EBX with ECX (unsigned)
jbe first_condition_true ; Jump if EBX <= ECX (unsigned)
jmp done            ; If condition fails, jump to done

first_condition_true:
   ; Now compare ECX and EDX (second condition: ECX > EDX)
cmp ecx, edx        ; Compare ECX with EDX (unsigned)    ja
second_condition_true ; Jump if ECX > EDX (unsigned)    jmp
done            ; If second condition fails, jump to done

second_condition_true:
   ; Both conditions are true, set EAX and EDX
mov eax, 5          ; Set EAX to 5    mov edx, 6
; Set EDX to 6

done:
```

NAME: WALEED GHANI ID: 012-0082

ASSIGNMENT 10 LAB 10

```
  ; Print EAX and EDX values    mov edx,
offset prompt1    call WriteString    ;
Print "EAX = "    mov eax, eax       ; Move
EAX to print it    call WriteInt      ; Print
EAX value    call Crlf          ; New line

  mov edx, offset prompt2    call
WriteString    ; Print "EDX = "    mov
eax, edx       ; Move EDX to print it    call
WriteInt       ; Print EDX value    call Crlf
; New line

  ; Exit the program

  exit
main ENDP

END main
```

7. Compound Expression with OR [1/2]:

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:

Expression:

```
if (al > bl) OR (bl > cl)
X = 1;
```

8. Compound Expression with OR [2/2]:

Expression:

```
if (al > bl) OR (bl > cl)
X = 1;
```

We can use "fall-through" logic to keep the code as short as possible:

```
cmp al,bl          ; is AL > BL?

ja L1              ; yes

cmp bl,cl          ; no: is BL > CL?

jbe next           ; no: skip next statement

L1: mov X,1        ; set X to 1

next:
```

9. WHILE Loop:

ASSIGNMENT 10 LAB 10

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:

while(eax < ebx)
eax = eax + 1;

This is a possible implementation:

```
top:cmp eax,ebx          ; check loop condition
jae next                 ; false? exit loop
inc eax                  ; body of loop
jmp top                  ; repeat the loop
next:
```

Task Example: Implement the following loop, using unsigned 32-bit integers:

while(ebx <= val1)
{ ebx = ebx +
5; val1 = val1
- 1

}

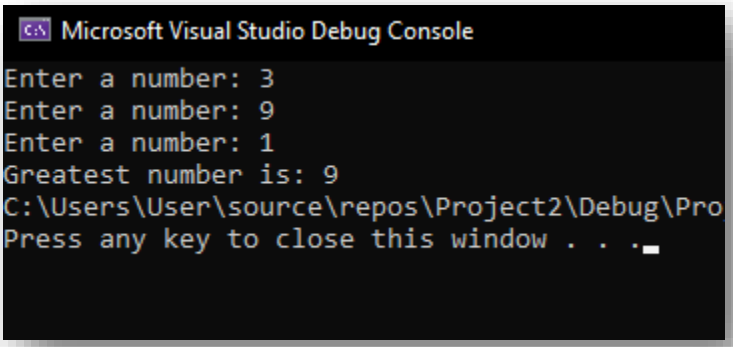
Lab 10- Exercise:

1. Write an Assembly Language program to find the largest of three numbers from the user input.

```
INCLUDE Irvine32.inc

.DATA
 num1 DWORD ?
 num2 DWORD ?
 num3 DWORD ?
 res DWORD ?

 msg1 BYTE "Enter a number: ", 0
 msg2 BYTE "Greatest number is: ", 0

.CODE
main PROC
   mov edx, offset msg1
      call writeString
      call readDec
      mov num1, eax

   mov edx, offset msg1
      call writeString
```

ASSIGNMENT 10 LAB 10

```
    call readDec
    mov num2, eax

  mov edx, offset msg1
    call writeString
    call readDec
    mov num3, eax

    start:
    mov eax, num1
    cmp eax, num2
    jge second
    mov eax, num2

    second:
    cmp eax, num3
    jge result
    mov eax, num3

    result:
    mov edx, offset msg2
    call writeString
    call writeDec

  exit
main ENDP
END main
```



2. Write an Assembly Language program to accept a coordinate point in a XY coordinate system and determine in which quadrant the coordinate point lies.

```
INCLUDE Irvine32.inc
.data
  promptX BYTE "Enter the X-coordinate: ", 0
  promptY BYTE "Enter the Y-coordinate: ", 0
  quad1Msg BYTE "The point lies in Quadrant 1.", 0
  quad2Msg BYTE "The point lies in Quadrant 2.", 0
  quad3Msg BYTE "The point lies in Quadrant 3.", 0
  quad4Msg BYTE "The point lies in Quadrant 4.", 0
  axisMsg  BYTE "The point lies on an axis.", 0
  x DWORD ?
  y DWORD ?

.code
main PROC
  ; Prompt for X-coordinate
  mov edx, OFFSET promptX
  call WriteString
```

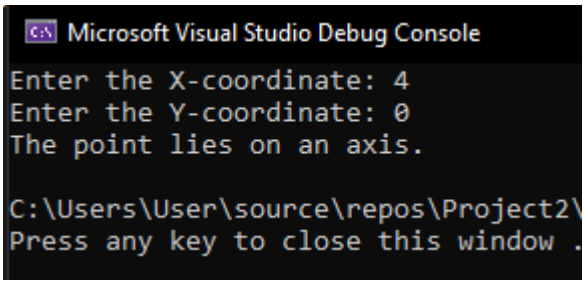
ASSIGNMENT 10 LAB 10

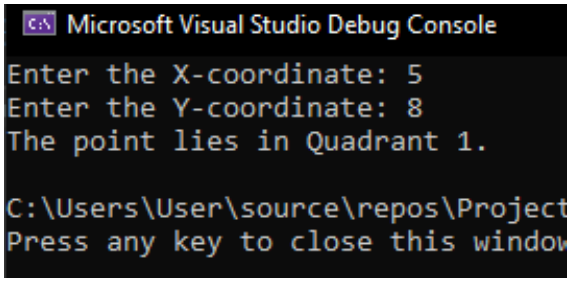
```
    call ReadInt
    mov x, eax      ; Store X-coordinate in x
    ; Prompt for Y-coordinate
    mov edx, OFFSET promptY
    call WriteString
    call ReadInt
    mov y, eax      ; Store Y-coordinate in y

    ; Determine the quadrant
    mov eax, x      ; Load X-coordinate
    cmp eax, 0      ; Compare X with 0
    je checkAxis    ; If X == 0, check Y-axis

    mov ebx, y      ; Load Y-coordinate
    cmp ebx, 0      ; Compare Y with 0
    je checkAxis    ; If Y == 0, check X-axis
    ; Check for Quadrant 1
    cmp eax, 0
    jg checkQ1Q4    ; If X > 0, check Q1 or Q4
    ; Check for Quadrant 2 or 3
    jl checkQ2Q3    ; If X < 0, check Q2 or Q3

checkQ1Q4:
    cmp ebx, 0
    jg displayQ1    ; If Y > 0, it's Q1
    jl displayQ4    ; If Y < 0, it's Q4

checkQ2Q3:
    cmp ebx, 0
    jg displayQ2    ; If Y > 0, it's Q2
    jl displayQ3    ; If Y < 0, it's Q3

checkAxis:
    mov edx, OFFSET axisMsg
    call WriteString
    call Crlf
    jmp done

displayQ1:
    mov edx, OFFSET quad1Msg
    call WriteString
    call Crlf
    jmp done

displayQ2:
    mov edx, OFFSET quad2Msg
    call WriteString
```

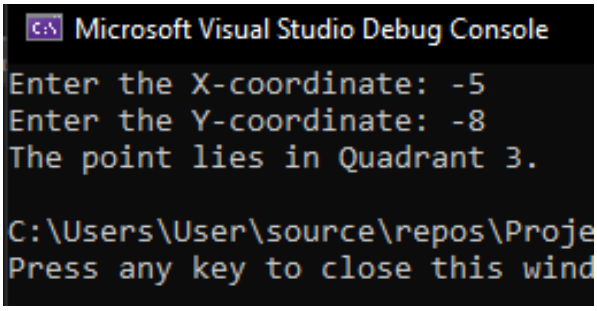
NAME: WALEED GHANI **ID: 012-0082**

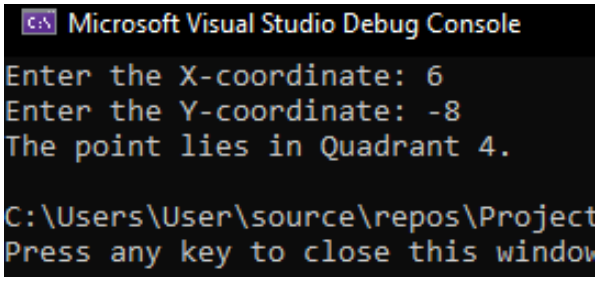
ASSIGNMENT 10 LAB 10

```
    call Crlf
    jmp done

displayQ3:
    mov edx, OFFSET quad3Msg
    call WriteString
    call Crlf
    jmp done

displayQ4:
    mov edx, OFFSET quad4Msg
    call WriteString
    call Crlf
    jmp done
done:
    exit
main ENDP

END main
```









3. Write an Assembly Language program to read temperature in centigrade and display a suitable message according to temperature state below :

Temp < 0 then Freezing weather
Temp 0-10 then Very Cold weather
Temp 10-20 then Cold weather
Temp 20-30 then Normal in Temp
Temp 30-40 then Its Hot
Temp >=40 then Its Very Hot
Test Data :
42
***Expected Output* :**
It's very hot.

ASSIGNMENT 10 LAB 10

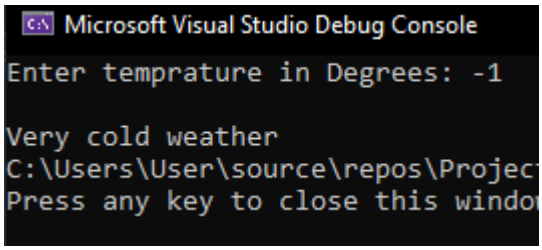
```
INCLUDE Irvine32.inc

.data
   msg BYTE "Enter
temprature in Degrees:
", 0
   Tempfreeze BYTE
"Freezing weather", 0
   TempvCold BYTE
"Very cold weather", 0
   Tempcold BYTE "Cold
weather", 0
   Tempnormal BYTE
"Normal in Temp", 0
   Temphot BYTE "It's
hot", 0
   TempvHot BYTE "It's
very hot", 0


.code
main PROC
   mov edx, offset msg
   call writeString
   call ReadDec

   L1:
   cmp eax, 0
   jl Freeze
   jmp L2

   L2:
   cmp eax, 10
   jle Verycold
   jg L3

   L3:
   cmp eax, 20
   jle Cold
   jg L4

   L4:
   cmp eax, 30
   jle Normal
   jg L5

   L5:
```

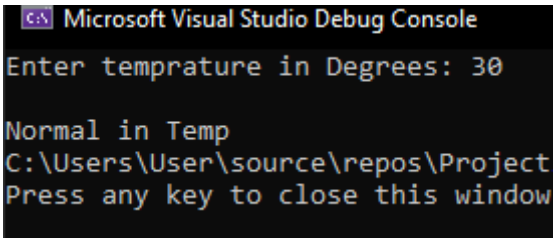
ASSIGNMENT 10 LAB 10

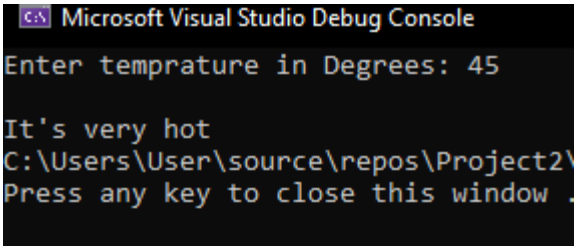
```
    cmp eax, 40
    jle Hot
    jg Veryhot

Freeze:
    mov edx, offset
Tempfreeze
    call crlf
    call writeString
    jmp done
Verycold:
    mov edx, offset
TempvCold
    call crlf
    call writeString
    jmp done
Cold:
    mov edx, offset
Tempcold
    call crlf
    call writeString
    jmp done
Normal:
    mov edx, offset
Tempnormal
    call crlf
    call writeString
    jmp done
Hot:
    mov edx, offset
Temphot
    call crlf
    call writeString
    jmp done
Veryhot:
    mov edx, offset
TempvHot
    call crlf
    call writeString
    jmp done

done:
    exit
main ENDP

END main
```

NAME: WALEED GHANI **ID: 012-0082**

ASSIGNMENT 10 LAB 10







4. Write an Assembly Language program to check whether a triangle can be formed by the given value for the angles.

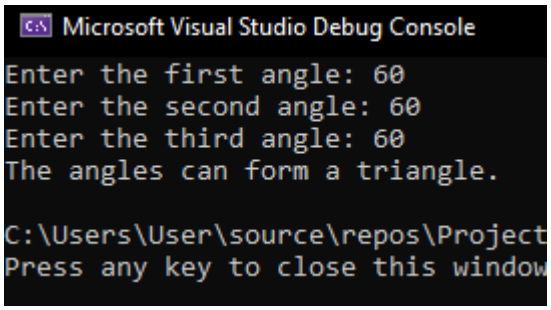
```
INCLUDE Irvine32.inc

.data
   prompt1 BYTE "Enter the first angle: ", 0
   prompt2 BYTE "Enter the second angle: ", 0
   prompt3 BYTE "Enter the third angle: ", 0
   validMsg BYTE "The angles can form a triangle.", 0
   invalidMsg BYTE "The angles cannot form a triangle.", 0
   angle1 DWORD ?
   angle2 DWORD ?
   angle3 DWORD ?
   sum DWORD ?

.code
main PROC
   ; Prompt for the first angle
   mov edx, OFFSET prompt1
   call WriteString
   call ReadInt
```

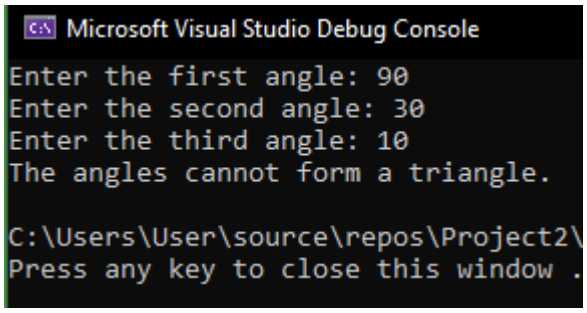
ASSIGNMENT 10 LAB 10

```
    mov angle1, eax      ; Store the first angle in angle1

    ; Prompt for the second angle
    mov edx, OFFSET prompt2
    call WriteString
    call ReadInt
    mov angle2, eax      ; Store the second angle in angle2

    ; Prompt for the third angle
    mov edx, OFFSET prompt3
    call WriteString
    call ReadInt
    mov angle3, eax      ; Store the third angle in angle3

    ; Calculate the sum of the angles
    mov eax, angle1      ; Load angle1 into EAX
    add eax, angle2      ; Add angle2
    add eax, angle3      ; Add angle3
    mov sum, eax         ; Store the sum in sum

    ; Check if the sum is equal to 180
    cmp eax, 180
    je validTriangle     ; If sum == 180, it's a valid triangle
    jmp invalidTriangle  ; Otherwise, it's not a valid triangle

validTriangle:
    mov edx, OFFSET validMsg
    call WriteString
    call Crlf
    jmp done

invalidTriangle:
    mov edx, OFFSET invalidMsg
    call WriteString
    call Crlf

done:
    exit
main ENDP

END main
```

NAME: WALEED GHANI **ID: 012-0082**

ASSIGNMENT 10 LAB 10





NAME: WALEED GHANI **ID: 012-0082**