June 1, 2009

COMPUTER ENGINEERING DEPARTMENT

COE 205

COMPUTER ORGANIZATION & ASSEMBLY PROGRAMMING

Major Exam II

Second Semester (082)

Time: 7:00 PM-9:30 PM

Student Name : __KEY____

Student ID. : ____

Question	Max Points	Score
Q1	40	
Q2	36	
Q3	24	
Total	100	

Dr. Aiman El-Maleh

[40 Points]

(Q1) Fill the blank in each of the following:

(1) Assume that ESP=00000020H, EAX=12345678H and EBX=90ABCDEFH. After executing the instruction PUSH EAX, the content of ESP=ESP-4=0000001CH and EAX=12345678H.

(2) Assume that ESP=00000020H, EAX=12345678H and EBX=90ABCDEFH. After executing the following sequence of instructions, the content of ESP= ESP-4-4+4=0000001CH and EAX=90ABCDEFH.

```
PUSH EAX
PUSH EBX
POP EAX
```

(3) Assuming that ESP=00000020H, after executing the instruction RET 12, the content of ESP= ESP+4+12=00000030H.

(4) Assuming that ESP=00000020H, after executing the instruction Call MyProc, the content of ESP= ESP-4=0000001CH.

(5) Assuming thar register AL contains an alphabatic character, to convert the content of register AL to lower case, we use the following instruction OR AL, 20h.

(6) The code to Jump to label L1 if bits 0, 2, and 5 in AL are all set is:

```
AND AL, 00100101b
CMP AL, 00100101b
JE L1
```

(7) The assembly code given below implements the high-level statement

if ((AL > BL) && (BL > CL)) {X = 1;}; unsigned comparison

```
        CMP AL, BL
        JBE NEXT
        CMP BL, CL
        JBE NEXT
        MOV X,1
NEXT:
```

(8) The assembly code given below implements the high-level statement

if ((AL > BL) || (AL > CL)) {X = 1;} ; signed comparison

```
        CMP AL, BL
        JG  L1
        CMP AL,CL
        JLE NEXT
L1:     MOV X,1
NEXT:
```

(9) The assembly code given below implements the high-level statement

while (EBX <= VAR1) { ; unsigned comparison

EBX = EBX + 5;

VAR1 = VAR - 1

}

```
        CMP EBX,VAR1
        JA  NEXT
TOP:    ADD EBX, 5
        DEC VAR1
        CMP EBX, VAR1
        JBE TOP
NEXT:
```

(10) Assuming that AX=5678H and CL=85H, executing the instruction SHL AX, CL will set AX=CF00H and CF=0.

(11) Assuming that AX=8678H and CL=0CH, executing the instruction SAR AX, CL will set AX=FFF8H and CF=0.

(12) Assuming that AX=6789H and CL=20H, executing the instruction ROL AX, CL will set AX=6789H and CF=unchanged.

(13) Assuming that AX=1234H and BX=5678H, executing the instruction SHRD AX, BX, 8 will set AX=7812H and BX=5678H.

(14) To multiply the content of register EAX by 23 without using multiplications instructions, we use the following instructions:

```
MOV EBX, EAX
SHL EBX,  3          ; EBX = 8 * EAX
SUB EBX, EAX         ; EBX = 7 * EAX
SHL EAX, 4           ; EAX = 16 * EAX
ADD EAX, EBX         ; EAX = 23 * EAX
```

(15) Assuming that AX=02ECH and BX=0020H, executing the instruction DIV BL will result in AX=0C17.

(16) Assuming that AX=FFF4H and BX=FFFBH, executing the instruction IDIV BL will result in AX=FE02.

(17) Assuming that AX=02ECH and BX=0020H, executing the instruction MUL BX will result in AX=5D80 and CF=0.

(18) Assuming that AX=FFF4H and BX=FFFBH, executing the instruction IMUL BX will result in AX=003C and CF=0.

(19) Macros are more efficient than procedures in execution time and less efficient in code size.

(20) We can define the macro SAVE_REGS to save only the registers passed as arguments by pushing them on the stack as follows:

```
SAVE_REGS  MACRO  REGS
               IRP D, <REGS>
                  PUSH D
               ENDM
             ENDM
```

[36 Points]

(Q2) Answer the following questions. Show how you obtained your answer:

(i) Given that **TABLE** is defined as: **TABLE Byte 'Ahmad Ali Anas'**

Determine the content of register **AH** after executing the following code:

```
        XOR AH, AH
        MOV ECX, lengthof TABLE
        LEA EBX, TABLE
        DEC EBX
Next:   JECXZ ENL
        INC EBX
        MOV AL, [EBX]
        OR AL, 20H
        CMP AL, `a`
        LOOPNE Next
        JNE  ENL
        INC AH
        JMP Next
ENL:
```

The content of register AH will be 5 as this program counts the number of occurrences of either character 'a' or character 'A'.

(ii) Determine the content of registers **EAX** and **EBX** after exeucting the following code:

```
        MOV EAX, 7532h
        MOV ECX, 32
        XOR EBX, EBX
Next:
        ROL EAX, 1
        ADC EBX, 0
        LOOP Next
```

The content of EBX will be 8 which is the count of the number of 1's in EAX. However, the content of EAX will not change.

(iii) Determine what will be displayed after executing the following code**:**

```
        MOV  EAX, 0F5h
        XOR  ECX, ECX
        MOV  EBX, 10
L1:     XOR  EDX, EDX
        DIV  EBX
        ADD  DL, '0'
        PUSH EDX
        INC  ECX
        CMP  EAX, 0
        JNZ  L1
L2:     POP  EAX
        Call WriteChar
        LOOP L2
```

The code displays the decimal content of register EAX which is 245.

(iv) Determine what will be displayed after executing the following code**:**

```
        MOV EAX, 1
        JMP   MT[EAX*4]
L1:     MOV AL, 'C'
        JMP EL
L2:     MOV AL, 'O'
        JMP EL
L3:     MOV AL, 'E'
EL:     Call WriteChar
        exit
        MT  DWORD L1, L2, L3
```

The code will display character ‘O’.

(v) Determine what will be displayed after executing the following code**:**

```
PUSH 4
PUSH 3
CALL MYPROC
exit
MYPROC:
        JMP SKIP
        MSG BYTE 10, 13, "Greater!!", 0
                  BYTE 10, 13, "Smaller!!", 0
    Skip:
        MOV EBP, ESP
        LEA EDX, MSG
        MOV ESI, [EBP+4]
        MOV EDI, [EBP+8]
        CMP ESI, EDI
        JG Display
        ADD EDX,  lengthof  MSG
Display:
        Call WriteString
        RET 8
```

The procedure MYPROC gets the two parameters passed from the stack i.e. 3 and 4 and compares the second parameter with the first. If the second parameter is greater than the first, it will print in a new line Greater!!, otherwise it will print in a new line Smaller!!. In this case, since 3 is less than 4, it will print: Smaller!!.

(vi) Determine what will be displayed after executing the following code**:**

```
DDIV MACRO  X, Y

          MOV EAX, X
          MOV EBX, Y
          XOR EDX, EDX
          DIV EBX
          CALL WriteDec
          MOV AL, '.'
          CALL WriteChar
          MOV EAX, 10
          MUL EDX
          DIV EBX
          CALL WriteDec

ENDM
            DDIV 15, 6
```

This macro dispalys the result of dividing X by Y within a single decimal fraction digit. Thus, it will display 2.5.

[24 Points]

(Q3)

(i) Write a procedure, **SelectionSort**, to sort an array of integers (i.e. 32-bit signed numbers) in an **ascending** order. The number of integers to be sorted and the address of the array to be sorted are assumed to be passed on the stack. The procedure should maintain the content of all registers to their state before its execution. **Do not use the USE directive, local directive, pusha and popa instructions in your solution**.

The pseudocode for the **SelectionSort** procedure is given below:

```
SelectionSort (Array, Size)
    for (position= 0 to Size-2)
        MinValue = Array[position]
        MinPosition = position
        for (j=position+1  to Size-1)
            if (Array[j] < MinValue) then
                MinValue = Array[j]
                MinPosition = j
            end if
        end for
        if (position ≠ MinPosition) then
            Array[MinPosition] = Array[Position]
            Array[Position] = MinValue
        end if
    end for
end SelectionSort
```

(ii) Write a complete program, showing the place of procedure definition, to use the procedure **SelectionSort** to sort the Array given below:

Array Dword 10, 2, 0, 15, 25, 30, 7, 22

Note that the Content of Array after sorting will be:
Array Dword 0, 2, 7, 10, 15, 22, 25, 30

```
.686
.MODEL FLAT, STDCALL
.STACK
INCLUDE Irvine32.inc
.DATA
Array DD 10, 2, 0, 15, 25, 30, 7, 22

.CODE
main PROC

    PUSH offset Array
    PUSH lengthof Array
    CALL SelectionSort

  exit      ; exit to operating system
main ENDP
```

```
    SelectionSort PROC
                  PUSH EBP                 ; save registers
                  MOV EBP, ESP
                  PUSH EAX
                  PUSH EBX
                  PUSH ECX
                  PUSH EDX
                  PUSH ESI
                  PUSH EDI

                  MOV ESI, [EBP+8]         ; size of array
                  MOV EBX, [EBP+12]         ; address of array
                  DEC ESI                  ; ESI=size-1
                  MOV EDI, ESI
                  DEC EDI                  ; EDI=size-2
                  XOR ECX, ECX             ; position
    FOR_LOOP:                              ; for (position= 0 to Size-2)
                  CMP ECX, EDI
                  JG END_FOR_LOOP
                  MOV EAX, [EBX+ECX*4]     ; EAX= MinValue
                  MOV EDX, ECX             ; EDX= MinPosition
                  PUSH ECX                 ; save postion

                  INC ECX                  ; 2nd for loop ECX=j
    FOR_LOOP2:                             ; for (j=position+1  to Size-1)
                  CMP ECX, ESI
                  JG END_FOR_LOOP2
                  CMP [EBX+ECX*4], EAX
                  JGE END_IF
                  MOV EAX, [EBX+ECX*4]     ; MinValue=Array[j]
                  MOV EDX, ECX             ; MinPosition=j
    END_IF:
                  INC ECX
                  JMP FOR_LOOP2
    END_FOR_LOOP2:
                  POP ECX                  ; restore position
                  CMP ECX, EDX             ; if (position != MinPosition)
                  JE END_IF2
                  MOV EBP, [EBX+ECX*4]     ; Array[MinPosition] =
                                           ; Array[Position]
                  MOV [EBX+EDX*4], EBP
                  MOV [EBX+ECX*4], EAX     ; Array[Position] = MinValue
    END_IF2:
                  INC ECX
                  JMP FOR_LOOP
    END_FOR_LOOP:
                  POP EDI                    ; restore registers
                  POP ESI
                  POP EDX
                  POP ECX
                  POP EBX
                  POP EAX
                  POP EBP

                  RET 8
    SelectionSort ENDP
END main
```