December 31, 2007

COMPUTER ENGINEERING DEPARTMENT

COE 205

COMPUTER ORGANIZATION & ASSEMBLY PROGRAMMING

Major Exam II

First Semester (071)

Time: 7:00 PM-9:30 PM

Student Name : __KEY____

Student ID. : ____

Question	Max Points	Score
Q1	30	
Q2	10	
Q3	25	
Q4	15	
Q5	20	
Total	100	

Dr. Aiman El-Maleh

[30 Points]

(Q1) Determine whether the following is true or false, and if it is false **correct it**:

(1) (**True**, False) Assume that the instruction JMP NEXT is at offset address 000000A1H in the code segment, its size is 2 bytes, and the label NEXT is at offset 00000020H. Then, the address stored in the assembled instruction for the label NEXT is 7DH.

The address stored = NEXT – IP = 20h – (A1h + 2) = 20h – A3h = 20h + 5Dh = 7Dh.

(2) (True, **False**) After executing the instruction SAL AX, 2, the content of register AX is equal to 2*AX, for both signed and unsigned content.

It is equal 2^2*Ax=4*AX.

(3) (True, **False**) Assuming that EBX=FFFFFFFE and ESI=00000010, the address of the source operand in this instruction MOV AL, [EBX+ESI*2-5] is 00000019 and its addressing mode is Indexed.

The address of the sourse operand is FFFFFFFE + 00000010*2 – 5 = -2 + 32 – 5 = 25d = 00000019h. The addressing mode is **Based Indexed**.

(4) (True, **False**) Given that EAX=FFFF5783**,** executing the instruction CWD will make the content of EAX=00005783.

EAX will not change. DX will be set to 0000.

(5) (**True**, False) The conditional jump instructions JB and JC are equivalent.

(6) (True, **False**) The instruction IN CL, DX inputs the byte whose port address is in DX to register CL.

We cannot use CL in the desitination. The desination must be AL, AX or EAX.

(7) (True, **False**) The code given below implements the conditional statement

if ((CX < 1) AND (AX > 100)) Then CX=0

```
        CMP CX, 1
        JL Zero_index
        CMP AX, 100
        JLE end_if
Zero_index:
        XOR CX, CX
End_if:
```

It implements **if ((CX < 1) OR (AX > 100)) Then CX=0**.

(8) (True, **<u>False</u>**) Assuming that AX=0FFFH and BX=100F, executing the instruction SHLD AX, BX, 4 will set AX=FFF1 and BX=00F0.

BX will not change and its value remians 100FH.

(9) (True, **<u>False</u>**) The interrupt flag (IF) is used to mask all kinds of interrupts.

It is used to mask only maskable hardware interrupts.

(10) (**<u>True</u>**, False) The address of the interrupt service routine for INT 21H is stored in the interrupt vector table (IVT) at entry 84H.

The address is 21h*4 = 84h.

(11) (**<u>True</u>**, False) Assuming that AL contains an Alphabatic character, the instruction AND AL, 0DFH will guarantee that the character in AL is an upper case character. Note that the ASCII code of character ‘A’ is 41H while that of character ‘a’ is 61H.

(12) (True, **<u>False</u>**) Assuming that AL=91H, executing the instruction SAR AL, 33 will make AL=48H.

AL is shifted by one bit to the right. Thus, AL=1100 1000B=C8H.

(13) (**<u>True</u>**, False) Assuming that AX=1234H and DX=0001H, executing the sequence of instructions: {PUSH DX; PUSH AX; POP EAX} will result in EAX=00011234H.

(14) (True, **<u>False</u>**) Assuming that AX=00F2H and BX=0008H, executing the instruction DIV BL will result in AX=1E02H.

It will result in AX=021E.

(15) (True, **<u>False</u>**) Executing the instruction IRET pops one double word from the stack and stores it into EIP.

It pops **a double word** and stores in the EIP register, then pops a word and stores in CS register and finally it pops a double word and stores it in EFLAGS register.

[10 Points]

(Q2) Suppose that you have the following initial content of registers and memory after fetching each of the instructions shown below:

EAX=00001F20H EBX=FFFFFC55H ESP=00001000H EIP=000030B0H

Determine the content of ESP, modified registers, modified flags, and modified memory locations after the execution of each of the following instructions starting from the **initial content** of the registers and memory for the execution of each instruction.

(i) POP EAX.

(ii) PUSH BX.

(iii) Call Sub, where Sub is at an offset address 00001000H.

(iv) RET 2.

Memory Location	Content
00000FFA	FF
00000FFB	10
00000FFC	20
00000FFD	30
00000FFE	40
00000FFF	50
00001000	60
00001001	70
00001002	80
00001003	90
00001004	A0
00001005	B0
00001006	C0

(i) POP EAX

EAX = 90807060
ESP=ESP+4=00001000+4=00001004

(ii) PUSH BX

ESP=ESP-2=00001000-2=00000FFE
[00000FFF:00000FFE]=FC55

(iii) Call Sub

ESP=ESP-4=00001000-4=00000FFC
[00000FFF:00000FFC]=EIP=000030B0
EIP=00001000

(iv) RET 2

EIP=90807060
ESP=ESP+4+2=00001000+6=00001006

[25 Points]

(Q3) Answer the following questions. Show how you obtained your answer:

(i) Given that **TABLE1** and **TABLE2** are defined as:

TABLE1 BYTE ‘I like COE 205’
TABLE2 BYTE ‘I like COE 308’

Determine the content of **AX** after executing the following code:

```
         MOV ECX, lengthof TABLE1
         MOV EBX, -1
         XOR AX, AX
AGAIN:   JECXZ DONE
         INC EBX
         MOV DL, TABLE1[EBX]
         CMP DL, TABLE2[EBX]
         LOOPE AGAIN
         JE DONE
         INC AX
         JMP AGAIN
DONE:
```

The content of register AX will be 2 as the program counts the number of mismatch characters between the two tables.

(ii) Given that **ARRAY** is defined as: **ARRAY BYTE ‘ABCDEF’**

Determine the content of **ARRAY** after executing the following code:

```
      PUSH DS
      POP ES
      STD
      LEA ESI, ARRAY[4]
      LEA EDI, ARRAY[5]
      MOV BH, [EDI]
      MOV ECX, 5
REP   MOVSB
      MOV [EDI], BH
```

The content of ARRAY wil be **FABCDE**.

(iii) Given that **TABLE** is defined as shown below**:**

TABLE BYTE 16 DUP(?)

Determine the content of **TABLE** after executing the following code:

```
        MOV AX, 0E765H
        MOV ECX, 16
        LEA EBX, TABLE
AGAIN:  XOR DL, DL
        ROL AX, 1
        ADC DL, '0'
        MOV [EBX], DL
        INC EBX
        LOOP AGAIN
```

The content of TABLE will be '1110011101100101' as this program stores the binary content of register AX in TABLE.

(iv) Determine the content of register **EAX** after exeucting the following code:

```
        MOV EAX, 739
        MOV EBX, 10
Next:
        XOR ECX, ECX
Again:
        XOR EDX, EDX
        DIV EBX
        ADD ECX, EDX
        TEST EAX, EAX
        JNZ Again
        MOV EAX, ECX
        CMP EAX, 9
        JA Next
```

The content of EAX will be 1. The program adds individual digits of the number in EAX to get another integer.This process is repeated until a single digit is obtained. Thus, initially 7+3+9=19. Then, 1+9=10. Finally, 1+0=1.

(v) Determine the content of register **EAX** after executing the fllowing code:

```
.686
.MODEL FLAT, STDCALL
.STACK

INCLUDE Irvine32.inc
.DATA
TABLE DWORD -10, 20, 30, -50, 66, 12, 330, 1
.CODE
main PROC

PUSH offset TABLE      ; pushed as 32-bit
PUSH lengthof TABLE  ; pushed as  32-bit
CALL MYPROC
exit
main ENDP
MYPROC:
       MOV EBP, ESP
        PUSH EBX
        PUSH ECX
       MOV ECX, [EBP+4]
       MOV EBX, [EBP+8]
       MOV EAX, [EBX]
        DEC ECX
        ADD EBX, 4
NEXT:
       CMP EAX, [EBX]
       JL SKIP
       MOV EAX, [EBX]
SKIP:
      ADD EBX, 4
       LOOP NEXT
       POP ECX
      POP EBX
       RET 8
END main
```

The content of register EAX will be FFFFFFCE=-50d as the program computes the minimum of the elements of TABLE.

[15 Points]

(Q4)

(i) Write a procedure **DISPAVG** that receives as arguments the address of an array of unsigned integers (i.e. DWORD), **Array**, and the number of elements in the array, **Size**. The procedure will then compute the average of the numbers in the array and display it within a single decimal fraction digit. The procedure should preserve the content of all registers used.

(ii) Use the procedure **DISPAVG** to display the average of the given array

Array DWORD 15, 20, 30, 40

Note that your procedure should display the following in a new line:

Average = 26.2

Note that the procedure **WriteDec** can be used for displaying the content of EAX in unsigned decimal format to standard output. The procedure **WriteString** writes a null-terminated string whose address is stored in EDX to standard output. The procedure **WriteChar** writes the character in register AL to standard output. The procedure **Crlf** writes end of line sequence (CR, LF) to standard output.

```
.686
.MODEL FLAT, STDCALL
.STACK

INCLUDE Irvine32.inc

.DATA

   Array DWORD 15, 20, 30, 40

.CODE
main PROC

      MOV ESI, offset Array
      MOV ECX, lengthof Array
      CALL DISPAVG

  exit    ; exit to operating system
main ENDP
```

```
;--------------------------------------------------------------------
; DISPAVG: Computes the average of an array of integers
; Receives: ESI = pointer to an array of doublewords
;               ECX = number of array elements
; Returns:  Displays the average upto a single decimal digit
;--------------------------------------------------------------------
DISPAVG PROC

           PUSHAD
          MOV EBX, ECX

; display message
         LEA EDX, MSG
         CALL WriteString

; compute the average
        XOR EAX, EAX

Next:
        ADD EAX, [ESI] ; compute the sum
        ADD ESI, 4
        LOOP Next
        XOR EDX, EDX
        DIV EBX           ; compute the average

; print integer part
       CALL WriteDec

; print the decimal point
       MOV AL, '.'
       CALL WriteChar

; print fractional part
       MOV EAX, 10
       MUL EDX
       DIV EBX
       CALL WriteDec

       POPAD
       RET
MSG   BYTE 10, 13, "Average = ",0
DISPAVG ENDP

END main
```

[20 Points]

(Q5)

(i) Write a procedure **BinarySearch** to search an array which has been previously sorted in an ascending order. Each element in the array is a 32-bit signed integer. **Three parameters should be passed on the stack**: the address of the array to be searched, the size (number of elements) of the array, and the number to be searched. If the numer is found then **BinarySearch** returns in the EAX register the position of the number in the array. Otherwise, -1 is returned in EAX. All registers except EAX must be preserved by the procedure.

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

```
BinarySearch (array, size, number) {
        lower = 0;
        upper = size-1;
        while (lower <= upper) {
                middle = (lower + upper)/2;
                if (number == array[middle])
                        return middle;
                else if (number < array[middle])
                        upper = middle–1;
                else
                        lower = middle+1;
        }
        return -1;
}
```

(ii) Write a complete program to use the procedure **BinarySearch** to search for the number **3** in the sorted array given below:

Array DWORD 1, 3, 4, 5, 9, 11, 20, 29

Note that the size of the array in this case is 8 and the **BinarySerach** procedure should return the position of number 3 as 1.

```
.686
.MODEL FLAT, STDCALL
.STACK
INCLUDE Irvine32.inc
.DATA
   Array DWORD 1, 3, 4, 5, 9, 11, 20, 29
.CODE
main PROC
      PUSH  offset Array
      PUSH  lengthof Array
      PUSH  3
```

```
        CALL BinarySearch
  exit    ; exit to operating system
main ENDP

BinarySearch PROC
        PUSH EBP
        MOV EBP, ESP
        PUSH EBX                ; save registers
        PUSH ECX
        PUSH ESI
        PUSH EDI
        PUSH EDX

        MOV EBX, [EBP+8]    ; number to be searched
        MOV ECX, [EBP+12]  ; array size
        MOV ESI, [EBP+16]    ; address of array

        XOR EDI, EDI            ; lower=0
        DEC ECX                 ; upper=size-1
WhileL:
        CMP EDI, ECX            ; while (lower <= upper) {
        JA EndWhile
        MOV EDX, EDI            ;  middle = (lower + upper)/2;
        ADD EDX, ECX
        SHR EDX, 1
        CMP EBX, [ESI+EDX*4] ; if (number == array[middle])
        JNE Else1
        MOV EAX, EDX        ; return middle;
        JMP EndWhile
Else1:
        JGE Else2               ; else if (number < array[middle])
        MOV ECX, EDX        ;  upper = middle-1;
        DEC ECX
        JMP Skip
Else2:
        MOV EDI, EDX         ;  lower = middle+1;
        INC EDI
Skip:
        JMP WhileL

EndWhile:
        POP EDX                 ; restore registers
        POP EDI
        POP ESI
        POP ECX
        POP EBX
        POP EBP
        RET 12                  ; return and free parameters
BinarySearch ENDP
END main
```