Dec. 18, 2010

COMPUTER ENGINEERING DEPARTMENT

COE 205

COMPUTER ORGANIZATION & ASSEMBLY PROGRAMMING

Major Exam II

First Semester (101)

Time: 8:15 PM-10:30 PM

Student Name : \_KEY\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student ID. : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Question** | **Max Points** | **Score** |
| **Q1** | **30** |  |
| **Q2** | **36** |  |
| **Q3** | **10** |  |
| **Q4** | **24** |  |
| **Total** | **100** |  |

Dr. Aiman El-Maleh

# **[30 Points]**

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

## Assume that ESP=0000012FH, AX=1234H and BX=5678H. Assume that the address of MPROC is 00200FFA. After executing the instruction sequnece{PUSH AX, PUSH BX, CALL MPROC } the content of ESP=ESP-8=00000127H.

##  Assume that ESP=0000012FH. After executing the instruction RET 4, the content of ESP=ESP+8=00000137H.

## Assuming that register AL contains an alphabatic character, to convert the content of register AL to upper case, we use the following instruction AND AL, 0DFH.

##  The code to Jump to label L1 if regiser AL bits 0 and 2 are 1 or bits 1 and 4 are zero is:

## Test AL, 10010b

## JZ L1

## Test AL, 100b

## JZ Skip

## Test AL, 1

## JNZ L1

## Skip:

## Assuming that EAX=12345678H and ECX=9ABCDEF0H, executing the instruction SHLD EAX, ECX, 12 will set EAX=456789ABH and ECX=9ABCDEF0H.

## To multiply the **signed** content of register EAX by 14.5 without using multiplications instructions, we use the following instructions:

## MOV EBX, EAX

## SHL EAX, 4

## SUB EAX, EBX

## SAR EBX, 1

## SUB EAX, EBX

## Assuming that all variables are 32-bit signed integers, the assembly code implementing the following equation **var3 = -4\*(var1+2)/(15-8\*var2)** is:

## MOV EAX, var1

## ADD EAX, 2

## NEG EAX

## SHL EAX, 2

## MOV EBX, 15

## MOV ECX, var2

## SHL ECX, 3

## SUB EBX, ECX

## CDQ

## IDIV EBX

## MOV var3, EAX

## Suppose that we have a 64-bit number stored in memory in the variable I defined as I Qword. The assembly code to multiply this number by 9 is:

## MOV EAX, DWORD PTR I

## MOV EBX, DWORD PTR I+4

## SHLD, EBX, EAX, 3

## SHL EAX, 3

## ADD DWORD PTR I, EAX

## ADC DWORD PTR I+4, EBX

## Given that the CPU is receiving a word in AX register from the printer. Assume that bits 5 to 10 represent a number. The assembly code to display the decimal value of this number is:

## SHR AX, 5

## AND EAX, 111111b

## Call WriteDec

## Suppose that we would like to translate 8-bit numbers into characters according to a given tanslation table. Part of the translation table is shown below. The assembly code to translate a number in register AL according to the translation table below and store the resulting character in the same register is:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | … |
| ‘e’ | ‘A’ | ‘y’ | ‘Z’ | ‘o’ | ‘B’ | ‘1’ | ‘!’ | ‘h’ | … |

## We need to define the translation table in the data segment as:

## TTABLE BYTE “eAyZoB1!h……”

## Then, we use the following instruction:

## MOVZX EAX, AL

## MOV AL, TTABLE[EAX]

## Assuming signed operands, the assembly code to implement the high-level statement if ( ( (AL ≥ BL) && (BL ≠ CL) ) || (AL<100) ) {X = X+1;} is:

## CMP AL, BL

## JL NEXT

## CMP BL, CL

## JE THEN

## NEXT: CMP AL, 100

## JGE ENDIF

## THEN: INC X

## ENDIF:

## 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 the following definition in the data segment:

## **Array dword 0, 1, 2, 3, 4**

## **dword 5, 6, 7, 8, 9**

## **dword 10, 11, 12, 13, 14**

## Determine the content of Array after executing the following code**:**

 MOV ECX, lengthof Array

 MOV ESI, 1\*sizeof Array

 MOV EDI, 2\*sizeof Array

 XOR EBX, EBX

Next:

 MOV EAX, Array[ESI+EBX\*4]

 XCHG EAX, Array[EDI+EBX\*4]

 MOV Array[ESI+EBX\*4], EAX

 INC EBX

 LOOP Next

The content of Array will be as follows as row 1 and row 2 are swapped:

## **Array dword 0, 1, 2, 3, 4**

## **dword 10, 11, 12, 13, 14**

## **dword 5, 6, 7, 8, 9**

## **(ii)** Determine the content of EBX after executing the following code**:**

 MOV EAX, 7

 CALL MyProc

MyProc PROC

 CMP EAX, 2

 JAE Next

 MOV EBX, EAX

 RET

Next:

 PUSH EAX

 SUB EAX, 2

 CALL MyProc

 POP EAX

 ADD EBX, EAX

 RET

MyProc ENDP

 The content of EBX will be =10h=16d=7+5+3+1

## **(iii)** Given the following definition in the data segment:

##  Array Dword 4, 2, 1, 3

## Determine the content of Array after executing the following code**:**

MOV ESI, lengthof Array

DEC ESI

MOV EDI, ESI

DEC EDI

XOR ECX, ECX

FLOOP:

CMP ECX, EDI

JG EFLOOP

MOV EAX, Array[ECX\*4]

MOV EDX, ECX

MOV EBX, ECX

INC ECX

FLOOP2:

CMP ECX, ESI

JG EFLOOP2

CMP Array[ECX\*4], EAX

JGE EIF

MOV EAX, Array[ECX\*4]

MOV EDX, ECX

EIF:

INC ECX

JMP FLOOP2

EFLOOP2:

MOV ECX, EBX

CMP ECX, EDX

JE EIF2

MOV EBP, Array[ECX\*4]

MOV Array[EDX\*4], EBP

MOV Array[ECX\*4], EAX

EIF2:

INC ECX

JMP FLOOP

EFLOOP:

The content of Array will be sorted in increasing order and will be:

## Array Dword 1, 2, 3, 4

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

## **TABLE1 BYTE ‘This is COE 205’**

**TABLE2 BYTE ‘This is 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 AX will be 2 which is the number of unequal characters between TABLE1 and TABLE2.

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

PUSH 23

PUSH 5

CALL MTest

MTest PROC

 MOV EBX, [ESP+4]

 MOV EAX, [ESP+8]

 XOR EDX, EDX

 DIV EBX

 CALL WriteDec

 MOV AL, '.'

 CALL WriteChar

 MOV EAX, 10

 MUL EDX

 DIV EBX

 CALL WriteDec

 RET 8

MTest ENDP

The program will display 4.6 which is the result of dividing 23 by 5.

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

MOV EAX, 5

MOV EBX, 9

XOR ECX, ECX

Next:

SHR EBX, 1

JNC Skip

ADD ECX, EAX

Skip:

SHL EAX, 1

CMP EBX, 0

JNE Next

MOV EAX, ECX

##  Call WriteDec

The program will display 45 which is the result of multiplying 9 by 5.

**[10 Points]**

#  **(Q3)** Write a macro, **DigitSum**, to compute and display the sum of the digits of a number **n** passed as a aparmeter to the macro. The macro should preserve the content of all temporary registers used. Then, use the macro to compute the sum of the digits in the number 123. Your macro should display the result as 6 since 6=1+2+3.

# DigitSum MACRO n

# LOCAL NEXT

# PUSH EAX

# PUSH EBX

# PUSH ECX

# PUSH EDX

# XOR ECX, ECX

# MOV EAX, n

# MOV EBX, 10

# NEXT:

# XOR EDX, EDX

DIV EBX

ADD ECX, EDX

# CMP EAX, 0

JNE NEXT

MOV EAX, ECX

CALL WriteDec

# POP EDX

# POP ECX

# POP EBX

# POP EAX

# ENDM

# Using the macro to compute the sum of the digits in the number 123 will be done as follows:

# DigitSum 123.

**[24 Points]**

#  **(Q4)**

# **(i)** Write a procedure, **MatrixMul**, to multiply two matrices of integers where Matrix1 is of size R1xC1 integers and Matrix2 is of size R2xC2 integers. If C1≠R2, then the two matrices cannot be multiplied and the procedure prints a statement indicating that the two matrices cannot be multiplied. Otherwise, the procedure computes the resultant matrix and displayes it. The addresses of the two matrices and their dimensions are assumed to be passed on the stack. The procedure should maintain the content of all registers to their state before its execution.

The pseudocode for multiplying two matrices **m1** and **m2** and storing the result in matrix **mult** is given below:

for(i=0; i<R1; i++){
       for(j=0; j<C2; j++){
                mult[i][j]=0;
                for(k=0; k<C1; k++){
                    mult[i][j] = mult[i][j]+m1[i][k]\*m2[k][j];
                }
       }

}

# **(ii)** Write a complete program, showing the place of procedure definition, to use the procedure **MatrixMul** to multiply the two matrices given below:

M1 dword 1, 2, 3

dword 4, 5, 6

M2 dword 1, 2

dword 3, 4

dword 5, 6

Your program should display the result of matrix multiplication as:

22 28

49 64

.686

.MODEL FLAT, STDCALL

.STACK

INCLUDE Irvine32.inc

.data

 M1 dword 1, 2, 3

 dword 4, 5, 6

R1 EQU 2

C1 EQU lengthof M1

M2 dword 1, 2

 dword 3, 4

 dword 5, 6

R2 EQU 3

C2 EQU lengthof M2

MSG BYTE "The two matrices cannot be multiplied", 10, 13, 0

MULT DWORD 20 DUP (20 DUP(0))

.code

main PROC

 PUSH offset M1

 PUSH R1

 PUSH C1

 PUSH offset M2

 PUSH R2

 PUSH C2

 CALL MatrixMul

main ENDP

MatrixMul PROC

 PUSH EBP

 MOV EBP, ESP

 SUB ESP, 12 ; allocate 3 local variables for i, j, k

 ; i=[EBP-4], j=[EBP-8], k=[EBP-12]

 PUSHAD

 ; [EBP+8]=C2, [EBP+12]=R2, [EBP+16]=offset of M2

 ; [EBP+20]=C1, [EBP+24]=R1, [EBP+28]=offset of M1

 ; check if C1 != R2

 MOV EAX, [EBP+20]

 CMP EAX, [EBP+12]

 JNE NEQUAL

 MOV DWORD PTR [EBP-4], 0; i=0

FOR1: MOV EDX, [EBP-4]

 CMP EDX, [EBP+24]

 JGE EFOR1

 MOV DWORD PTR [EBP-8], 0; j=0

FOR2: MOV EDX, [EBP-8]

 CMP EDX, [EBP+8]

 JGE EFOR2

 MOV EAX,[EBP+8]

 SHL EAX, 2

 MUL DWORD PTR [EBP-4]

 MOV ECX, EAX; ECX contains starting address of row i of mult

 MOV EDX, [EBP-8]

 SHL EDX, 2

 ADD EDX, ECX

 MOV MULT[EDX], 0; mult[i][j]=0

 MOV DWORD PTR [EBP-12], 0; k=0

FOR3: MOV EDX, [EBP-12]

 CMP EDX, [EBP+20]

 JGE EFOR3

 MOV EAX, [EBP+20]

 SHL EAX, 2

 MUL DWORD PTR [EBP-4]

 MOV EBX, EAX; EBX contains starting address of row i of m1

 MOV EAX, [EBP+8]

 SHL EAX, 2

 MUL DWORD PTR [EBP-12];

 MOV ESI, EAX; ESI contains starting address of row k of m2

 MOV EDX, [EBP-8]

 SHL EDX, 2

 ADD EDX, ECX; EDX contains address of mult[i][j]

 PUSH EDX

 PUSH DWORD PTR MULT[EDX]; mult[i][j]

 MOV EDX, [EBP-12]

 SHL EDX, 2

 ADD EDX, EBX;

 MOV EDI, [EBP+28]

 MOV EAX, [EDI+EDX]; m1[i][k]

 MOV EDX, [EBP-8]

 SHL EDX, 2

 ADD EDX, ESI; EDX contains address of m2[k][j]

 MOV EDI, [EBP+16]

 MUL DWORD PTR [EDI+EDX]; m1[i][k]\*m2[k][j]

 POP EDX

 ADD EAX, EDX

 POP EDX; EDX contains address of mult[i][j]

 MOV MULT[EDX], EAX; mult[i][j] = mult[i][j]+m1[i][k]\*m2[k][j];

 INC DWORD PTR [EBP-12]

 JMP FOR3

EFOR3: INC DWORD PTR [EBP-8]

 JMP FOR2

EFOR2: INC DWORD PTR [EBP-4]

 JMP FOR1

EFOR1: ; printing the result of multiplication

 XOR ESI, ESI

FOR4: CMP ESI, [EBP+24]

 JGE EFOR4

 XOR EDI, EDI

FOR5: CMP EDI, [EBP+8]

 JGE EFOR5

 MOV EAX, [EBP+8]

 SHL EAX, 2

 MUL ESI

 MOV EBX, EDI

 SHL EBX, 2

 ADD EAX, EBX

 MOV EAX, MULT[EAX]; print mult[i],[j]

 Call WriteDec

 MOV AL, ' '

 Call WriteChar

 INC EDI

 JMP FOR5

EFOR5: CALL Crlf

 INC ESI

 JMP FOR4

EFOR4: JMP DONE

NEQUAL: MOV EDX, offset MSG

 CALL WriteString

DONE: POPAD

 MOV ESP, EBP; free local variables

 POP EBP

 RET 24

MatrixMul ENDP

END main