FAST- National University of Computer & Emerging Sciences, Karachi.
Department of Computer Science,
Mid Term II Examinations, Spring 2019.
3rd April, 2019, 09:00 am – 10:00 am

SOLUTION

Instructions:

- Attempt all questions. Return the question paper.
- Read each question completely before answering it. There are **3 questions on 2 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.
- All the answers must be solved according to the SEQUENCE given in the question paper.
- Where asked for values, only provide the **hex-decimal** values.
- Problems needing iterations should be coded using *iterative instructions*. No points will be awarded otherwise.
- Where asked for example, provide suitable example, otherwise no points will be awarded.

Time Allowed: 60 minutes. **Maximum Points**: 30 points

==========

Q No. 1 Answer each of the following questions (**in not more than 20 words**): **[2 x 5 = 10 Points]**

(i) What is the difference between CALL and JMP instructions in context of branching?

CALL pushes return address on the stack in order to return to the point after the call, JMP simply braches to destination.

(ii) "Stack parameters are more convenient than register parameters", prove with an example.

Registers are limited in number, hence it is more convenient to use stack parameters.

(iii) Why sign extension is necessary before signed division (IDIV)?

In order to preserve the SIGN of operand.

(iv) With the help of an example, clarify when does one-operand IMUL set the Carry Flag and the Overflow Flag?

IMUL sets the Carry Flag and Overflow flag when upper half of product is NOT the sign extension of lower half.

E.g.

```
mov     al,48
mov     bl,4
imul    bl                          ; AX = 00C0h, OF = 1
```

(v) How LOOPE is different from the LOOP instruction? Provide an example.

Before looping, LOOPE checks for ZF == 1 along with CX > 0 , whereas LOOP only checks for CX > 0

Q No. 2 Given that EAX = 0Ah, EBX = 05h, ECX = 05h, EDX = 01h, and ESP = 0000 010Fh, draw out the run-time stack (diagrams), with addresses after each CALL and RET instructions. No points will be awarded if addresses are found missing/wrong. [**10 Points**]

```
        main PROC
0001    SHL   AL, 2; 28h
0002    PUSH  EBX
0003    PUSH  EAX
0004    CALL  f1     ;#1
0005    POP   EBX
0006    RET          ;#5
        main ENDP
```

```
        f1 PROC
000C    STC
000D    RCL   DL, 2; 06h
000E    SHR   CL, 1; 02h
000F    ROR   AH, CL; 00
0010    PUSH  EAX; 0028
0011    PUSH  EDX; 0006
0012    CALL  f2     ;#2
0013    POP   EDX
0014    POP   EAX
0015    RET          ;#4
        f1 ENDP
```

```
        f2 PROC
0017    PUSH  EBX
0018    PUSH  ECX
0019    POP   EBX
001A    POP   ECX
001B    RET          ;#3
        f2 ENDP
```

#1

0000 010Bh	0000 0005h
0000 0107h	0000 0028h
0000 0103h	0000 0005h

#2

0000 010Bh	0000 0005h
0000 0107h	0000 0028h
0000 0103h	0000 0005h
0000 00FFh	0000 0028h
0000 00FBh	0000 0006h
0000 00F7h	0000 0013h

#3

0000 010Bh	0000 0005h
0000 0107h	0000 0028h
0000 0103h	0000 0005h
0000 00FFh	0000 0028h
0000 00FBh	0000 0006h

#4

0000 010Bh	0000 0005h
0000 0107h	0000 0028h

#5

0000 010F	

$\Rightarrow$ **EIP** = 0000 010Bh

Q No. 3

(a) Write an x86 assembly procedure FACTORIAL that will compute factorial (N!) for an unsigned variable N. **[5 Points]**

HINT: **N!** = 1 if N == 1
N! = N x (N – 1) x (N – 2) x … x 1 if N > 1

SOLUTION

```
            MOV   EAX, 1
            MOV   CX, N

            L1:
            MUL   CX
            LOOP  L1

            MOV   N, AX
```

(b) The greatest common divisor (GCD) of two integers is the largest integer that will evenly divide both integers. Implement this procedure in x86 assembly language. **[5 Points]**

SOLUTION

```
            ; Assuming all the integers are unsigned WORD variables

sample PROC
            MOV   EDX, 0
            MOV   EAX, 0

            MOV   CX, int1

            L2:   MOV DX, 0
            MOV   AX, int2
            DIV   CX
            CMP   DX, 0
            JNE   L1

            MOV   AX, int1
            MOV DX, 0
            DIV CX
            CMP   DX, 0
            JE    L3

            L1: LOOP L2

            L3: MOV GCD, CX
            ret
sample ENDP
```

----STAY BRIGHT----