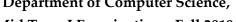
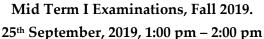


FAST- National University of Computer & Emerging Sciences, Karachi. Department of Computer Science,







SOLUTION

Course Code: EE 213 Course Name: Computer Organization and Assembly Language
Instructors: Dr. Nouman M Durrani, Muhammad Danish Khan, and Syed Zain-ul-Hassan

Instructions:

- Except your Roll No and Section, DO NOT SOLVE anything on this paper.
- 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 must not contradict any statement in the question paper.
- All the answers must be solved according to the SEQUENCE given in the question paper, otherwise points will be deducted.
- This paper is subjective.
- 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.

Time Allowed: 60 minutes. Maximum Points: 30 points

Q No. 1 Briefly answer (2-3 lines) each of the following:

 $[5 \times 2 = 10 \text{ points}]$

- (i) Explain why memory access takes more clock cycles than register access?
- (ii) After a program has been loaded into memory, how does it begin execution?

 Instructions are maintained on an INSTRUCTION QUEUE where each individual instruction is pointed to by EIP and to start its execution cycle.
- (iii) What are linkers? Why do we need linkers? Linkers are utility programs to link the system resources (libraries etc) with the object code.
- (iv) Give one example instruction for each of the following addressing modes:
 - a. Indirect Addressing Mode

MOV EAX. [ESI]

b. Base Indexed

MOV ARR[6], 13

(v) Why it is not allowed to directly set the EIP register?

EIP is aimed at pointing to next executable instructions, setting EIP can damage the execution plan.

Q No. 2

a. Assuming that array1 is a WORD array containing decimal numbers from 0-99

```
array1 WORD 0,1,2,3,4...99
```

Write an assembly language code that should sum up all the ODD numbers in the array and stores the resulting value in a variable named **result**. [05 Points]

```
VOM
     ECX, 50
     ESI, OFFSET [ARRAY1+2]
MOV
MOV
     result, 0
VOM
     AX, 0
L1:
     ADD
          AX, [ESI]
     ADD
           ESI, 4
LOOP L1
MOV
     result, AX
```

b. Consider the following data definitions:

```
arr1 BYTE 10h, 20h, 30h, 40h, 50h arr2 BYTE 5 DUP(?)
```

Write a nested loop that fills arr2 with each element of arr1 sequentially i.e. during each iteration of the outer loop, the inner loop must fill arr2 with a new element from arr1.

Hint: After first iteration of outer loop, arr2 = {10h,10h,10h,10h,10h}, after second iteration arr2 = {20h,20h,20h,20h}, and so on [05 Points]

```
.data
     arr1 BYTE 10h, 20h, 30h, 40h, 50h
     arr2 BYTE 5 DUP(?)
     temp DWORD ?
.code
     MOV
           ECX, LENGTHOF ARR1
     MOV
           ESI, OFFSET arr1
                     EDI, OFFEST arr2
     outer:
                MOV
                      temp, ECX
                MOV
                VOM
                      al, [esi]
                      ecx, lengthof arr2
                MOV
                inner:
                                  [edi], al
                            mov
                            inc
                                  edi
                LOOP inner
                      ecx, temp
                MOV
                 INC
                      ESI
     LOOP outer
```

Q No. 3

(a) Assuming that X is an SBYTE operand and Z is a DWORD operand, identify the problem(s) with each of the following instructions: $[5 \times 1 = 5 \text{ Points}]$

(i) MOV WORD PTR [Z], [ESI]

Two memory operands are not allowed.

(ii) MOV ECS, OFFSET Z

ECS cannot be used as general purpose.

(iii) MOVZX BH, AX

Invalid Extension, destination must be larger.

(iv) MOVZX EDX, X

X is a signed variable, MOVSX should've been used.

(v) PUSH X

X is an SBYTE variable, cannot be pushed.

(b) Given that EAX = 0Ah, ECX = 05h, EDX = 03h, and ESP = 0000 001Fh, draw out the run-time stack (diagrams), with addresses after each numbered (#1 and #2) instruction. No points will be awarded if addresses are found missing/wrong. [05 Points]

main PROC			
ADD	AL, 2	;EAX = 0000 00 0C h	
INC	СН	;ECX = 0000 01 05h	
PUSH	ECX	;[0000 001B] = 0000 0105	
XCHG	DH, DL	;EDX = 0000 0300	
SUB	DH,AL	;EDX = 0000 F7 00	
PUSH	EDX ; #1	;[0000 0017] = 0000 F700	
NEG	EDX		
PUSH	EAX	;[0000 0013] = 0000 000C	
POP	EDX ; #2	;EDX = 0000 000C	
		;[ESP] =[0000 0017]=0000 F700	
POP	EAX	;EAX = 0000 F700	
main ENDP			

#1

 0000 001B
 0000 017
 0105

 0000 0017
 0000 F700

#2

0000 001B	0000	0105
0000 0017	0000	F700

STAY BRIGHT