FAST- National University of Computer & Emerging Sciences, Karachi
School of Computing,
Final Examinations, Fall 2020
8th January, 2021, 03:00 pm – 06:00 pm.

Course Code: EE 229	**Course Name:** Computer Organization and Assembly Language
Instructors: Dr. Muhammad Nouman Durrani, Muhammad Danish Khan, Shoaib Rauf	
Student's Roll No:	**Section:**

SOLUTION

Time Allowed: 3 hrs. **Maximum Points**: 80 Points

==========

PART – I

Short Questions

Question No.1 Answer all the questions in Part-I **[20 x 2 = 40 Points]**

(i) How do assemblers and linkers work together?

The assembly code is converted into machine language (object code) by assembler, Linker then links all the necessary libraries and predefined code with the object file to make it executable.

(ii) How many bytes are allocated as a result of the following data definition directive:

```
X1    BYTE    20 DUP(30 DUP('COAL'))
```

2400 BYTES

(iii) Differentiate between the following Assembly Language instructions:

```
MOV   EBX, OFFSET X1
MOV   EBX, X1
```

The first instruction copies the base address of X1 in EBX, second instruction copies the contents.

(iv) Elaborate the difference between MOVZX and MOVSX instructions through an example.

MOVZX is unsigned extension of source operand, MOVSX extends the sign.
e.g.

```
MOV         AL, 9C
MOVZX       CX, AL          ; CX = 009Ch
MOVSX       DX, AL          ; DX = FF9Ch
```

(v) Consider the following code. What is the value of AX after execution of the following instructions?

```
MOV AX, 10H
MOV CX, 0AH
L1:
      INC AX
      DEC CX
```

```
LOOPNE L1
```

AX = 15h

(vi) Consider the following code snippet. What is the value of AX after execution of the following instructions?

```
MOV   AX, 90F8h
MOV   CX, 08h
L1:   SHRD AX, CX, 1
JC    L2
LOOP  L1
L2:   RET
```

AX = A90Fh

(vii) Elaborate the difference between RCR and ROR with the help of working example.

ROR (Rotate Right) Instruction shifts each bit to the right and copies the lowest bit into the Carry flag and the highest bit position (MSB).
The **RCR (rotate carry right)** instruction shifts each bit to the right, copies the Carry flag into the MSB, and copies the LSB into the Carry flag:

(viii) Write equivalent x86 assembly instructions for the following operation:

```
POP   EBX
```

```
MOV   EBX, [ESP]
ADD   ESP, 4
```

(ix) When does RECURSION is preferred over LOOPING? Give some real world example.

Free Responses.

(x) Write the x86 assembly PROTOYPE for following sample function:

```
void sample(int length, char ch, int arr[]);
```

```
sample      PROTO, length: DWORD, ch: BYTE, arrPtr: PTR DWORD
```

(xi) Write a valid x86 assembly INVOKE instruction for the following sample function:

```
int sample(int arr[], int num, int* x)
```

```
INVOKE sample, addr arr32, mem32, addr var32
```

(xii) Write a valid x86 assembly function signature using PROC for the following sample function:

```
void sample(char ch, int num, char chArr[])
```

```
sample PROC, ch: BYTE, num: DWORD, chPTR: PTR BYTE
```

(xiii) What will happen, if immediately upon entering a subroutine you execute a "POP" instruction?

```
The subroutine will lose its return address.
```

(xiv) Write a single x86 instruction to replace each of the following two instructions:

```
LEA    EBX, X1
MOV    EBX, OFFSET X1

LOADSB
MOV    AL, [ESI]
```

(xv) When do you typically use the `CBW` and `CWD` instructions?

With 8bits, and 16bits signed divisions respectively.

(xvi) Assume a four-byte x86 machine code as `B9 C5 12 00`. How would you determine the size of register used in this instruction? Explain.

The W-bit(0^{th} bit) of Opcode byte determines the size of register in this instruction.

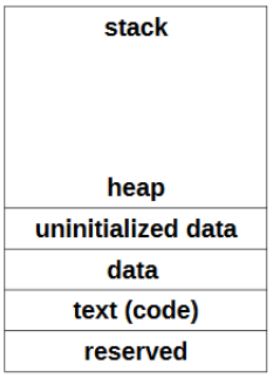
(xvii) What are the key benefits of RISC architecture (like MIPS)?

Faster
Lesser Complexity

(xviii) In MIPS, how 16 bit value is loaded from a memory location into the destination register? How a specified immediate value is loaded into a destination register? Give Instruction examples only.

```
LH    $12, 80($3)
LI    $12, 100
```

(xix) Draw and briefly discuss the general memory layout of a program in MIPS architecture.



(xx) Explain the execution steps of MIPS instruction `BNEQ $t0, $t1, L1`. Assuming `L1` as a valid label.

This pseudo instruction is first converted to bare-instruction which then is fetched, decoded, and executed accordingly.

PART – II
Descriptive Questions

Question No. 2 **[4+4 = 8 Points]**

(i) Given the following array, write a procedure that should replace prime numbers with their mathematical TWICE using the LOOP instruction:

```
wArray          WORD        1, 3, 4, 6, 7, 8, 11, 13,19, 21

MOV     ESI, OFFSET wArray
MOV     CX, LENGTHOF wArray

L1:     PUSH  CX
        CMP   [ESI], 1
        JE    isPrime
        MOV   CX, [ESI]
        MOV   BX, CX
        DEC   CX

        L2:   CMP   CX, 1
              JE    isPrime
              MOV   AX, BX
              DIV   CX
              CMP   AH, 0
              JE    Continue
              LOOP  L2

isPrime:      MOV   BX, [ESI]
              ADD   BX, BX
              MOV   [ESI], BX

Continue:     ADD   ESI, typeof wArray
              POP   CX
              LOOP  L1
```

(ii) Write a short code segment, using the LOOP instruction that calculates AVERAGE of an integer array `intArray` in EAX.

```
MOV     ECX, LENGTHOF intArray - 1
MOV     ESI, OFFSET intArray
MOV     EAX, 0
MOV     EDX, 0
L1:     ADD   EAX, [ESI]
        ADD   ESI, TypeOf intArray
LOOP    L1
MOV     EBX, LENGTHOF intArray
DIV     EBX
```

Question No. 3 **[8 Points]**

Implement the following pseudo-code in an x86 assembly program. Your program must maintain and clean the stack used for this program, don't use ENTER/LEAVE instructions and LOCAL directive.

```
int array[] = {10,60,20,33,72,89,45,65,72,18};
int sample = 50;
```

```
   void main(){
      int sum = 0;
      int index = 0;
      func1 (index, &sum);
   }

   void func1(int i, int* s)
   {
      int arraySize = sizeof Array / sizeof sample;
      while(i<arraySize)
      {     if (array[i] <= sample)
              { *s += array[i]; }
            i++;
      }
   }
```

```
.data
      Array       DWORD       10, 60, 20, 33, 72,89,45,65,72,18
      Sample            DWORD       50
main  PROC
      PUSH        EBP
      MOV         EBP, ESP
      SUB         ESP, 8            ; allocating space for LOCAL DATA
      MOV         [EBP-4], 0  ; sum
      MOV         [EBP-8], 0  ; index
      INVOKE            func1, [EBP-8], EBP-4
      MOV         ESP, EBP    ; cleaning main's LOCAL DATA
      POP         EBP
      RET
ENDP  main
Func1 PROC, i: DWORD, s: PTR DWORD
      PUSH        EBP
      MOV         EBP, ESP
      SUB         ESP, 4            ; arraySize
      MOV         ESI, s
      MOV         EAX, sizeof ARRAY
      MOV         EDX, 0
      MOV         ECX, sizeof Sample
      DIV         ECX
      MOV         [EBP-4], EAX            ;     arraySize =sizeOfArray/sizeOf sample
      MOV         EBX, i
      L1:         CMP         [EBP-4], EBX
                  JA          continue
                  MOV         EBX, sample
```

```
                CMP         array[i], EBX
                JA          continue
                MOV         ECX, array[i]
                ADD         [ESI], ECX
    Continue:   INC         i
    ……
    .
    MOV         ESP, EBP
    POP         EBP
    RET         8
Func1 ENDP
```

Question No. 4 **[4+4= 8 Points]**

(i) Write an x86 assembly procedure that takes two string arrays as ARGUMENTS, compares them, and displays whether both the strings are same or not.

```
Sample PROC, p1: PTR BYTE, p2:PTR BYTE
   MOV      ESI, p1
   MOV      EDI, p2
   MOV      ECX, lengthof String    ;Assuming given length
   REPE     CMPSB
   JNE      notSame                 ;Printing 'Not Same'
   JMP      isSame                  ;Printing Same
```

(ii) Given that `EBX` points to the following array `ewords`, write x86 code snippet to process this array and print the starting offset of each word starting with letter `e` or `E`.

```
            ewords byte "The eagle eyed snake eagerly attacked Easter eggs", 0
MOV   ESI, OFFSET ewords
MOV   ECX, LENGTHOF ewords
DEC   ECX,
L1:   CMP   [ESI], 'E'
      JE    condition2
      CMP   [ESI], 'e'
      JE    condition2
      JMP   continue
condition2: CMP   [ESI-1], ' '
            JNE   continue
            MOV   EDX,  ESI
            ; call display
Continue:   INC   ESI
Loop  L1
```

Question No. 5 Encode the following x86 assembly instructions **[2 x 4 = 8 Points]**

(i) `CMP BX, [BP+08h]`

`3B 01 011 110 <- 08`

`= 3B 5E 08h`

(ii) `MOV [BX+4099h], 4F87h`

`C7 10 000 111 <-99 40 <- 874F`

`= C7 87 9940 874Fh`

(iii) `SUB BX, 7E90h`

`81 + 03 <- 907E`

`= 84 907Eh`

(iv) `INC WORD PTR [SI+2A6h]`

`FF 10 000 100 <- A6 02`

`= FF 84 A6 02h`

Question No.6 MIPS (RISC) Pipelining and Hazards **[4+4 = 8 Points]**

(i) Given the MIPS instructions below, **draw** the neat and clean pipelined diagram for these instructions and **identify** the structural hazards and data hazards in given instructions:

```
ADD     $2, $7, $9
LW      $11, 100($2)
OR      $10, $2, $9
LH      $13, 100($9)
NOR     $8, $6, $13
```

Solution:

I1	IF	ID	OF	WB					
I2		IF	ID	OF	MEM	WB			
I3			IF	ID	OF	WB			
I4				IF	ID	OF	MEM	WB	
I5					IF	ID	OF	WB	

I1 and I2: raw hazard over $2

I2 and I3: Structural Hazard

I4 and I5: RAW hazard over $13

I4 and I5: Structural Hazard

(ii) Now resolve the data hazards by **RESCHEDULING** and structural hazards by **STALLING**. Draw the updated pipeline of the rescheduled instructions. How many cycles are required, if the above code is rescheduled.

Solution:

```
ADD     $2, $7, $9
LH      $13, 100($9)
```

```
LW          $11, 100($2)
OR          $10, $2, $9
NOR         $8, $6, $13
```

*****STAY BRIGHT*****

Instruction	Opcode (in hexa-decimal)	Opcode Extension Bits
CMP reg16, mem16	3B	
CMP reg16/mem16, reg16	39	
CMP reg16/mem16, imm16	81	111
INC reg16/mem16	FF	000
INC reg8/mem8	FE	000
MOV reg8/mem8, reg8	88	
MOV reg16/mem16, reg16	89	
MOV reg8, mem8	8A	
MOV reg16, imm16	B8	
MOV mem16, imm16	C7	000
SUB mem8/reg8, reg8	28	
SUB mem16/reg16, reg16	29	
SUB reg8, mem8	2A	
SUB reg16, mem16	2B	
SUB reg16/mem16, imm16	81	101

Mod=11			Effective Address Calculation			
R/M	W=0	W=1	R/M	Mod= 00	Mod= 01	Mod= 10
000	AL	AX	000	[Bx] + [SI]	[BX] + [SI] + D_8	[BX] + [SI] + D_{16}
001	CL	CX	001	[BX] + [DI]	[BX] + [DI] + D_8	[BX] + [DI] + D_{16}
010	DL	DX	010	[BP] + [SI]	[BP] + [SI] + D_8	[BP] + [SI] + D_{16}
011	BL	BX	011	[BP] + [DI]	[BP] + [DI] + D_8	[BP] + [DI] + D_{16}
100	AH	SP	100	[SI]	[SI] + D_8	[SI] + D_{16}
101	CH	BP	101	[DI]	[DI] + D_8	[DI] + D_{16}
110	DH	SI	110	Direct Address	[BP] + D_8	[BP] + D_{16}
111	BH	DI	111	[BX]	[BX] + D_8	[BX] + D_{16}