FAST- National University of Computer & Emerging Sciences, Karachi
School of Computing,
Mid Term II Examinations, Fall 2020
25th November, 2020, 02:30 pm – 03:30 pm.

SOLUTION

Course Code: EE 229	**Course Name:** Computer Organization and Assembly Language
Instructor: Dr. Muhammad Nouman Durrani, Muhammad Danish Khan, Shoaib Rauf	

Instructions:

- Attempt all questions and return the question paper with the answer copy.
- 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, otherwise points will be deducted.
- Where asked for values, only provide the **hex-decimal** values.
- Examples are mandatory where asked, no point will be awarded otherwise.

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

==========

Question No. 1

Briefly answer each of the following questions, examples are necessary where asked. [**5 x 2 = 10 Points**]

(i) Explain the difference between **CMP** and **SUB** through some working example.

CMP and SUB both subtract the second operand from first operand, CMP doesn't store the results in first operand whereas SUB does.

```
e.g.    MOV   AL, 7
        CMP   AL, 7          ; AL= 7    ZF = 1
        SUB   AL, 7          ; AL = 0   Zf = 1
```

(ii) How can we replace **CBW** instruction with some other x86 instruction(s)? Give an example.

```
MOV     BL, AL
MOVSX   AX, BL
```

(iii) Elaborate the difference between **ret** and **ret 8**.

RET simply returns the control to calling procedure, whereas **RET 8** returns the control to calling procedure and cleans up the arguments (8 bytes in this case) off the stack frame, 8 is added to ESP to de-allocate the arguments.

(iv) Given that **AL = - 9**, write some x86 instructions to replace the contents of Accumulator with its mathematical cube (x^3).

```
MOVSX   BX, AL; -9 X -9 X -9
IMUL    AL ;  -9 X -9
IMUL    BX;
```

(v) How **SHLD** instruction is different from **SHL** instruction? Elaborate through some working example.

SHL logically shifts the destination bits towards left and fills the newly created bit position with '**zero**', whereas **SHLD** shifts the destination's bits to left and fills the newly created bit positions with **Most Significant Bits** of source.

e.g.

```
MOV   AX, 81C1h
MOV   BX, C9C0h
SHL   AX, 8          ;    AX = C100h
MOV   AX, 81C1h
MOV   BX, C9C0h
SHLD  AX, BX, 8      ;    AX = C1C9h
```

Question No. 2 **[2 X 5 = 10 Points]**

(i) Write an assembly language procedure that should replace all the negative numbers in the given array with their absolute(positive) values without using **MUL/IMUL**:

```
Numbers    SWORD      -12, 13, 15, -17, -9, 3 DUP(-3), 100, 20
```

```
MOV   ESI, OFFSET Numbers
MOV   ECX, LENGTHOF Numbers
L1:   CMP   [ESI], 0
      JGE   L2
      NEG   SWORD PTR [ESI]
      L2:   ADD   ESI, TYPE Numbers
LOOP  L1
```

(ii) Write down the equivalent x86 assembly code for the given code, use only flag based conditional jumps for conditional processing:

```
while(y > 10)
{
 y = y*2;
 EAX += y;
 --y;
}
```

```
MOV   EAX, Y
MOV   EBX, 2
L1:   CMP   Y, 10
JZ    EX
JC    EX
MUL   EBX
MOV   Y, EAX
ADD   EAX, Y
DEC   Y
JMP   L1
EX:   RET
```

Question No. 3 **[2 x 5 = 10 Points]**

(i) Considering the stack segment starts at **0000 1000h**, draw the stack frame with correct addresses and contents for each of the following procedures (just before de-allocation). Assume the data segment starts at **0000 0000h** and **EBP= 0000 0000** initially. Also write down the **Arr1** after all the code gets executed.

```
.data
      Arr1        BYTE        79h, 19h, 59h, 39h, 79h
```

```
.code
        main PROC

FFC10F  ENTER 0,0
FFC113  MOV EAX, 0
FFC117  MOVZX AX, [arr1+2]
FFC11A  PUSH EAX
FFC11F  INVOKE    P1,    ADDR
        arr1, LENGTHOF arr1
FFC123  POP EAX
FFC125  LEAVE
FFC126  RET
FFC127  main ENDP
```

```
        P1 PROC, x: PTR BYTE,
        count: DWORD
FF727C  ENTER 0, 1
FF727F  MOV ESI, x
FF7382  MOV ECX, count
FF7386  L1: ADD [ESI], 1
FF7389      ROR [ESI], 4

FF738B      ADD ESI, 1
FF738E  LOOP L1
FF7391  INVOKE P2, ESI, count
FF7395  LEAVE
FF7397  RET
        P1 ENDP
```

```
        P2   PROC,   y:   PTR
        BYTE, count: DWORD
00E908F  ENTER 0, 2
00E908C  MOV ESI, ~~*~~ y
00E9089  MOV ECX, count
00E9085  L1: SHR [ESI],1
00E9081      SUB ESI,1

00E907E  LOOP L1
00E907B  LEAVE
00E9070  RET
         P2 ENDP
         END main
```

0000 1000h	RET ADDRESS	*;Ret: address to system*	**Main's Stack frame**
0000 0FFC	0000 0000	*; EBP's Initial Value*	
0000 0FF8	0000 0059	*;Value of EAX*	
0000 0FF4	0000 0000	*;Addr of Arr1 (x)*	**P1's Stack Frame**
0000 0FF0	05	*;Length of Arr1 (count)*	
0000 0FEC	00FF C123	*;Ret: Address to main*	
0000 0FE8	0000 0FFC	*;EBP's Previous: Value*	
0000 0FE4	0000 0004	; Value of ESI (y)	**P2's Stack Frame**
0000 0FE0	05	; count	
0000 0FDC	FF7395	;Ret Address to P1	
0000 0FD8	0000 0FE8	;EBP's Previous Value	

Resulting **Arr1:**

```
Arr1    BYTE  04, 01, 03, 02, 04
```

(ii) Suppose an OCR machine reports errors using an error byte called **status_byte**. The meaning of different bits of status byte is shown below, a respective error message is displayed if a bit turns **ON**:

Bit	Message	Meaning
0	Short document	The document just read is shorter than anticipated
1	Long document	The document just read is shorter than anticipated
2	Close feed	Current document is too close to the preceding document
3	Multiple feed	Two documents were detected at the same time
4	Excessive skew	The document is skewed (crooked) in the transport
5	Document misfeed	The document fails to feed into the transport
6	Document jam	The document jammed in the transport
7	Unspecified error	An unknown/unspecified error occurred

Now write a procedure that should check each bit of the status byte and print the respective error message.

```
MOV AL, status_byte
MOV ECX, 8
MOV EBX, 0
MOV BL, 01 ;                    0000 0001
L1:     TEST AL, BL
        JZ NEXT
        MOV EDX, STR1[EBX]          ;str1 contains strings, error 0 and error 1 have same meaning.
        CALL WriteStr
next:   SHL BL, 1
LOOP L1
```

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