National University of Computer and Emerging Sciences

Islamabad Campus

Computer Organization and Assembly Language (EE2003)

Course Instructor(s):
Mr. Farrukh Bashir, Mr. Shams Farooq

Section(s): All Sections

Sessional-II Exam

Total Time (Hrs): 1
Total Marks: 85
Total Questions: 4

Date: Nov 5, 2024

_____ _____ _____

Roll No **Course Section** **Student Signature**

Do not write below this line.

Attempt all the questions.

INSTRUCTIONS

- Your final answer should be written with permanent pen, Attempt with pencil is unacceptable.

	Q-1	Q-2	Q-3	Q-4	Total
Marks					
Total	30	15	20	20	85

Question 1 [10+10+10=30 Marks]

i. Consider the given C code & convert it to equivalent assembly code. Your program should pass **Parameters through stack** and create **local variable** in swap PROC.

	C Code	Assembly

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
	`void swap(int *a,int *b)`
	`{`
	`  int temp=*a;`
	`  *a=*b;`
	`  *b=temp;`
	`}`
	`main()`
	`{`
	`  int a=5,b=6;`
	`  swap(&a,&b);`
	`}`
		`.data`
		`a DWORD 5`
		`b DWORD 6`
		`.code`
		
		`  push OFFSET b`
		`  push OFFSET a`
		
		`  call swap`
		
		`swap PROC`
		
		`  push ebp`
		`  mov ebp, esp`
		
		`  sub esp, 4`
		
		`  ; temp = *a`
		`  mov eax, [ebp+8]  ; Get pointer to 'a' (first argument)`
		`  mov eax, [eax]  ; Dereference to get *a`
		`  mov [ebp-4], eax  ; Store *a into temp (local variable)`

```
    ; *a = *b
    mov eax, [ebp+12]    ; Get pointer to 'b'
(second argument)
    mov eax, [eax]       ; Dereference to get
*b
    mov ecx, [ebp+8]     ; Get pointer to 'a'
    mov [ecx], eax       ; Store *b into *a

    ; *b = temp
    mov eax, [ebp-4]     ; Load temp
    mov ecx, [ebp+12]    ; Get pointer to 'b'
    mov [ecx], eax       ; Store temp into *b


    mov esp, ebp         ; Restore stack
pointer
    pop ebp              ; Restore base pointer
    ret 8                ; Return from
procedure
swap ENDP
```

ii. Run the following instructions and update registers and **Carry flag (CF)**

NOTE: *Answer should be in* ***Hex****. Process data in Binary.* ***(2 marks for each shift &rotate instruction)***

```
CLC
MOV ax,0BA7DH
SAR ax,3
STC
MOV al,0ACH
SAL al,1
CLC
MOV ax,0A6B7H
ROL ax,4
STC  ;set carry
MOV al,01101001b
RCR al,2
CLC;clear carry
MOV ax,7321o
RCL ax,1
```

	Register	Rough work	CF
CLC	CLC		0
AX	0BA7DH		
AX	0F74FH		
STC			1
AL	0ACH		
AL	058H		
CLC			0
AX	0A6B7H		
AX	06B7AH		
STC			1
AL	069H		
AL	0DAH		
CLC			0
AX	0ED1H		
AX	01DA2H		

iii. Update flags after every **CMP** instruction and calculate jumps

```
MOV ax,2
CMP cx,-2
JA L1
JNBE L2
JNLE L3
JG L4
```

	TAKEN	NOT TAKEN
JA		1
JNBE		1
JNLE	1	
JG	1	

CF	OF	AUX	SIGN	PARITY	ZERO
1	0	1	0	1	0

Subtraction to Calculate Flags for unsigned numbers

0000 0010 (2)

1111 1110 (-2)

-

0000 0100

2's Complement addition to calculate flags for signed number

0000 0010 (2)

0000 0010 (-2, with 2's complement)

+

0000 0100

Question 2 [5+5 +5= 15 Marks]

i. Convert the following **if condition** to equivalent Assembly code using conditional jumps

C Code	Assembly
	MOV AL, 120 ; Load AL with initial value 120

`int al=120;`
 	
`if(al<127 && al>-55)`
`{`
`  al=255;`
`}`
 	
`else`
`{`
`  al=-128;`
`}`
	`CMP AL, 127  ; Compare AL with 127`
	`JAE SKIP  ; Jump to SKIP if AL >= 127`
	
	`CMP AL, -55  ; Compare AL with -55`
	`JLE SKIP  ; Jump to SKIP if AL <= -55`
	
	`MOV AL, 255  ; Set AL=255 if condition true`
	`JMP END_IF  ; Skip SKIP block`
	
	`SKIP:`
	`  MOV AL, -128 ; Set AL=-128 if condition false`
	
	`END_IF:`
	`  ; Exits after this`

ii. Convert the following Assembly code equivalent **C Code.**

Assembly	**C Code**

`mov bx,10`
`mov ax,20`
`L1:`
`  cmp ax,5`
`  jbe L2`
`  dec bx`
`  sub ax,3`
`  jmp L1`
`L2:`
	`int bx = 10;`
	`int ax = 20;`
	
	`while (ax > 5) {`
	`  bx = bx - 1;`
	`  ax = ax - 3;`
	`}`

iii. SHL instruction performs unsigned multiplication when the multiplier is a power of 2 and any other number can be expressed in powers of 2. Write instructions to find the product of **AL** by **29**, where **AL=4.** *You are not supposed to use any **MUL instruction**.*

```
; AL = AL x 29
= 2^4 + 2^3 + 2^2 + 2^0
mov bl, al
shl bl, 4

mov cl, al
shl cl, 3

mov dl, al
shl dl, 2

add al, bl
add cl, dl
add al, cl
```

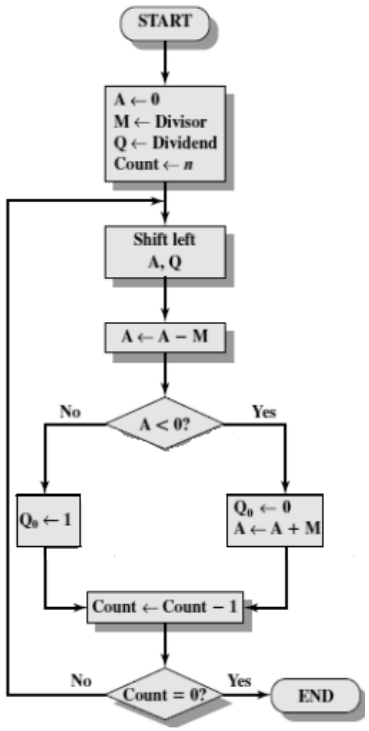
Question 3 [15+5=20 Marks]

Divide $(10011)_2/(11)_2$ using unsigned division. Check your answer by performing binary division.

Division	**FLOW CHART**

M	00011	- M	11101		
	A		**Q**	**Steps**	**Count**
	00000		10011	Initialize	5

00001 11101 ----------	0011?	SHL A - M
11110 00001	00110	A < 0 (YES) A + M COUNT = 4
00010 11101 ----------	0110?	SHL A - M
11111 00010	01100 01100	A < 0 (YES) A + M COUNT = 3
00100 11101 ----------	1100?	SHL A - M
00001	11001	A < 0 (NO) Q_0 = 1 COUNT = 2
00011 11101 ----------	1001?	SHL A - M
00000	10011	A < 0 (NO) Q_0 = 1 COUNT = 1
00001 11101 ----------	00011?	SHL A - M
11110 00001	00110	A < 0 (YES) A + M COUNT = 0



Answer Check By Binary Division

```
          0 1 1 0 (quotient)
      ------------------
11  ) 1 0 0 1 1
      0 0
      -----
        1 0 0
        0 1 1
        -----
          0 0 1 1
            1 1
            -----
          0 0 0 0 1
          0 0 0 0 0
          ---------
          0 0 0 0 1(remainder)
```

Quotient	**00110**	**Remainder**	**00001**

Question 4 [10+7+3 = 20 Marks]

Carefully dry run the given program and show complete traces of runtime stack for both fill and clear phases by writing the actual offset. Also reflect changes to register and memory after every instruction Where memory starts at **0xB4000H** and stack starts **0x00F64H**

```
1
2     .data
3
4      ary db 12h,34h,56h
5     .code
6
7     function Proc
8        push ebp
9        mov ebp,esp
10       push ecx
11       push esi
12       push eax
13       mov esi,[ebp+12]
14       mov ecx,[ebp+8]
15
16       mov al,[esi]
17       ror al,4
18       mov [esi],al
19       inc esi
20       dec ecx
21       cmp ecx,0
22       je exit
23
24       push esi
25       push ecx
26       call function
27       exit:
28
29       pop eax
30       pop esi
31       pop ecx
32       pop ebp
33      ret 8
34    function Endp
35
36    main PROC
37       mov eax,0
38       push offset ary
39       push lengthof ary
40       call function
41       mov eax,0
42    INVOKE ExitProcess,0
43    main endp
      END main
```

Address	Value
00F10	Eax (43)
00F14	Esi B4002
00F18	Ecx (1)
00F1C	Ebp 00F38
00F20	Ret 28
00F24	Ecx (1)
00F28	Esi B4002
00F2C	Eax (21)
00F30	Esi B4002
00F34	Ecx (2)
00F38	Ebp 00F54
00F3C	Ret 28
00F40	Ecx (2)
00F44	Esi B4001
00F48	Eax (0)
00F4C	Esi B4001
00F50	Ecx (3)
00F54	Ebp -
00F58	Ret 42
00F5C	Ecx -
00F60	Offset B4000
00F64	

EAX	**0**	**12**	**21**	**34**	**43**	**56**	**65**							
EBP	**-**	**00F54**	**00F38**	**00F1C**	**00F1C**									
ESI	**4000**	**4001**	**4001**	**4002**	**4002**	**4003**								
ECX	**3**	**2**	**2**	**1**	**1**	**0**								

MEMORY					

ROUGH WORK