


Name: _____ Roll Number: _____ Section: _____

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Computer Organization and Assembly Language	Course Code:	EE2003
	Program:	BS (CS, DS)	Semester:	Fall 2021
	Duration:	3 hours	Total Marks:	100
	Paper Date:	7-Jan-2022	Weightage:	45
	Section(s):	All	Page(s):	12
	Exam:	Final exam	Section:	_____
			Roll No:	_____

Instruction/Notes:

- Exam is Open book, Open notes.
- Properly comment your code.
- You **CANNOT** use an instruction **NOT** taught in class.
- If there is any ambiguity, make a reasonable assumption. Questions during the exam are not allowed.
- Write your answer in the space provided. You **can take extra sheets BUT they WON'T BE ATTACHED WITH THE QUESTION PAPER OR MARKED.**
- All other rules pertaining to examinations as per NUCES policy apply.

Question 1 [45 Marks]:

- i. (4 marks): What are the effective and physical addresses generated by the following memory access?

Memory access	Effective Address	Physical Address
[cs: bx + di]	011Fh	1122Fh

Given: BX=00FFh, CS = 1111h, DS = 3333h, SS = 2526h, IP = 1232h, SP = 1100h, and DI = 0020h

Show your working here:

Effective Address = bx + di = 011Fh

Physical Address = CS * 10h + Offset (Effective Address) = 11110h + 0011Fh = 1122Fh

Name: _____ Roll Number: _____ Section: _____

- ii. (6 marks): The assembly code is provided in method 1 (column 1) to calculate the sum of all the elements of an array? Optimize (with respect to number of lines) this program by using displacement addressing modes.

<u>Method 1: increment bx to advance to each value</u>	<u>Method 2: use bx with displacements to access each value</u>
<pre> List db 10h, 20h, 30h, 40h sum db 0 mov bx, List mov al, [bx] ; AL = 10h inc bx ; BX points to 20h add al, [bx] ; AL = 30h inc bx ; BX points to 30h add al, [bx] ; AL = 60h inc bx ; BX points to 40h add al, [bx] ; AL = 0A0h mov si, sum ; SI points to sum mov [si], al ; SUM = 0A0h </pre>	<pre> mov bx, List mov al, [bx] ; AL = 10h add al, [bx+1] ; AL = 30h add al, [bx+2] ; AL = 60h add al, [bx+3] ; AL = 0A0h mov [bx+4], al ; store sum in next memory location (sum) </pre>

- iii. (6 marks): Suppose that AX=0x3412, BX=0x7856, CX= 0x1CAB, and SP=0x100. Give the contents of AX, BX, CX, and SP after executing the following instructions:

	AX	BX	SP
push ax	0x3412	0x7856	0xFE
push bx	0x3412	0x7856	0xFC
xchg ax, cx	0x1CAB	0x7856	0xFC
pop cx	0x1CAB	0x7856	0xFE
push ax	0x1CAB	0x7856	0xFC
pop bx	0x1CAB	0x1CAB	0xFE

Name: _____ Roll Number: _____ Section: _____

- iv. (7 marks): AX contains a number between 0-15. Write code to complement the corresponding bit in BX. For example, if AX contains 6; complement the 6th bit of BX. (Note: First bit in BX is at 0th position and last bit is at 15th position).

Hint: Use Bit Manipulation

```
[org 0x0100]
```

```
mov ax,3; suppose ax contains 3
```

```
mov dx,1; initial value for complement with bx
```

```
cmp ax,0; if ax contain 0
```

```
je complement; go for complement 1st bit
```

```
mov cx,ax ;other wise move the value in cx
```

```
shl dx,cx; shift 1 to left for number value in ax
```

```
complement:
```

```
xor bx,dx; perform complement operation
```

```
mov ax,4ch
```

```
int 21h
```

Name: _____ Roll Number: _____ Section: _____

v. (4 marks): Given below the listing file of a code. What is the size of the code i.e. .com file?

```
1                                     [org 0x100]
2
3 00000000 A1[1700]                  mov ax, [num1]
4 00000003 8B1E[1900]                mov bx, [num2]
5 00000007 01D8                      add ax, bx
6 00000009 8B1E[1B00]                mov bx, [num3]
7 0000000D 01D8                      add ax, bx
8 0000000F A3[1D00]                  mov [num4], ax
9
10 00000012 A1004C                   mov ax, 0x4c00
11 00000015 CD21                     int 0x21
12
13
14
15 00000017 0500                     num1: dw 5
16 00000019 0A00                     num2: dw 10
17 0000001B 0F00                     num3: dw 15
18 0000001D 0000                     num4: dw 0
```

Answer: 31 bytes

vi. (4 marks): Find the value of the Carry flag after the execution of the following code.

(a) mov ax, 85h add ax, 92h	CF = 0
(b) mov ax, 15h add ax, 72h	CF = 0

Name: _____ Roll Number: _____ Section: _____

- vii. (7 marks): In the code given below, we are using timer and keyboard interrupts to print a specific count on the screen. As a result of the execution of this code, what will be the changes on the screen?

<pre> [org 0x0100] jmp start seconds: dw 0 timerflag: dw 0 oldkb: dd 0 printnum: ; copy from Listing 9.7 (lines 9-47) kbisr: push ax in al, 0x60 cmp al, 0x36 jne nextcmp cmp word [cs:timerflag], 1 je exit mov word [cs:timerflag], 1 jmp exit nextcmp: cmp al, 0xb6 jne nomatch mov word [cs:timerflag], 0 jmp exit nomatch: pop ax jmp far [cs:oldkb] exit: mov al, 0x20 out 0x20, al pop ax iret ; hint: right shift key's press code = 0x36 ; hint: right shift key's release code = 0xb6 ;(code is continued in the second column) </pre>	<pre> timer: push ax push bx push dx cmp word [cs:timerflag], 1 jne skipall inc word [cs:seconds] mov ax, [cs:seconds] mov dx, 0 mov bx, 2 div bx cmp dx, 0 je skipall push word [cs:seconds] call printnum skipall: mov al, 0x20 out 0x20, al pop dx pop bx pop ax iret start: ; copy from Listing 9.8 (lines 95-113) </pre>
--	---

When the program is executed and the right shift key is pressed, the odd numbered counter starts incrementing on the top right corner of the screen. However, if the right shift key is released, then the counter stops.

Name: _____ Roll Number: _____ Section: _____

viii. (7 marks): You are given a piece of code and information on when specific interrupts occurred or when keyboard was pressed during the execution of this code. Considering the code and the occurrences of interrupts/keystroke as given, write out the sequence in which the **instructions** are executed. Each executable instruction in code is numbered so your answer should be as follow:

Sample answer:

Instructions executed in following order

I11

I6

I10

....

I1	jmp start	<p>Write your Answer here that is the sequence in which instructions executed</p> <p>1</p> <p>14-21</p> <p>9-13</p> <p>2-8</p> <p>22-26</p>
	tickcount: dw 0	
	kbisr:	
I2	in al, 0x60	
I3	cmp al, 0x2a	
I4	jne end	
I5	mov bl, 0x2a	
	end:	
I6	mov al, 0x20	
I7	out 0x20, al	
I8	iret	
	timerISR:	
	;---assume that keyboard was pressed by user at this point	
I9	push ax	
I10	inc word [cs:tickcount]; increment tick count	
I11	mov al, 0x20	
I12	out 0x20, al ; end of interrupt pop ax	
I13	iret	
	start:	
I14	xor ax, ax	
I15	mov es, ax	
I16	cli	
I17	mov word [es:9*4], kbisr	
I18	mov [es:9*4+2], cs	
I19	mov word [es:8*4], timerISR	
I20	mov [es:8*4+2], cs	
	;-----int 8h occurred here	
I21	sti	
I22	mov ax, 20	
I23	mov bx, 15	
I24	add ax, bx	
I25	mov ax, 0x4C00	
I26	int 0x21	

Name: _____ Roll Number: _____ Section: _____

Question 2 [25 Marks]

- i. **(3 marks):** Increasing the number of pipeline stages decreases the clock cycle time. However, give a reason why processors should not have hundreds or thousands of pipeline stages.

To have hundreds or thousands of stages, the processor should have a very high processing frequency, which the current technology may not support. Another reason is that the clock frequency increases from more pipeline stages diminish as the overheads (such as hazards, and latch time) dominate.

- ii. **(4 marks):** Identify all data dependencies in the following code. Fill in the table below for each data dependency you find. For example, if I3 depends on register di from I1, you would write “di from 1”. If there is no data dependency leave the table entry blank.

Instruction	Depends on Register from
I1: mov ax, [bx]	
I2: mov bx, [bp]	bx from 1
I3: add ax, bx	ax from 1, bx from 2
I4: mov [bx], ax	bx from 2 & 3, ax from 1 & 3

- iii. **(4 marks):** Find at least 4 possible data hazards (WAW, RAW, WAR) which may occur in the instructions given for Q2(ii). Use following method to write hazard between two instructions.

RAW: Instruction X & Instruction Y, Instruction X & Instruction Z

WAR: Instruction A & Instruction B, Instruction C & Instruction D

WAW: Instruction X & Instruction L, Instruction M & Instruction N

RAW: I2 & I3, I1 & I4, I3 & I4
WAR: I1 & I2, I1 & I4, I3 & I4
WAW: I1 & I3, I1 & I4, I3 & I4

- iv. **(5 marks):** Assume a pipeline with four stages: fetch instruction (FI), decode instruction and calculate addresses (DA), fetch operand (FO) and execute (EX). Complete the pipeline schedule given below for a sequence of 7 instructions, in

Name: _____ Roll Number: _____ Section: _____

which the third instruction is a conditional branch to instruction 15. The schedule for the first two instructions (i.e., I1 and I2) is already filled out. We will assume that there are no data dependencies and no branch hazard detection mechanism is used.

		Clock Cycle												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction	I1	FI	DA	FO	EX									
	I2		FI	DA	FO	EX								

Solution:

	1	2	3	4	5	6	7	8	9	10
I1	FI	DA	FO	EX						
I2		FI	DA	FO	EX					
I3			FI	DA	FO	EX				
I4				FI	DA	FO				
I5					FI	DA				
I6						FI				
I15							FI	DA	FO	EX

- v. (2 mark): There are 128 blocks in a cache memory, which can store one word each. To which block number does main memory word address 900 would map in the case of a direct mapped cache?

Block number = $900 \% 128 = 4$

- vi. (2 mark): Which replacement policy can be used to update an n-way set associative cache?

Name: _____ Roll Number: _____ Section: _____

Random and LRU

- vii. (5 marks): Map following physical addresses of a RAM to 2-way set associative cache and complete the table given below. Replacement algorithm is Least Recently Used (LRU) and the following block access sequence is used:

0, 24, 0, 3, 24

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
24	0	miss	Mem[0]	Mem[24]		
0	0	hit	Mem[0]	Mem[24]		
33	1	miss	Mem[0]	Mem[24]	Mem[33]	
24	0	hit	Mem[0]	Mem[24]		

Question 3 [30 Marks]: You are required to implement a game Bomb_Diffuser with the following requirements.

Name: _____ Roll Number: _____ Section: _____

- i. Game total time is 1000 timer ticks. **(5 marks)**
- ii. Random 15 characters from (A-0) will be placed on screen at the start of the game (Assume you have a **characterRandom** function that return x and y coordinate in register ah and al. You have to maintain an array of 15 words. When you run rand function first time it gives coordinates for character A, 2nd time it gives coordinates for character B and then so on).
- iii. Out of these 15 characters, 3 characters contains bomb (Assume you have an array of 3 random characters between A-0. You are required name it as **bomb_arr**. Hardcoded but can be any characters so in code you have to find the bomb by traversing this 3-character array)
- iv. Once game have started when a user presses a key other than A-0 nothing will happen on the screen. No need of Scan key comparison in the code just convert scan key to ASCII and use ASCII for checking valid characters in hooked ISR. Assume you have a function named **ScanToAscii** that converts scan key to ASCII and return ASCII in AX. ASCII values A-0 in decimal (65-79) and hex (41h-4Fh).
- v. If user presses key that contains bomb game will end. And an appropriate message along with score will be displayed.
- vi. If user presses key that doesn't contain bomb character, that character will be removed by making it black and 10 points will be incremented in the score. **(points ii-vi have 8 marks)**
- vii. Score needs to be updated live. You have to display timer ticks and score in first row of screen **(4 marks)**
- viii. Game will end in three cases. 1) Timer tick becomes greater than 1000 2) Bomb character pressed 3) Only Bomb Characters left on screen. **(6 marks)**
- ix. On game end you have to clear screen and display score on the screen. **(2 marks)**
- x. Use proper subroutines and stacks. No marks for code without subroutine and stack implementation. Maintain proper flow and declare data properly. **(5 marks)**

Subroutines Required:

Start, clearScreen, startDisplay(initialize the screen with score, time and characters), timerISR (updated time isr), scoreUpdate (updates score when correct character pressed), endScreen (call that when game ended clear screen and display score), kbISR (check valid characters, find bomb, removes valid character. You can also use nested subroutines like bombCheck, removeCharacter, checkBombLeft)

```
jmp start

; Data Declaration

characterArray dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
bombArray db 'fkn', 0
Ascii dw 41h
Score dw 0
Time dw 0

clearScreen: ; code from book

startDisplay: ; push register and maintain stack
              ; print initial Score and time in first row hard coded coordinates
              call printScoreAndTime
              ; get coordinates and set it in array with following code
              Mov bx, 0
              Mov cx, 15
L1:           call characterRand
              Mov [characterArray+bx], ah
              Mov [characterArray+bx+1], al
              Add bx, 2
              ; Push coordinates, attribute and charater to display on screen
              Mov dx, 0
              Mov dl, ah
```

```

        Push dx
        Mov dl, al
        Push dx
        Mov dx, 0x07
        Push dx
        Mov dx, ascii
        Push dx
        ; below function prints character at specific coordinates
        Call printCharacter
        Inc ascii
        Loop L1
        ; pop register and maintain stack

timerISR:  push ax
           Cmp word [cs:Time], 1001
           Je endTimer
           Inc word [cs:Time]
           Push word [cs:Time]
           ; PrintTime prints time ticks at specific coordinates
           Call PrintTime
endTimer:  call endScreen
           mov al, 0x20
           out 0x20, al
           pop ax
           iret

scoreUpdate: ; push register and maintain stack
            Add score, 10
            ; PrintScore prints score at specific coordinates
            Call printScore
            ; pop register and maintain stack

endScreen:  call clearScreen
            ; print score anywhere on screen

kbISR:     ; push register and maintain stack
           in al, 0x60
           mov ah, 0
           push ax
           call scanToAscii
           cmp ax, 65
           jl exit
           cmp ax, 79
           jg exit
           mov bx, ax
           push ax
           ; bombCheck traverse the bombArray and return 1 in ax if bomb found
           call bombCheck
           cmp ax, 1
           je endgame
           push bx
           ; removeCharacter removes character from screen after finding the coordinates
           call removeCharacter

```

Name: _____ Roll Number: _____ Section: _____

```
                ; checkBombLeft traverse the characterArray and return 1 in ax if only bombs left
                call checkBombLeft
                cmp ax, 1
                je endgame
                jmp exit
endgame:        call endScreen
exit:           mov al, 0x20
                out 0x20, al
                ; pop register and maintain stack

start:          ; hook interrupt 9 (keyboard) and 8 (Timer)
                Call startDisplay
                ; unhook interrupt 9 (keyboard) and 8 (Timer)
                ; end program
```