National University of Computer and Emerging Sciences, Lahore Campus

Course:	COAL	Course Code:	EE2003
Program:	BSCS,BSDS,BSR	Semester:	Fall 2023
Duration:	3 Hour	Total Marks:	90
Paper Date:	18-Dec-2023	Page(s):	13
Section:	All	Roll No.	_____
Exam:	Final	Your Section:	_____

Instruction/Notes: **This is an open notes/book exam. Sharing notes and calculators is NOT ALLOWED.** All the answers should be written in provided space on this paper. Rough sheets can be used but will not be collected and checked. In case of any ambiguity, make reasonable assumptions. Questions during exams are not allowed.

Question 1 [CLO 1] [1+1+1+2+5 = 10 Marks]: Answer following questions:

(i) **Which of the following bus is unidirectional?**

a. Data Bus **b.** Address Bus **c.** Control Bus

(ii) **Which of the following instruction can change the value of IP register?**

a. Mov **b.** Call **c.** push

(iii) **Which of the following instruction does not change the flag register?**

a. cli **b.** std **c.** jc

(iv) **What is the last physical address of the segment 0xB432? _0xc431f_____**

Show your working to get credit:

(v)

Content of Memory starting from offset 0x139D is given below. Write the updated memory content after execution of the following code. There is not syntax error in this code.

```
mov bx, 0x319E
mov si, 4
mov cx, 0xAB01
mov [bx], cx
mov al, [0x31A4]
mov [bx+3], al
mov word [bx+si], 10
```

Memory Content before Execution of Code:

12	34	2D	1F	F5	89	9A	BC	A0

Memory Content after Execution of Code:

12	01	AB	1F	BC	0A	00	BC	A0

Question 2 [CLO 2] [2+3+5+5+5 = 20 Marks]:

(i)

Following Code is trying to print '*' on top left cell of display memory with black background and white foreground (0x07). **Complete the code.** (Ascii of '*' is 0x2A.)

```
[ORG 0x0100]
mov ax, 0xb800
mov es, ax

mov byte[es:0], 0x____0x2A

mov byte[es:1], 0x_____0x07_____

MOV  AX, 0x4C00          ; Terminate Program
INT  0x21
```

(ii)

Following code is trying to highlight the first five characters from top left of the video memory. Originally the whole screen is white foreground on blue background, without blinking. After highlight the specified characters should be blinking with red background, rest should remain the same. **Write the mask instruction to accomplish this task.**

```
[ORG 0x0100]
 mov ax, 0xb800
mov es, ax
mov si, 0
mov cx,5

label1:
mov ax, [es:si]

_____xor ah, 0xD0/ or ah,_____ ; Apply required mask in one instruction
mov [es:si], ax
add si,2
loop label1  ;runs the loop cx times

MOV  AX, 0x4C00          ; Terminate Program
INT  0x21
```

(iii)

Suppose when the following program loads, the Buffer is at offset 0x0103 in memory. CS=DS=ES=SS=0x19F5

What will be the content of 'buffer' after execution of code? Explain in one line only.

At what Logical Address (Base:Offset) the above code is starting copying the data?

```
[ORG 0x0100]
jmp start
buffer: times 80 dw 0 ;reserves 80 words in memory by placing zeros.

start:
mov ax, 0xb800
mov ds, ax
mov si,0

mov bx, buffer
mov es, bx
mov di, 0

mov cx, 80
cld

rep movsw

MOV  AX, 0x4C00          ; Terminate Program
INT  0x21
```

(iv) Suppose that AX= 1234h, BX= 5678h, CX = 9ABCh, and SP=0100h. **Write the contents of AX, BX, CX, and SP after executing the following instructions:**

AX: _____

BX: _____

CX: _____

SP: _____

```
PUSH AX
PUSH BX
XCHG AX, CX
POP CX
PUSH AX
POP BX
```

(v) We are required to write a program that disables space key on Dosbox. After running our program if user tries to write anything on dosbox, he should not be able to add space, rest of the keys should run normal with command prompt. Partial code is given; **complete the following program such that it fulfills the required functionality**.

```
[org 0x0100]
jmp start
oldisr: dd 0

kbisr:      push ax
            in al, 0x60
 ;Add your code here (if required)

cmp al, 0x39

jne nomatch
jmp exit

exit:       mov al, 0x20
            out 0x20, al
            pop ax
            iret

nomatch:    pop ax
            jmp far [cs:oldisr]
```

```
start:  xor ax, ax
        mov es, ax

        mov ax, [es:9*4]
        mov [oldisr], ax
        mov ax, [es:9*4+2]
        mov [oldisr+2], ax

        cli
        mov word [es:9*4], kbisr
        mov [es:9*4+2], cs
        sti
;Add your code here (if required)

Unhook:
        mov ax, [oldisr]
        mov bx, [oldisr+2]
        cli
        mov [es:9*4], ax
        mov [es:9*4+2], bx
        sti

        mov ax, 0x4c00
        int 0x21
TSR:
        mov dx, start
        add dx, 15
        mov cl, 4
        shr dx, cl

        mov ax, 0x3100
        int 0x21
```

Question 3 [CLO 4] [10+10+10 = 30 Marks]:
Q3 Part (A) [2x5 = 10 Marks]

Consider the cache has 4 blocks/indexes and we have following memory references:

5, 12, 13, 17, 4, 12, 13, 6, 4

What is the hit ratio for the following caches? Show your working below to get credit.

i. Fully/Simple Associative with FIFO Replacement Algorithm	
ii. Fully/Simple Associative with LRU Replacement Algorithm	
iii. Direct mapping	
iv. 2-Way Set Associative using LRU Replacement Algorithm	

v. Consider a pipelined and a non-pipelined architecture. In both, one instruction completes in 6 stages and each stage takes 2 μs.

Fill-in following information:

	Clock Cycle Time	**Frequency**
Pipelined Architecture		
Non-Pipelined Architecture		

Show your working here:

Q3 Part (B) [2+2+6 = 10 Marks]

Assume we have declared an array in memory of 200 bytes and we have an (fully) associative cache of size 8 bytes and each cache index stores 1 byte of data. **For the following code, answer the questions below:**

```
mov ax, 0
mov bx, 0
loop1: add al, [array1+bx]
add al, [array1+bx+1]
add bx, 1
cmp bx, 100
jne loop1
```

i. **What is the total number of memory references in the code? _____**

ii. **If we have total memory size of 1Mb, how many bits are required for the tag in the cache? _____**

iii. **What is the hit rate for the cache for the above code? Use FIFO replacement algorithm. Assume that the cache is initially empty.**

_____

SHOW YOUR WORKING HERE TO GET CREDIT:

Q3 Part (C) - Pipelining Question *FOR ALL THE SECTIONS EXCEPT BCS-3A, BCS-3B, BCS-3C*:

Let us consider the following decomposition of the instruction processing. All stages take equal amount of time i.e 1μs **Fetch Instruction (FI):** Read the next expected instruction into a buffer. **Decode Instruction (DI):** Determine the opcode and the operand specifiers. **Fetch Operands (FO):** Calculate the effective address of each source operand and fetch each operand from the memory. Operand in registers need not to be fetched. **Execute Instruction (EI):** Perform the indicated operation and store the result if any, in the specified destination operand location. **Write Operand (WO):** Store the result in memory. **Following is a set of instructions. Their implementation through pipelining has some control hazards. <u>You have to solve those hazards by using Branch prediction (Predict taken) method.</u> <u>Also calculate latency and throughput.</u>** Note: Normal execution will be done in case of simple instructions. `Jump instructions calculation is done in the execution phase.	**<u>Set of instructions</u>** **I1:** jmp start **I2:** n1: db 10,20 **I3:** start: mov ax, 1 **I4:** mov cx, 8 **I5:** jcxz l1 **I6:** add ax, cx **I7:** mov bx, 8 **I8:** l1: add cx, bx **I9:** add ax, 4 **I10:** add si, 4

Solution:

Instruction No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Latency: _____

Throughput:_____

Q3 Part (C) - Pipelining Question *<u>FOR BCS-3A, BCS-3B and BCS-3C ONLY</u>*:

Show the Pipeline execution of the code given below. Clearly show the stall AND/OR forwarding where required. In case of forwarding, clearly draw the arrow. Assume you have optimized pipelined MIPS Architecture (with all the hazards control implementation) as we have studied in class.

and r1, r2, r3

sub r4, r1, r2

lw r5, 20 (r1)

lw r6, 30 (r1)

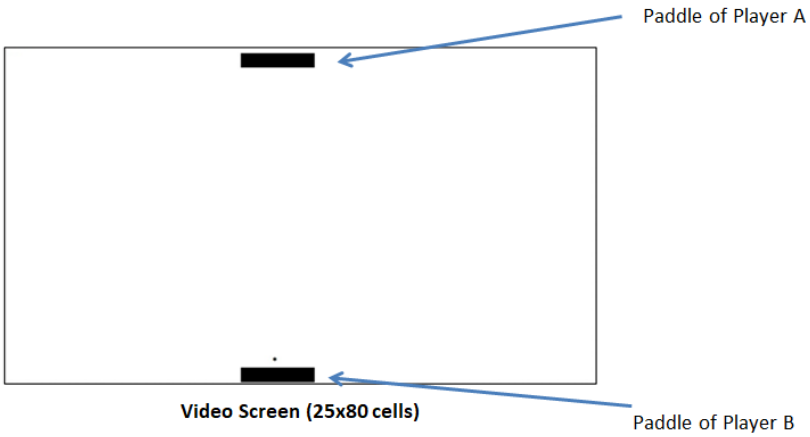
add r7, r8, r6

sub r9, r10, r11

Solution:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12		

Question 4 [CLO 3] [30 Marks]: You are required to implement a game "**PingPong**" in 8088 Assembly Language with following requirements:



1. **[Initial Screen – 5 Marks]** The game will start with screen shown above,
   a. All the screen will be clear (black background, attribute 0x0720, *TO SAVE PRINTER'S INK FIGURE HAS BEEN SHOWN IN INVERTED COLORS*)
   b. First row of the screen (Row no. 0) contains paddle of Player A in the middle. Paddle is a 20 cells wide white space on white background starting from column 30.
   c. Last row of the screen (Row no. 24) contains Paddle of Player B with same requirements.
   d. The boundary wall shown above is just for explanation, you are not required to draw it on screen.
   e. **Ball's Start Position** will be at Player B's side. A White star '*' on Black background at 2nd last row, col 40.
2. **[Ball Movement – 10 Marks]** **After every timer tick,** the ball will move by ONLY one cell diagonally following these rules:
   a) **Ball's Start Direction** will be UpRight Diagonal i.e. initially it will move from Paddle of Player B in UpRight Diagonal (*jump by one row and one column makes diagonal*).
   b) After reaching Player A's row (0), it will bounce back in opposite diagonal as shown in Figure 1.
   c) Every time the ball hits any wall or any paddle, it will bounce back in opposite diagonal within game's boundary. **Due to lack of time we are NOT HANDLING all the movement cases. WE ARE ONLY IMPLEMENTING requirements a, b, d and e in Ball Movement Section and other requirements stated in the question.**
   d) If the ball touches the right wall, reset its position and direction to **"Ball's Start Position and Direction".**
   e) So, the ball will keep moving on the track shown in Figure 1 until the game ends.

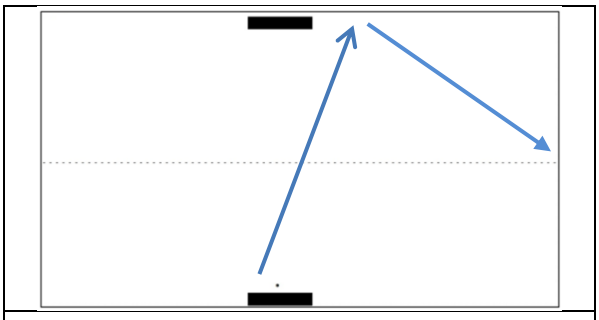


Figure 1
Ball's direction change from UpRight to DownRight diagonal

3. **[Player Turns – 5 Marks]:** After hitting 1st or last row, the turn of opposite player will start. For example, first turn is of Player A when ball is moving from Player B towards Player A. After hitting his row, Player B's turn will start. If ball comes in Start Position it will be Player A's turn again and so on.
4- **[Moving the Paddles – 5 Marks]:** When it is Player A's turn pressing **Right/Left Key** will slide his Paddle towards right or left by one cell according to the key pressed. Similarly Player B's Paddle will slide on **Right/Left Key** if it is his turn. ***Safely assume that you are already given a function SlideRight that takes RowNo as parameter and slides/shifts this row towards right by one cell and it handles all the boundary conditions. YOU ARE NOT REQUIRED TO WRITE THIS FUNCTION, just PROPERLY call it if required. Similarly, a function SlideLeft (with parameter RowNo) is also already given.***

5- **[Scoring – 3 Marks]:** If a Player cannot stop opponent's ball coming to his wall using his paddle then score of the opponent will increase. ***For now, we are only handling Player B's score***.

6- **[Termination – 2 Marks]:** If score of player B reaches 5, the game will terminate. Other programs should run fine on command prompt after termination of your game.

Important Instructions:

- **If you want to use any function from textbook examples, simply call it PROPERLY. YOU DO NOT NEED TO RE-WRITE IT.**
- **If you want to use the code to hook or unhook Keyboard or Timer or make a TSR. Simply write:**
  "---Code to hook Timer comes here---"
  "---Code to make TSR comes here---"
  "---Code to unhook Timer comes here---"
  You do not need to re-write the code but do clearly mention this statement.
- **Properly comment your code and write functions where needed. Freely use global variables, if required.**

```
[org 0x0100]
jmp start
ballRow: dw 21   ; for initial position of Ball
ballCol: dw 40    ; for initial position of Ball
; Your code starts here. Declare other variables if required


[org 0x0100]
jmp start
move: dw 0
oldisr: dd 0
oldtimer: dd 0
turn: dw 0
ballRow: dw 21
ballCol: dw 40
ScoreB: dw 0

tickcount: dw 0
direction: dw 0 ;UpRight
flag: dw 0


;--------------------------------------------------------------------
SlideRight:
push bp
mov bp, sp
pusha



mov ax, 0xb800
mov es, ax
mov ds, ax

mov ax, 80
mul byte [bp+4]
```

```
shl ax, 1

mov si, ax
add si,156
mov di, si
add di,2

std
mov cx, 79
rep movsw

mov word[es:di],0x0720

popa
pop bp
ret 2
;-----------------------------------------------------------------
SlideLeft:
push bp
mov bp, sp
pusha


mov ax, 0xb800
mov es, ax
mov ds, ax

mov ax, 80
mul byte [bp+4]
shl ax, 1

mov di, ax
mov si,di
add si,2

cld
mov cx, 79
rep movsw

mov word[es:di],0x0720

popa
pop bp
ret 2
;-----------------------------------------------------------------
PrintBall:
      pusha
;PRINTING BALL

      mov al, 80                              ; load al with columns per row
```

```
        mul byte [cs:ballRow]           ; 80 x r
        add ax, [cs:ballCol]                    ; word number (80xr) + c
        shl ax, 1                       ; byte no (((80xr) + c)x2)

        mov di, ax                              ; point di to required location


        mov al, '*'
        mov ah, 0x87

        mov ax, 0xb800
        mov es, ax

        mov [es:di], ax

        popa
        ret
;---------------------------------------------------------------------
; subroutine to clear the screen
clrscr:     push es
        push ax
        push cx
        push di

        mov ax, 0xb800
        mov es, ax ; point es to video base
        xor di, di ; point di to top left column ... es:di-->b800:0000

        mov ax, 0x0720 ; space char in normal attribute
        mov cx, 2000 ; number of screen locations

        cld ; auto increment mode
        rep stosw ; clear the whole screen

        ; PRINTING CENTER LINE
        mov al, '-'
        mov ah, 0x07


        mov di, 11*80*2 ; point di to top left column ... es:di-->b800:0000


        mov cx, 80 ; number of screen locations

        cld ; auto increment mode
        rep stosw ; clear the whole screen

        ;PRINTING FIRST PADDLE
        mov di, 30*2
        mov al, ' '
```

```
        mov ah, 0x77

        mov cx, 20 ; number of screen locations

        cld ; auto increment mode
        rep stosw ; clear the whole screen



        ;PRINTING 2ND PADDLE
        mov di, ((22*80)+30)*2
        mov al, ' '
        mov ah, 0x77

        mov cx, 20 ; number of screen locations

        cld ; auto increment mode
        rep stosw ; clear the whole screen


        call PrintBall

        pop di
        pop cx
        pop ax
        pop es
        ret
    ;-------------------------------------------------------------------
    RemoveBall:
        pusha

        mov al, 80
        mul byte [cs:ballRow]
        add ax, [cs:ballCol]
        shl ax, 1

        mov di, ax

        mov ax, 0xb800
        mov es, ax

        mov ax, 0x0720

        mov [es:di], ax

        popa
        ret
    ;-------------------------------------------------------------------
    CheckTurn:
    pusha
```

```
cmp word[cs:ballRow], 0
jne skipturn
mov word[cs:turn], 1
jmp ReturnCheckTurn


skipturn:
cmp word[cs:ballRow], 21
jne ReturnCheckTurn
mov word[cs:turn], 0

ReturnCheckTurn:
popa
ret


;-------------------------------------------------------------------
CheckGoal:
pusha

cmp word[cs:ballRow], 1
jne skipCheckGoal
mov bx,0xb800
mov es,bx

mov ax,0
mov al, 80                      ; load al with columns per row
    mul byte [cs:ballRow]       ; 80 x r
    add ax, [cs:ballCol]                ; word number (80xr) + c
    shl ax, 1                   ; byte no (((80xr) + c)x2)

    mov di, ax                          ; point di to required location

    sub di, 160

    mov ax, [es:di]
    cmp ax, 0x0720 ;if it is paddle
    jne skipCheckGoal
    inc word[cs:ScoreB]

skipCheckGoal:
popa
ret



;-------------------------------------------------------------------
MoveBall:
pusha
```

```
        cmp word [cs:direction], 0
        je UpRight
        cmp word [cs:direction], 1
        je DownRight
        cmp word [cs:direction], 2
        je DownLeft
        mov word[cs:flag],1
        jmp Return

    UpRight:
        cmp word[cs:ballRow], 0
        jne skip
        mov word [cs:direction], 1
        ;mov word[cs:flag],1
        jmp Return


    skip:
        dec byte [cs:ballRow]
        inc byte [cs:ballCol]
        jmp Return

    DownRight:
        cmp word[cs:ballCol], 80
        jne skip2
        mov word [cs:direction], 0
        mov word[cs:ballRow],21
        mov word[cs:ballCol],40
        ;mov word[cs:flag],1
        jmp Return


    skip2:
        inc byte [cs:ballRow]
        inc byte [cs:ballCol]
        jmp Return



    DownLeft:
        cmp word[cs:ballRow], 23
        jne skip3
        mov word [cs:direction], 0
        mov word[cs:ballRow],21
        mov word[cs:ballCol],40
        jmp Return


    skip3:
        inc byte [cs:ballRow]
```

```
        dec byte [cs:ballCol]
        jmp Return

    Return:
    call CheckTurn
    call CheckGoal
    popa
        ret



    ;--------------------------------------------------------------------
    ; TIMER interrupt service routine
    ;--------------------------------------------------------------------
    timer:              push ax

                        inc word [cs:tickcount]; increment tick count
                        cmp word [cs:tickcount], 2 ; is the printing flag set
                        jne skipall ; no, leave the ISR

                        mov word[cs:move], 1
                        mov word[cs:tickcount], 0


    skipall:
                        mov al, 0x20
                        out 0x20, al ; end of interrupt

                        pop ax
                        iret ; return from interrupt
    ;--------------------------------------------------------------------
    ; keyboard interrupt service routine
    ;--------------------------------------------------------------------
    kbisr:              pusha
                        in al, 0x60

                        cmp al, 0x4d
                        je RightKey
                        cmp al, 0x4b
                        je LeftKey

                        jmp exit

    RightKey:
                        cmp word[cs:turn], 0
                        jne abc
                        push 0
                        jmp xyz
```

```
abc:
                    push 22
xyz:
                    call SlideRight
                    jmp exit


LeftKey:
                    cmp word[cs:turn], 0
                    jne abc1
                    push 0
                    jmp xyz1


abc1:
                    push 22
xyz1:
                    call SlideLeft
                    jmp exit

;LeftKey:
;                   push 0
;                   call SlideLeft
;                   jmp exit


exit:               mov al, 0x20
                    out 0x20, al
                    popa
                    iret

nomatch:  popa
                    jmp far [cs:oldisr]


;---------------------------------------------------------------------
sleep:              push cx
          mov cx, 0xFFFF
delay:              loop delay
          pop cx
          ret

start:
          call clrscr

          xor ax, ax
          mov es, ax

          mov ax, [es:9*4]
          mov [oldisr], ax
```

```
                mov ax, [es:9*4+2]
                mov [oldisr+2], ax

                mov ax, [es:8*4]
                mov [oldtimer], ax
                mov ax, [es:8*4+2]
                mov [oldtimer+2], ax

                cli
                mov word [es:9*4], kbisr
                mov [es:9*4+2], cs
                mov word [es:8*4], timer
                mov [es:8*4+2], cs
                sti

                ;JMP $
                        ;call sleep
l1:
                        cmp word[cs:move], 1
                        jne l1
                        call RemoveBall
                        call MoveBall
                        call PrintBall
                        mov word[cs:move], 0
                        cmp word[cs:ScoreB],5
                        je Terminate
skipMove:
                        jmp l1


;l1:                    mov ah, 0                                                       ;
service 0 – get keystroke
;                       int 0x16                                                ; call BIOS
keyboard service

;                       push 22
;                       call SlideRight

;                                               cmp al, 27
                ; is the Esc key pressed
;                       jne l1                                                          ; if no,
check for next key

                        ;jmp $

Terminate:
                xor ax, ax
                mov es, ax

                        mov ax, [cs:oldisr]
```

```
; read old offset in ax
                mov bx, [cs:oldisr+2]
; read old segment in bx

                mov cx, [cs:oldtimer]
; read old offset in ax
                mov dx, [cs:oldtimer+2]
; read old segment in bx

                cli
        ; disable interrupts
                mov [es:9*4], ax
; restore old offset from ax
                mov [es:9*4+2], bx
; restore old segment from bx

                mov [es:8*4], cx
; restore old offset from ax
                mov [es:8*4+2], dx
; restore old segment from bx

                sti
        ; enable interrupts

                mov ax, 0x4c00
; terminate program
                int 0x21
```