Solution.

Roll Number: ______________________ Section: ______________________________

National University of Computer and Emerging Sciences, Lahore Campus

Course:	COAL	Course Code:	EE229
Program:	BS(Computer Science)	Semester:	Fall 2020
Duration:	3 Hours	Total Marks:	110
Paper Date:	22-02-2021	Weight	45%
Section:	All (Your section ____________)	Page(s):	11
Exam:	Final	Roll. No	

Instruction/Notes: This is an open note/book exam. 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, take reasonable assumption. Questions during exam are not allowed. **ATTEMPT ALL QUESTIONS UNLESS GIVEN EXPLICIT INSTRUCTIONS FOR YOUR SECTION. SHARING CALCULATOR IS NOT ALLOWED.**

Question 1: Short Questions [10 x 5 = 50 Marks]

I. **What will be the content of memory (in HEX) before and after the execution of the following code?**

```
[org 0x0100]
jmp start
num1: db 0xA
dw 0x1234
dd 0xABCDEF09

start:      mov ax, [num1+5]       CD
            add ax, [num1+2]   CD+12
            mov [num1], ax
            mov ax, 0x4c00 ; terminate program
            int 0x21
```

Memory Configuration BEFORE Execution:

Address	Num1+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
Content	0A	34	12	09	EF	CD	AB			

Memory Configuration AFTER Execution:

Address	Num1+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
Content	DF	34	12	09	EF	CD	AB			

II. Write Assembly language code for the following
if dx <= cx, ax = 1, else ax = 2

```
cmp dx,cx                        label :
ja label1                            mov ax, 2
mov ax = 1                       end_condition
jmp end_condition
```

Section: ______________________

Roll Number: ______________________

III. Find the values of Sign, Carry, Overflow, and Zero flag given the values of registers and operations.

	CF	OF	SF	ZF
MOV AL , 10 MOV BL, 20 ADD AL, 10 SUB AL, BL	0	0	0	1
MOV AL, 66H MOV BL, 1AH ADD BL, AL	0	1	1	0

IV. A function takes three parameters P1, P2, P3, returns two values Output1, Output2, declares one local variable (local1), and saves AX, BX and CX registers. What will be the configuration of the stack for this function after pushing all these variables and registers? Fill in the given stack. Also specify where BP and SP should be pointing?

Solution of Part (IV)

Sp →	
	CX
	bx
	ax
	LV
bp →	bp
	ret address
	P3
	P2
	P1
	output2
	output1

Stack

V. Fill in the blanks

a. If SP = 1735h and SS:SP = 37B35h, ~~SS~~ SS*16 + 01735 = 37B35h

SS = 0x 3640

b. If SS = FE07h and SS:BP = FF565h, FE070h+BP= FF565h

BP = 0x 1475

c. A parallel port is mapped to interrupt number 0x0F, address of its service routine (handler) can be found at ~~[illegible]~~ [0 : 0 * 0F * 4] 0 : [0 ~~0F~~ * 4 + 2]

d. Timer tick comes 18.2 times per second.

VI. The following code is trying to copy the arrays 0th to 9th element to 1st to 10th elements. For example if array initially 1,2,3,4,5,6,7,8,9,10,11,12 after code runs it should be 1,1,2,3,4,5,6,7,8,9,10,12 But there is a mistake in this code. Identify and correct the mistake. (Hint: Source and Destination are overlapping)

Code with Mistakes	Corrections (Write Correct Lines Only)
00 push ds 01 pop es 02 Mov cx, 10 03 Mov si, array 04 Mov di, array+1 05 cld 06 Rep movsb 07 mov ax, 4c00h 08 int 21h 09 array: db 1,2,3,4,5,6,7,8,9,10,11,12	→ mov si, arr+9 mov di, array+10. → STD.

f registers and

ZF

1

0

(IV)

Section: ____________________

Roll Number: ____________________

VII. **Dry run the following code and find the values of given registers and arrays. Also, write in one line what this code is doing. Assume address of Array is 103h**

```
Jmp start:
    array: db 5,6,3,7,8,9,1
    len_of_array: db 7
start:
    mov cx, [len_of_array]
    push ds
    pop es
    mov si, array
    mov di, array
    cld
    loop1:
       lodsb
       add al, 30
       stosb
       loop
```

Array: 35 36 33 37 38 39 31

SI: 10A

DI: 10A

What is this code doing?

Adding 30 to all elements of array.

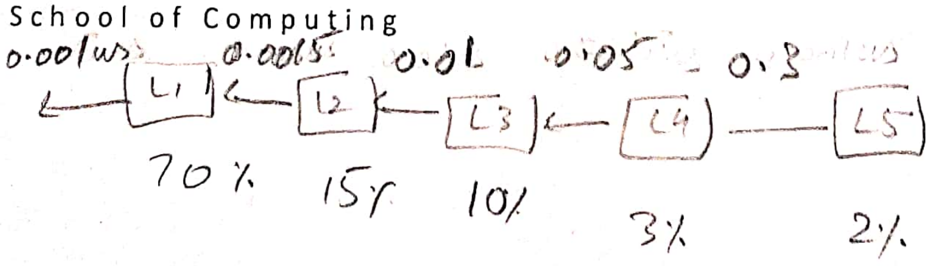
VIII. The following keyboard custom ISR is trying to block the numbers from 0 to 9 from the keyboard, the rest of the keys should work as before. **Complete the code of KBSIR.** (Only write the lines that are to be added in kbISR, don't write the whole code again. Assume that kbsir has been hooked by start and oldisr variable saves the values of old keyboard ISR. DON'T WRITE START CODE)

Code	Additions in Code

`oldisr: dd ; stores  old isr`
`kbisr:`
`    push ax`
`    ;read a char from keyboard`
`     in ax, 0x60; asci in ah and scan code in al`
`    cmp ah, 30h; asci of 0`
`    JL exit`
`    cmp ah,39h ; asci of 9`
`    JG exit`
`    exit:`
`        mov al, 0x20`
`        out 0x20, al`
`        pop ax`
	→ jmp far [oldISR]
	→ iret

IX. [For All Sections Except Section E] Consider 5 cache levels. It takes 0.001us to read from cache L1, 0.0015μs to read from L2, 0.01μs from L3, 0.05 μs from L4 and 0.3 μs from L5. Data is found in L1 70% of time, 15% of the time in L2, 10% of the time in L3, 3% of the time in L4 and 2% of the time in L5. What is the average access time? (μs stands for micro second 1 μs = 10^{-6} s)

$$\Big[(0.001)*70\% + (0.001+0.015)*15\% + (0.001+0.0015+0.01)*10\% + (0.001+0.0015+0.01+0.05)*3\% + (0.001+0.0015+0.001+0.05+0.3)*2\%\Big] / 100 = 0.01145\,\mu s$$



Section: ____________

Roll Number: ____________

Part (IX) [for Section E ONLY] Following program is trying to print the right aligned "This is COAL Final" in the last row of video memory using BIOS service 0x10. Modify the code to produce required output (sample output is shown below). Clearly highlight the lines/segment having errors and write only modified (neat) code in the same space.

Sample Output:

```
                                                            This is COAL Final
```

```
[org 0x0100]
jmp start
message: db 'Hello World'
message2: db 'This is COAL Final'
start:          mov ah, 0x13
                mov al, 1
                mov bh, 0
                mov bl, 7
                mov cx, 11
                mov dx, 0x0A03
                push cs
                pop ds
                mov si, message
                INT 0x10
                mov ax, 0x4c00 ; terminate program
                int 0x21
```

X. [For All Sections Except Section E] For each of the following questions, you are given some information and you are required to find one value. In case the given information is not enough to find the values, write "Info not enough" in answer and specify which info is missing.

	Question	Answer
i.	Given the clock cycle of a pipelined processor is1μs, what is the freq?	1/1us .
ii.	Given the freq of a non-pipelined processor is x what is the throughput of this processor for n instructions?	x
iii.	Given the freq of a pipelined processor is y what the throughput of this processor for n instructions is?	not enough info k & n required
iv.	A pipelined processor with 5 stages, stage 1 to 5 takes 2 μs, 2.5μs, 2.6μs, and 0.5μs, Latch time is 0.1μs. What is the Clock Cycle?	2·6 + 0.1us = 2·7us
v.	Consider the following instructions in a 6 stage pipeline. Assume that there are no data hazards or resource hazards. If the 1st instruction is a conditional branching to I6, how many cycles will be wasted before the branch?	4 cycles are wasted

I1	FI	DI	CO	FO	EI	WO					
I2		FI	DI	CO	FO	EI	WO				
I3			FI	DI	CO	FO	EI	WO			
I4				FI	DI	CO	FO	EI	WO		
I5					FI	DI	CO	FO	EI	WO	
I6						FI	DI	CO	FO	EI	WO

ll Number: ____________ Section: ____________

Part (X) [For Section E ONLY] Consider two different processors P1 and P2 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0.

a. Which processor has better performance expressed in instructions per second? Show your calculations to get credit.

b. If each processor executes a program in 10 seconds, find the number of cycles and the number of instructions.

Solution:

Question 2 [12 Marks]: It takes 10µs to complete one instruction in a non-pipelined processor. We were able to convert the circuit to a 5 stage pipeline processor. Stage 1 to 5 take 2µs, 1.5µs, 3µs, 2µs, 1.5µs resp. Latch time is 1µs. Calculate the following values for pipeline and non-pipelined processor (Write the answer in the given table) **Note for Section E:** Ignore Latch Time.

Value	Non-Pipeline	Pipeline
Clock Cycle	4us = 3us + 1us .	10us .
Frequency (clock speed)	$1/4us$	$1/10us$.
Latency (Time it takes to complete one instruction)	5 * 4us = 20us .	10us .
Throughput for 100 instructions **[For all Section except E]**	$\frac{100}{(100+5-1)*4us} = 0.24$	$\frac{100}{100*10us} = \frac{1}{10us} = 0.1$
Time required to complete 100 instructions **[For Section E ONLY]**		
Speedup of pipeline processor for 1 instruction	$10us/20us = 0.5$	
Speedup of pipeline processor for 100 instructions	$\frac{100*10us}{(100+5-1)4us} = \frac{1000}{104*4} = 2.4$.	

$\frac{100}{(104)\,4us}$

Roll Number: ____________ Section: ____________

Question 3 [for all sections except Section E] [8 Marks]

I1	FI	DI	CO	FO	EI	WO					
I2		FI	DI	CO	FO	EI	WO				
I3			FI	DI	CO	FO	EI	WO			
I4				FI	DI	CO	FO	EI	WO		
I5					FI	DI	CO	FO	EI	WO	
I6						FI	DI	CO	FO	EI	WO

Conflict

Consider the following instructions in the 6 stage pipeline. Assume that there are no data hazards or control hazards. Given the following memory reads are required by different stages of each instruction add the stalls to omit resources hazards. Fill the table given below

- FI of all instructions read from memory.
- FO of only I1, I3, and I6 from memory.
- WO of only I2 and I3, I4 is from memory.

Rest of the FO and WO are from registers and have no conflict.

**Note the FI stands for Fetch Instructions, DI for Decode Instruction CO for Calculate operands, FO for Fetch operands, EI for Execute instruction, and WO for write operands*

I_1	FI	DI	CO	FO	EI	WO												
I_2		FI	DI	CO	FO	EI	WO											
I_3			FI	DI	CO	FO	EI	WO										
I_4				Stall	FI	DI	CO	FO	EI	WO								
I_5						Stall	Stall	Stall	FI	DI	CO	FO	EI	WO				
I_6										Stall	FI	DI	CO	FO	EI	WO		

Section: ______________

Roll Number: ______________

Question 4 - Cache [10+10 = 20 Marks]: Consider a sequence of memory address references given below. In the sequence, each word address is provided in both the decimal and binary formats. Below each address, the relative time at which these references occur is also listed. Memory contents and addresses are shown in the second table.

Data Access Sequence		
Time	**Address Decimal**	**Aaddress Binary**
1	28	00 01 11 00
2	40	00 10 10 00
3	36	00 10 01 00
4	16	00 01 00 00
5	52	00 11 01 00
6	36	00 10 01 00
7	8	00 00 10 00
8	12	00 00 11 00
9	36	00 10 01 00
10	40	00 10 10 00
11	36	00 10 01 00
12	40	00 10 10 00

Memory		
Address Decimal	**Address Binary**	**Data**
8	00 00 10 00	25
12	00 00 11 00	45
16	00 01 00 00	4
20	00 01 01 00	83
24	00 01 10 00	12
28	00 01 11 00	39
32	00 10 00 00	53
36	00 10 01 00	52
40	00 10 10 00	96
44	00 10 11 00	63
48	00 11 00 00	57
52	00 11 01 00	236
56	00 11 10 00	263
60	00 11 11 00	55

Now consider two different 8-word caches shown below. Assume that each of the caches was used independently to facilitate memory access for the sequence above. For each cache type, assume that the cache is initially empty. Assume that the least-recently used (LRU) scheme is used where appropriate. Also, when inserting an element into the cache, if there are multiple empty slots for one index, you should insert the new element into the left-most slot (first available slot).

Part (A): Use the <u>direct-mapped cache</u> to facilitate memory access for the memory sequence above. You should fill in the binary form of the Tag values. **Show the final contents of the cache in the table below, and compute the hit rate and miss rate.**

Index	Cache		
	V	Tag	Data
0 000		(2) 00101 M (4) 00010 M	96/4/25/96
1 001		(7) 00001 M (10) 00101 M (12) H	
2 010			
3 011			
4 100		(1) 00011 M (3) 00100 M	39/52/236
5		(5) 00110 M	52/45/52
6		(6) 00100 M	
7		(8) 00001 M (9) 00100 M (11) H	

all at index 0(000)

all at index 4(100)

Direct-Mapped Cache Summary

Hit Rate: 2/12

Miss Rate: 10/12

Cirled Number shows relative time of access

Number: ______________________ Section: ______________________

rt (B): Use the <u>2-way set associative cache</u> to facilitate memory access for the memory sequence ove. You should fill in the binary form of the Tag values. **Show the final contents of the cache in the table below, and compute the hit rate and miss rate.**

Set	2-way Set Associative Cache					
	V	Tag	Data	V	Tag	Data
00 0	all	① ~~000111~~ M ③ 001001 M	39/52/236/ 25/52		② 001010 M ④ ~~000100~~ M	96/4/52/45/96
01 1	at index	⑤ 001101 M		all	⑥ 001001 M	
10 2	00 set 1	⑦ ~~000000~~ M ⑨ 001001 M	⑪ H	at index	⑧ ~~000011~~ M ⑩ 001010 M	⑪ H
11 3				00 Set 1		

2-way Set Associative Cache Summary

Hit Rate: 2/12

Miss Rate: 10/12

V is 0 for both part as data is not written

Question 5 – Programming [20 Marks]: You are required to implement a game CollectCoins with following requirements:

- A star '*' will keep moving on the screen in Up, Down, Left or Right direction until the game is over, the '*' changes its position after every clock/timer tick.
- Initially '*' will be moving towards right i.e., the '*' will start from a starting position (0, 0) and it will keep moving from the first to last column of first row until the user changes its direction.
- Direction of the '*' can be changed by using Up, Down, Left or Right arrow keys (you may assume any scan codes for these keys).
- Downward movement means the '*' will keep moving from first to last row of same column until user changes its direction.
- <u>Due to time constraint, our game is not supporting Left and Up movement</u>, **so you are NOT required to handle Left and Up movement.**
- Assume there are randomly placed Green and Red cells (coins) on the screen at the start of your game (you are <u>NOT required</u> to place green and red cells on screen). If the '*' crosses a green cell, one point is added to your score and the cell is cleared (i.e. it becomes black). Displaying score on screen is also <u>NOT required</u>.
- If the '*' hits a Red cell, the game is over and it terminates successfully i.e. your program will terminate and DOSBOX and command prompt will run normally.

Important Note: Assume that you are already given a subroutine '*CalculateOffset*' that takes row and col as parameter, calculates position offset and saves it in 'DI' register. You may call it where required, do not write this function again. Instead of writing code to save state of all registers just comment "; pushing all registers here" and similarly just comment for restoring registers. If required, just call functions given in book, do not write those functions again. <u>Variables row, col and direction are already given to handle position and directions, do not use any other variable for this purpose.</u> **Comment your code properly.**

logical important code is witen in blue
assuming calculate-offeset with use global row , colum .

Roll Number: ____________ Section: ____________

```
[org 0x0100]
jmp start
; Use only following variables for position and
; direction
row: dw 0      ; initial position row = 0
col: dw 0      ; initial position col = 0
dir: ~~dw~~ db 'R'   ; initially direction is Right
; declare other variables here (if reqd.)
score: dw 0          oldkbisr: dd 0
terminate: dw 0      oldtimer: dd 0
; Write your start functionality here
start:
 ; Hook KbisR & timer ISR .
  xor ax, ax
  mov ex, ax
  mov ax, [es:9*4]
  mov [old kbisr], ax
  mov ax, [es: 9*4+2]
 mov [old kbisr+2], ax
  cli
 mov word [es : 9*4], kbisr
 mov word [es: 9*4+2], cs
  STI
; for timer same code as above 8
 instead of 9 & oldtimer instead of old kbisr
; check termination .
inf-loop:
       cmp terminate, 1
      jne inf-loop

;unhook .
mov ax, [oldkbisr] .
mov bx, [old kbisr+2]
cli
mov [es : 9*4], ax
mov [es : 9*4+2] bx
sti
[same code for unhooking timer, 8 instead of 9
& oldtimer instead of old Kbisr] .
mov ax, 4c00h
int 21h .
```

```
KBISR:
;Write your KBISR here
 ;initialize

   push ax
    push ds
    push cs
    push ds .

   in   ax, 0x60
   cmp  al , ↓ ;  scan code of down
                  ~~ascii~~ /
   jne  cmp_right
   mov byte [dir] , 'D'
   jmp  end_Kbisr

 cmp_right:
   cmp  al , →     scan code of
                   right
   jne  end_kbisr
   mov  byte [dir], 'R'

 end kbisr

     mov  al , 20h
     out  20h , al
     pop  ds
     pop  ax
     iret .
```

code written in blue is the logic building part & will carry most of the marks .

Roll Number: ____________ Section: ____________

```
; Write your Timer routine here
Timer:
;initialize
   push a
   push ds
   pop ds -
   mov ax, b800h
   mov es, ax
; clear current <row, col>
 call calc-offset
 mov [es, di], 0720h
; set new(row, col) based
 on direction
   cmp byte[dir], 'D'
   jne    cmp_right
   inc   [row]
   cmp  [row], 24
   jne  end-set_coordinates
    mov  [row], 0
```

if hits boundry

```
cmp-right
    cmp byte[dir], 'R'
   jne   end-set-coordinates
   inc [col]
   cmp [col], 79
   jne  end-set-coordinates
   mov [col], 0
```

reset if hits Right boundary

```
;Space for your code
 end-set-coordinates:
; check what color is at new [row, col]
; Eq terminate /mov/++score accordingly

 call   calc-offset .
 mov   ax , [es:di]
 cmp  ah , redcolor .
 jne  cmp-green .
  mov  [terminate], 1
  jmp  end_timer
cmp_green:
   cmp  ah, green color
   jne  ~~cmp-black~~ just_move
   inc  [score]
just_move

  mov [es:di], 07*h
                    ↳ asci
                      07 *
end_timer:

  mov al, 20h
 out 20h , al
  pop  a
  iret
```