
بسم الله الرحمن الرحيم

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

## Some Tips that I followed and created during my COAL Journey

- You should have prerequisite knowledge of how data is stored in memory, when to apply little endian, which instructions take 1 operand, and which takes 2. You must also know how flags are changed and how negative numbers are show in assembly.
- While Working with Dry Run Questions, make a Memory Area in the form of the table, after memory has been modified make a new row and show the memory after changes. In this way you will be able to track changes
- For the Function/Sub-Routine Definition, I generally used the following format whenever I have worked with functions

```
functionName:
     pusha          ; Pushes all the Registers
     mov bp, sp     ; Moves SP -> BP
     add bp, 16     ; Since 8 Registers are pushed
     ; Access the Parameters like [bp+2], [bp+4] …
     ; All my other Code is here
     ;
     popa
     ret
```

*Just pass the parameters through push and call it. Isn't more convenient?*

- Whenever working with Stack, Make it first in the form of Column Matrix. This stack works identical with the one you learnt in Data Structures following the LIFO (Last in First Out).
- For every push operation, Stack is decremented by 2. For every pop operation, Stack is incremented by 2.

| **Base Registers** | | **Index Registers** | |
|---|---|---|---|
| BX | BP | SI | DI |

- Only the Following Combinations are Possible with memory addressing
  - Mov ax, [base registers]
  - Mov ax, [index registers]
  - Mov ax, [base +- offset]
  - Mov ax, [index +- offset]
  - Mov ax, [base + index] // BOTH MUST NOT BE SAME
  - Mov ax, [index + base] // BOTH MUST NOT BE SAME
  - Mov ax, [base + index +- offset] // base and index Location can change. But there should maximum 1 base and maximum 1 index register.

- Only BP Register by default segment SS. All other Registers BX, SI, DI have Segment DS. *(Correct me If I am wrong)*
- Mov ax, [segment: ???] Here any segment will come and will be valid, CS,DS,ES,SS
- while(cx != 0 && condition) // This is the Logical Condition that gets checked after each repetition of string instruction. Every string instruction repetition depends on the cx register.
- Before Approaching the Program, First make a Logic of it. Then Try to identify whether your inputs are correct or not. A Common Example could be taken from the Sessional 2 Fall 2024 Paper, where we were asked to swap alternate rows, Some Students just looped over 24 rows, instead only half of them should be looped. No Doubt, everyone has its own logic, I am just asking you to take caution before proceeding. Every Mark Count in Absolute Courses.
- If BP+2 = 1234 and BP+4 = 5678. Always Register will be filled first. Notice it below

| L**DS** AX, [BP+2] | | L**ES** AX, [BP+2] | |
|---|---|---|---|
| AX | **DS** | AX | **ES** |
| 1234 | 5678 | 1234 | 5678 |

- Remember, Whenever During an ISR of a Hardware Interrupt. You must one of the following choices.
  - out 0x20, 0x20 and IRET
    *// Original Code is not run (Keyboard Input does not show by default)*
  - jmp far [cs: oldISR]
    *// Original Code is run (Keyboard Input functionality shows key pressed)*
- Always Always, See the Base System, Usually in Assembly Everything is in Hexa-Decimal, always see the Prefixes. And Give Attention to Detail
- Remember the Formulas
  - Offset Calculation: 160 x (Row Number, 0-24) + 80 x (Column Number, 0-79)
  - IVT: Offset, Segment. For Int 9 hooking, 9h*4 = ISR_Offset, 9h+2 = Segment (CS). The Formula is as:
    InteruptNumber * 4 = Offset,
    InteruptNumber * 4 + 2 = Segment
- There is difference between (ES: b800)
  mov word [es: 0], 0x0741
  mov byte [es: 1], 0x07
  *Notice how the location changed? It's because of little endianness*
- Remember, until unless you want to return something from a Sub-Routine, Clear the Stack through ret n. Because Instructors might even deduct number from this.

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

- 16 by 16 multiplication/division is different from 16 by 8 multiplication/division

| 16 by 8 | | 16 by 16 | |
|---|---|---|---|
| Upper | Lower | Upper | Lower |
| AH | AL | DX | AX |

- Short Jump is 2 Byte Instruction etc. E9 00. Near Jump (towards larger offset) is 3 bytes instruction. E9 00 00. Far Jump (to jumping to another segment) is 5 bytes instruction. E9 00 00 00 00 (E9 Offset Segment)
- Multi-Tasking is all about Manipulation of IRET Instruction by changing the Stack to point to our Task Location. The More you understand Stack and how return address is stored and fetched, the better you will get better at it.
- IVT Table is 1 KB, every interrupt consumes of 4 byte in IVT and there are 256 Interrupts (0x-0xFF or 0-255).
- You can convert a nested for loop into assembly language with LOOP and Stack *(Remember, you can write limited lines of code between them only)*

| **C++ Code** | **Assembly Code** |
|---|---|
| `for(int i = 0;i<100;i++)`<br>`{`<br>`  for(int j = 0;j<90;j++)`<br>`  {`<br>`  for(int k = 0;k<80;k++)`<br>`  {`<br>`  num++;`<br>`  }`<br>`  }`<br>`}` | L1:<br>  push cx<br>  mov cx, 90<br>  L2:<br>  push cx<br>  mov cx, 80<br>  L3:<br>  inc word [num]<br>  loop L3<br>  pop cx<br>  loop L2<br>pop cx<br>loop L1 |

- For Moving to the Same Line But above of the Cursor -160, For moving the next Line +160, For Moving to the next Character +2. For Moving Diagonally upright, -158, For Moving Diagonally upleft -162, Diagonally Bottomright +162, bottomleft +158

*I had tried my best to explain everything as simple as possible. If still unclear, you can drop me a DM and We could together sort it out.*

Best of Luck My Friends for the Upcoming **Finals**

Remember me and your Fellow Peers in your **Prayers.**

**Your Friend: Anonymous** **FAST-NUCES-LHR**

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

## Sample Dry Runs and Objective Type Questions

Suppose CS: 0x1004, AX: 0x01CA, BX: 0xFFFF, SS: 0x00FF, SP: 0x1234, DS: 0x1005
Calculate the Following Instructions Effective Addresses and Physical Addresses

| Instruction | Effective Address | Physical Address | Error, If any |
|---|---|---|---|
| `mov cx, [bx]` | | | |
| `mov bp, sp`<br>`mov cx, [bp]` | | | |
| `mov cx, [bx+bp]` | | | |
| `mov dx, [bx+10+bp]` | | | |
| `mov si, ax`<br>`mov dx, [cs: bp+si+0xD]` | | | |

Show how memory is shown before and after executing each of the following lines. DQ = Quad Word (8 Bytes). Use AFD for better understanding of the code. Also show the Flags after executing each of the instructions

| | CF | SF | ZF | OF | PF |
|---|---|---|---|---|---|
| `Org 100h` | | | | | |
| `jmp start` | | | | | |
| `num1: dq 0x12345910, 124, 420` | | | | | |
| `num2: dq 145810048,` | | | | | |
| `db 0xA,` | | | | | |
| `dw 100_10_100b` | | | | | |
| `num3: dw 0461o` | | | | | |
| `start:` | | | | | |
| `mov ax, [num1+2]` | | | | | |
| `mov bx, [num2+4]` | | | | | |
| `sub ah, bl` | | | | | |
| `add ax, -1` | | | | | |
| `dec bh` | | | | | |
| `mov cl, [num1+4]` | | | | | |
| `neg cl` | | | | | |
| `mov dx, [num3]` | | | | | |
| `mov [num3], cl` | | | | | |
| `mov dh, cl` | | | | | |
| `not dx` | | | | | |
| `add dx, ax` | | | | | |
| `sub ax, dx` | | | | | |
| `neg ax` | | | | | |
| `mov bl, [num1]` | | | | | |
| `mul bl` | | | | | |
| `mov [num1+4], bl` | | | | | |
| `mOv [num2+3], bx` | | | | | |

| | CF | SF | ZF | OF | PF |
|---|---|---|---|---|---|
| `Mov [num3], ax` | | | | | |
| `Mov [num2+0xA], cx` | | | | | |

A Student made a computer program, but there was a problem with it.

Code: mov ax, [bp+12]

If DS: 0x1000, BP: 0xA400

What will be the Effective and Physical Address from the Following Point of Views

| **If the Student saw the Code in AFD** | | **If the Student saw the Code in Notepad** | |
|---|---|---|---|
| Effective Address | Physical Address | Effective Address | Physical Address |
| | | | |

Notice how different POVs can change the Addresses

Write minimal Lines of Code to Right Shift a 64-bit Number.

- If a 64-Bit Number is represented by num: dq 0x0123456789ABCDEF
- If a 64-Bit Number is represented by num: dd 0x01234567, 0x89ABCDEF
- If a 64-Bit Number is represented by num: dw 0x0123, 0x4567, 0x89AB, 0xCDEF
- If a 64-Bit Number is represented by num:db 0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF

Write 3-4 Lines Code to read the data into AX Register located at Physical Address 0x0FFFF.

Write 2 Programs, 1 that shows Segment Wraparound and 2nd shows Address Wraparound.

16 Address Lines can address how many KB of Memory. 21 Address lines can address how many KBs of Memory. 28 Memory Lines can address how many MBs of Memory?

A Listing File has been provided to you, You have to answer the following questions.

```
   4                                start:
   5          31C0                  xor ax, ax
   6 00000002 31C0                  xor ax, ax
   7 00000004 31C0                  xor ax, ax
   8                                jmp end1
   9 00000009 31C0                  xor ax, ax
   (Some Lines Skipped)
 242 000001DB 31C0                  xor ax, ax
 243 000001DD E9                    jmp start
 244 000001E0 E9A200                jmp h1
 245                                dd: 12345678
```

- What is the Offset of the start label?
- What is the offset of the instruction jmp end1
- What is the Machine Code of jmp end1?
- On which offset will jmp h1 land on?
- Complete the jmp start Machine Code.
- Write the Offset of the Double Word (DD) Data Label
- Write the Machine Code of the Double Word (DD) as well
- Can you identify which type of jumps have been used in the program?

Identify the Wrap Around (See 2.7). Also write the Effective and Physical Address

| Instructions | Effective Address | Physical Address | Wraparound |
|---|---|---|---|
| `push 0xffff`<br>`mov bx, 0x200`<br>`pop cs`<br>`mov cx, [cs: bx]` | | | |
| `Push 0xffff`<br>`Pop bx`<br>`Mov cs, bx`<br>`Mov cx, [cs: bx+40]` | | | |
| `push 0xffff`<br>`pop bx`<br>`mov cx, [cs: bx+40]` | | | |

Notice the Difference and How Wrap around Occurs?

If you make a program that doesn't modify segment registers. Will there be overlapping segment registers?

```
mov [num], ax        ; It assembles successfully
mov [num], 0         ; Why doesn't it assemble correctly?
```

What is the size of a segment from 0-FFFF? that can be accessed by the user at once?

Can IP Register can be modified? If yes, then how?

Considering Pair of Segment: Offset. If Memory is 2F2F:002F we want to access

- What will be the offset if segment is 2F00?
- What will be the segment if the offset is 100A?
- Write another pair other than the above, that read the same memory as of 2F2F:002F

What is the Difference between JA (Jump if Above) and JG (Jump if Greater than)?

What is the Difference between CMP and TEST Instruction?

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

What's the difference between SAR and SHR?

What's the difference between SHL and SAL?

Write 2 Instructions that can

- 1st Instruction can convert any even number into odd number
- 2nd Instruction can convert any odd number into even number

Write 2 Line of Instructions that should only jump if the number is even, otherwise not. But you are not allowed to use TEST or CMP.

There are 2 Ways of Taking 2's Complement, what are they?

Convert a negative number to positive number in 1 or 2 Instruction. Bonus for 1 Instructions

Suppose a Data label is dd: 0x12345678, write a Program that converts it to 0x87654321.

| Instruction | Which Flags can be modified |
|---|---|
| MOV | |
| ADD | |
| SUB | |
| MUL | |
| DIV | |
| CMP | |
| TEST | |
| RCR | |
| ROR | |
| ROL | |
| SAR | |
| SHL | |
| INT | |
| CALL | |
| CALL FAR | |
| STI | |
| CLI | |

| | |
|---|---|
| RET | |
| RETF | |
| IRET | |

What is the Difference between jmp, jmp far, call, call far?

If SP: 0x10F0, AX: 123, BX: 3004, BP: 0x10EC. Write the Value of AX, BX, SP and BP after executing each of the following lines of code

| **Instructions** | **SP** | **AX** | **BX** | **BP** |
|---|---|---|---|---|
| push bp | | | | |
| push ax | | | | |
| push bx | | | | |
| push bp | | | | |
| push ax | | | | |
| pop bp | | | | |
| pop sp | | | | |
| pop bx | | | | |

What will be the Content of the Stack after executing the following lines of code

```
jmp start
func:
    ret 2
start:
    push 10
    push 100
    call func
```

```
jmp start
func:
    ret 1
start:
    push 10
    push 100
    call func
```

| What does the Following Program do? | How would do the stack look like during the execution? |
|---|---|
| `jmp start`<br>`cp:`<br>`  mov bp, sp`<br>`  mov bx, [bp+2]`<br>`  mov ax, [bp+4]`<br>`  ret 4`<br>`start:`<br>`  mov ax, 10`<br>`  mov bx, 20`<br>`  call cp` | `jmp start`<br>`cp2:`<br>`  pop bx`<br>`  ret`<br>`cp:`<br>`  call cp2`<br>`start:`<br>`  call cp` |

Where will the alphabet (A) print on the computer screen? after executing the following piece of code

```
push 0xb900
pop es
mov [es: 500], 0x0741
```

What will happen after the execution of the following program

```
jmp start
str1: db "Fizz", 0
str2: db "Buzz", 0
fun1:
     mov cx, 0xFFFF
     push cs
     pop es
     mov di, str1
     mov al, 0
     repne scasb
     inc di
     push di
     mov di, str2
     mov al, 0
     repne scasb
     sub di, str2
     mov cx, di
     mov si, str2
     pop di
     rep stosb
start:
     call fun1
```

- Write a small piece of code that shows your understanding of using LDS and LES.
- What is the Size of the IVT Table in Megabits and Megabyte? And What does it stand for?
- How many interrupts are there and also write its range.
- Write a Sub-routine that can serve multiple purposes, as far call, as an ISR, and as near call as well.

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

What Interrupts should be called after hooking the desired ISR.

| | |
|---|---|
| Jmp start<br>SubTr:<br>Retf<br>Start:<br>Push 0<br>Pop es<br>Mov [es: 36], subTr<br>Mov [es: 36+2], cs | Mov [es: 36*4], subTr<br>Mov [es: 36*4+2], cs<br>Mov [es: 48h], subTr<br>Mov [es: 48h+2], cs |

What are the issues in the Program?

```
jmp start
kbisr:
     retf
start:
     xor ax, ax
     mov es, ax
     mov [es: 8h], kbisr
     mov [es: 8h+2], cs
```

- Write Assembly Instructions to Disable Timer Interupt
- Write Assembly Instructions to Get the Last Pressed Keyboard Key

How to make this Program a TSR without changing the locations of the data label

```
org 100h
; Write your Code Here
mov ax, 0x3100
int 0x21
name: db "Hello World", 0
```

- What TSR is responsible for show previous executed commands with up and down arrow keys?
- Write a Code to Modify the PIT to tick after every 1ms.
- Which Port is responsible for keeping the special number which is the divisor.
- What flags needed to be cleared while hooking an interrupt.
- Write an ISR that terminates the program when the User Presses ESC key, even when its running an infinite loop.
- Modify the Division by Zero ISR to Display your Name instead of Division by Zero Error.

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

- In the Example 10.2 of Multi-Tasking, we only set next Task to execute in +28, But how does the Computer knows to move back to Task0 after the LastTask in the List. Think 🤔
- If the Tasks Limit were 8, What value should be stored in CL every time to be shifted?
- Why is there a dummy in PCB Layout of 10.2?
- In which Sub-routine is the Stack of the New Thread, handled and its space is reserved for? (10.2)
- What Instruction is mainly responsible behind the concept of multi-tasking and enables it.
- Write a Code to increase the speed of multi-tasking.

*Loved it, liked it or It Helped you. Drop me a Thanks in the DM. So, I may know my effort didn't go useless. Best of Luck for COAL Finals.*

## Practice Problems to Practice for COAL and COAL Lab

1. NASA Hired you as a Low-Level Engineer. You somehow passed the Interview. It is your First day on Job. They tasked you to make an assembly program that can perform 64-bit (DQ = 8 Byte) multiplication with an 8-bit number. Will you able to make such a program that would perform such multiplication? (For this Question you will have to understand how extended multiplication works and how are bytes is represented in memory)
2. Write a Generic Sub-Routine that can take parameters. The Pattern of that Sub-Routine should be the same throughout your all future sub-routines.
3. Write a Sub-Routine that takes 2 Parameters, Row(0-24) and Column Number(0-79) and then returns the Offset(0-3998) of the Screen to help you print there.
4. With every Ratio between Fibonacci number, there lies a ratio called Golden Ratio, User will give you the number (N) through Interrupt 21h, And You have to compute the Nth Fibonacci Number.
5. Cryptography, by a Form of Caesar Cipher. You are supposed to create 3 Sub-Routines.

   **takeMessage:** This Sub-routine should take string input and save it into a buffer

   **encryptMessage:** This should add the ASCII of the Buffer by its position, Like for Hello World, H is at 1st Position, so its ASCII will be added by 1, for e, it will be added by 2, If it gets out of range, you just have to wraparound from the start of its ASCII

   **decryptMessage:** This should now decrypt the encrypted buffer by doing the same process in reverse. In Encryption we added the ASCII by its position, In Decryption, we will subtract its ASCII by its position.
6. Now Modify the Above Sub-Routine in a way that

   **takeMessage** will now take another string as password

   **encryptMessage** will encrypt the message by the ASCII of the password, Like Hello World, H will be added by the ASCII of the first letter of Password and so on. If it reaches the end of the password, start again from first letter till the whole buffer is encrypted.

**decryptMessage** will now do the same process but in reverse. But this time. Again, user will be prompted to give a Password. Notice how different Password tend to destroy the meaning of the message.

7. Write a 3 Sub-Routines. 1st Sub-Routine should be called from start, 2nd Sub-Routine should be called from first, 3rd Sub-Routine should be called from second. But the Thing is After executing the 3rd Sub-Routine, instead of coming back to 2nd Sub-Routine, 3rd Sub-Routine should come back to start.

8. Write a Sub-Routine that takes a single Input, and Show the Following Based on the input:

"Negative Number" for Negative Number Input

"Positive Number" for Positive Number input

"Lowercase Alphabet" for Lowercase Alphabets

"Uppercase Alphabet" for Uppercase Alphabets

"Symbol" for Symbols

"Special to Computer" for Characters like Space, Line Feed and etc.

9. Write a Sub-Routine that takes a Date as String (DD-MM-YYYY), returns the day of the week on that date as a string

10. Write a Sub-Routine that takes a String which ends with a 0, Your Sub-Routine should print that string on the screen with and without string instructions.

11. Write a Sub-Routine that should be able to take input in decimals. It should stop receiving input, once a non-decimal character is entered.

12. Write a Sub-Routine that checks if a given string is palindrome or not.

13. Write a Sub-Routine that Takes a Decimal Number as Input and Prints in Octal Form, Hexa-Decimal Form and Binary Too

14. Write a Sub-Routine that manipulates the stack in such a way to return to next address of the original return address

15. Take a Your Name as Input, and Print it in Reverse Without using any Memory or Buffer or Data Labels

16. Write a Sub-Routine that takes a Character and Prints its ASCII on the Computer Screen. Now Take a String and call this again and again and print the ASCII on the Computer Screen.

17. Write a Sub-Routine that prints a rectangle on the computer screen. You should be able to pass its width and height. It should also output an error if it gets out of screen.

18. Write a Sub-Routine that takes a Single Digit Number and Displays it on the screen with the Help of Asterisk (*).

19. Write a TSR that Disables all the Numeric keys on the Keypad. All other keys should be working in normal condition

20. Write a TSR that clears the screen when 2 Keys are pressed at a Time, Ctrl + C. Ctrl and C should be working in normal condition after clearing screen as well.

21. Write a TSR that prints the Current Time on the Top Right Corner of the Screen in this Format (HH: MM: SS). After every second, the colon should disappear and reappear. The other time should remain intact. This timer should remain even after clear screen

22. Write a TSR should Display a Countdown in seconds After Keypress of Enter on the Top Right of the Computer Screen. After the Countdown reaches Zero, it should call interrupt 19h

23. Write a TSR that should Display the Number of Keys Typed so Far on the Top Right of the Computer Screen. The DOSBox should work intentionally as it used to be. The TSR should persist over clear screens and should unhook itself after ESC key is pressed

24. Write a Keyboard Key Checker Program. It should Print the Status of each key. If the key gets Pressed It should Show its status as OK, like this ESC: OK. You are supposed to hook 9h interrupt for this. And display all your keys on the screen. During the testing of the keys. Your Program shouldn't do anything and should exit once ESC key is pressed.

25. Write a Program that should rotate the screen content by 180 degrees. The Program when ran again should rotate the screen again by 180 degrees making its way back to the state it was before. Make Sure to do this using String Instructions

26. Make a Hexa-Decimal Calculator, Takes 2 Hexa-Decimal Operators and a Operand, Perform the Operations and Display the Results and Exit.