Q1: Briefly answer each of the following: **[10 x 2 = 20 Marks]**

I. Briefly discuss Pros and Cons of Register Parameters and Stack Parameters.

- Register Parameters are passed through CPU registers like EAX, EBX, ECX, etc. It is a speedy one but requires extra pushes and pops, restrictions and limited parameters as registers.
- Stack Parameters are passed by pushing them onto the stack. It Supports passing as many parameters as needed since the stack can grow dynamically. It needs to manage the stack explicitly, ensuring correct alignment and cleanup (e.g., adjusting the ESP register after a function call).

II. Explain how a LOCAL directive is Different from ENTER instruction? Give two reasons.

- A Local directive used to declare and allocate local (temporary) variables within a procedure.
- The assembler translates the LOCAL directive into equivalent instructions for reserving space on the stack

- The Enter instruction sets up a stack frame for a procedure by reserving space for local variables and saving the base pointer (EBP).
- Combines the functionality of saving the current stack frame (EBP) and allocating space in a single compact instruction.

iii. Both are invalid, parameters/Local variable address is not static , so can't used ADDR/OFFSET.

```
LEA esi, A1     ; Instruction 1
LEA edi, A2     ; Instruction 2
```

Rubrics: Corrected instruction is 0.5 points each. The answer both are invalid is 0.5 point each.

iv. AX=FFFA

V.

```
MOV EBX,EAX
MOV ECX, EAX
SHL EAX, 5 ; 32
SHL EBX, 2 ; 4
SUB EAX,EBX ; 32-4= 28
SUB EAX, ECX  ; 28-1 =27
```

Rubrics: Shifting values in of 1 point. Two times SUB is of 1 point.

Vi. Support the best statement about LOOPZ:

- ✓ b. LOOPZ instruction executes a block of code repeatedly by checking whether ECX is greater than zero and that Zero Flag is SET.

c. LOOPZ executes a block of code repeatedly by checking only whether ECX is equal to zero

d. LOOPZ instruction executes a block of code repeatedly by checking only whether Zero Flag is SET.

Vii. Explain the difference between CMP and SUB through some working example.
CMP and SUB both subtract the second operand from first operand, CMP doesn't store the results in first operand whereas SUB does.

```
e.g.        MOV  AL, 7
            CMP  AL, 7      ; AL= 7 ZF = 1
            SUB  AL, 7      ; AL = 0   Zf = 1
```

VIII. Elaborate through an example instruction, how does SCASW differ from LODSW?

Consider an array
arr WORD 1234h, 5678h, 9ABCh, 1234h, 5678h

SCASW:

```
mov ax, 1234h               ; Search for the word 1234h
mov esi, OFFSET arr         ; Point ESI to the start of the array
cld                         ; Clear the direction flag
mov ecx, 5                  ; Array has 5 elements
repne scasw                 ; Scan for 1234h
```

LODSW:

```
mov esi, OFFSET arr         ; Point ESI to the start of the array
cld                         ; Clear the direction flag
lodsw                       ; Load the first word (1234h) into AX
lodsw                       ; Load the second word (5678h) into AX
```

IX. Explain the difference between CBW and MOVSX instruction.

CBW:
Sign-extends an 8-bit value in the AL register into a 16-bit value in the AX register.
It Copies the sign bit (the most significant bit of AL) to all bits of AH (the upper byte of AX).

MOVSX:
Moves a signed value from a smaller source operand to a larger destination operand while performing sign-extension.

X. Elaborate through an example, how does PUSHFD differ from PUSHAD?

Example of using PUSHFD

```
pushfd ; Save the EFLAGS register onto the stack
add eax, ebx    ; Perform an operation that might change flags
popfd           ; Restore the original flags from the stack

; Example of using PUSHAD

pushad                  ; Save all general-purpose registers onto the stack
```

```
mov eax, 1234h      ; Modify some registers mov ebx, 5678h
popad               ; Restore all general-purpose registers from the stack
```

Q2: Convert the following independent Assembly Language instructions to Machine Language code – give your answers in hexadecimal (binary answers will not be graded): [5 x 2 =10 Marks]

SOLUTION

I) 0FB70E18

II) 8F072C1F

III) 89D3

IV) FE432C

V) C11703

Q3: Convert the following hexadecimal machine codes to assembly language mnemonics. State what each of the byte fields mean: [5 x 2 =10 Marks]

I.	**03 84 2B 1A**	**ADD AX , [SI +1A2B]**
II.	**2B 1D**	**SUB BX,[DI]**
III.	**F6 D3**	**NOT BL**
IV.	**56**	**PUSH SI**
V.	**0B 0A**	**OR CX,[BP][SI]**

Q4: Write a procedure ConvertBinToDec that takes an array of binary values as an input. You need to convert that binary value into its equivalent decimal value. Call WriteDec/WriteInt procedures are only allowed to display final result. [10 Marks]
consider the following example
input BYTE 0,0,1,0,1,0,0,0 ; Binary inputs given by user
call ConvertBinToDec
call WriteDec ; 40 is displayed

ConvertBinToDec PROC USES ESI ECX EBX **L1:** **movzx eax, byte ptr [esi + ecx]** **and eax, 1** **mul ebx** **mov edx, eax** **call WriteDec** **mov eax, '+'** **call writechar** **mov eax, edx** **add result, eax** **shl ebx ,1** **dec ecx** **cmp ecx, -1** **JnZ L1** **call crlf** **ret** **ConvertBinToDec ENDP**	**Correctness (4 marks) – Proper binary-to-decimal conversion with correct final output.** **Register Usage (2 marks) – Efficient and conflict-free use of ESI, ECX, EBX, and EAX.** **Clarity (1 mark)** **Efficiency (1 mark)** **Correct Indexing (1 mark) – Proper handling and indexing of the binary array.** **Correct Answer (1 mark)** **Total: 10 Marks**
main PROC **mov esi, OFFSET binaryArray** **mov ecx, LENGTHOF binaryArray - 1** **mov ebx , 1** **call ConvertBinToDec** **mov eax, result** **call WriteDec** **call Crlf** **exit** **main ENDP**	

Q5: Write an assembly language procedure that adds the given two 256-bit numbers. Assume a 32-bit architecture. [10 Marks]

```
op1 QWORD 4 DUP(1234567812345678h)
op2 QWORD 4 DUP(8765432187654321h)
```

SOLUTION

```
.code
MAIN PROC
   mov ecx, 8
   lea esi, op1
   lea edi, op2
   lea ebx, sum
   call Extended_Add
exit:
MAIN ENDP

Extended_Add PROC
   pushad
   clc
L1:
   mov eax, [esi]
   adc eax, [edi]
   pushfd
   mov [ebx], eax
   add esi, 4
   add edi, 4
   add ebx, 4
   popfd
   loop L1
   adc word ptr [ebx], 0
   popad
   ret
Extended_Add ENDP
END MAIN
```

Q6: Write an assembly language program to count the number of words in the given string.
[15 Marks]

.data
msg0 BYTE "string_1 Word Count is: ",0
string_1 BYTE "Nothing is worth more than this day",0
--------- -- -------- ------- ------ ---- ----
1 2 3 4 5 6 7
In the above string, collection of characters leading with space is a word. You must use STACK operations to perform the task. [Hint ASCII for space is 20h].

SOLUTION

```
.code
MAIN PROC
mov ecx, lengthOF string_1
dec ecx
mov esi,0
;------ Push all the elements of String_1 into the STACK --------
L1:
push string_1[esi]
Loop L1
;---------------- word Count ----------------
mov ecx, length
L2:
pop eax
cmp eax,20h
jnz FR
inc count
FR:
Loop L2
inc count
;---------------- Display word Count ----------------
mov edx, offset msg0
call WriteString
mov eax, count
call WriteInt
MAIN ENDP
```

Q7: Using String Primitive instructions, write an assembly language program to find common characters from the following Arrays. The common characters are required to be stored in a new array and should be displayed. [15 Marks]

```
.data
Destination BYTE "abcdef",0
Source BYTE "cfqe",0
Result BYTE 4 DUP(?)
```

Solution

```
INCLUDE Irvine32.inc

.data
Destination BYTE "abcdef", 0    ; First array
Source      BYTE "cfqe", 0      ; Second array
Result      BYTE 4 DUP(?)       ; Array to store common characters
ResultEnd   BYTE 0              ; Null-terminator for the Result array

.code
main PROC
   ; Initialize pointers and registers
   mov esi, OFFSET Destination ; ESI points to the Destination array
   mov edi, OFFSET Result      ; EDI points to the Result array
outer_loop:
   lodsb                    ; Load the next byte from Destination into AL
   cmp al, 0                ; Check if end of Destination string
   je done                  ; Exit if AL == 0 (null terminator)

   ; Compare AL with characters in Source
   push esi                 ; Save the current position in Destination
   mov esi, OFFSET Source   ; Reset ESI to start of Source
   mov ecx, LENGTHOF Source - 1 ; Set ECX to the length of Source

inner_loop:
   lodsb                    ; Load the next byte from Source into AL
   cmp al, 0                ; Check if end of Source string
   je restore_outer         ; Exit inner loop if AL == 0

   ; Compare characters
   cmp al, dl               ; Compare current Source character with the AL register
   jne inner_loop           ; If not equal, continue checking the next Source character

   ; If a match is found, store in Result array
   mov al, dl               ; Load the matching character into AL
   stosb                    ; Store AL into the Result array
   jmp restore_outer        ; Break out of inner loop

restore_outer:
```

```
    pop esi                 ; Restore ESI for outer loop
    loop outer_loop         ; Continue with the next character in Destination

done:
    ; Null-terminate the Result array
    mov byte ptr [edi], 0

    ; Display the Result
    mov edx, OFFSET Result  ; Load Result array address into EDX
    call WriteString        ; Display the Result array

    ; Exit program
    call Exit
main ENDP
END main
```

Question 8 **[5+5=10Marks]**

Solution:

```
; Assume that N, A, and B are stored in memory
; Assume the following registers:
;  EAX = N
;  EBX = A
;  ECX = B
WHILE_LOOP:
    ; Check if N > 0 (EAX > 0)
    CMP EAX, 0
    JLE END_LOOP  ; If N <= 0, exit the loop

    ; Check if N != 3 (EAX != 3)
    CMP EAX, 3
    JE ELSE_PART  ; If N == 3, jump to ELSE_PART

    ; Check if N < A or N > B
    ; First, check N < A (EAX < EBX)
    CMP EAX, EBX
    JGE CHECK_B    ; If N >= A, jump to check N > B

    ; If N < A, execute N = N - 2
    SUB EAX, 2
    JMP WHILE_LOOP ; Go back to the top of the loop

CHECK_B:
    ; Now check N > B (EAX > ECX)
    CMP EAX, ECX
    JLE WHILE_LOOP ; If N <= B, go back to the loop
```

```
   ; If N > B, execute N = N - 2
   SUB EAX, 2
   JMP WHILE_LOOP ; Go back to the top of the loop

ELSE_PART:
   ; If N == 3, execute N = N - 1
   SUB EAX, 1
   JMP WHILE_LOOP ; Go back to the top of the loop

END_LOOP:
   ; Exit point for the loop, N <= 0
   MOV N, EAX   ; Store the final value of N
```

Q8 (b)

```
SHL AL,3                    ; a. A0 H
MOV AL,0E4H
SAR AL,3                    ; b. FC H
STC
MOV AL,0ABH
ROL AL,27                   ; c. 5D H
STC
MOV AL,10H
RCR AL,1                    ; d. 88 H
```