**NAME : SUBHAN**

**NU-ID: 22K-4316**

**SECTION: 3F-BCS**

**COAL GRAND ASSIGNMENT**

**Question 1:**

**CODE:**

TITLE My First Program (Test.asm)

INCLUDE Irvine32.inc

.data

array1 sdword 40 , -90, -67 , 98 , 78 , -45 , 0 , 32

array2 sdword 15 , -90, -67 , 10 , 78 , -45 , 0 , 12

string BYTE "count: " , 0

.code

countMatches PROTO, lengthOfArr: DWORD, arr1: PTR SDWORD, arr2: PTR SDWORD

main PROC

mov edx, lengthof array1

invoke countMatches, edx, ADDR array1, ADDR array2

mov edx, offset string

call writeString

call writeDec

call DumpRegs

exit

main ENDP

countMatches PROC USES edx ecx esi edi, lengthOfArr: DWORD, arr1: PTR SDWORD, arr2: PTR SDWORD

mov esi, arr1

mov edi, arr2

mov edx, 0

mov eax, 0

mov ecx, lengthOfArr

l1:

mov edx, [esi]

cmp edx, [edi]

JNE \_else

add eax, 1

\_else:

add esi, 4

add edi, 4

loop l1

ret

countMatches ENDP

END main

**Question 2:**

**CODE:**

include irvine32.inc

.data

num1 qWORD 98761829AB426871h

num2 qWORD 094732987BC21834h

result qword 1 dup(0)

diffrenceStr byte "The difference is found to be: " , 0

sizeOfNum = SIZEOF num1 / TYPE WORD

.code

Extended\_Sub PROC

pushad

clc

L1:

mov eax, [esi]

sbb eax , [edi]

pushfd

mov [ebx] , eax

add esi , 4

add edi , 4

add eax , 4

popfd

loop l1

sbb word ptr [ebx] , 0

popad

ret

Extended\_Sub endp

main PROC

mov esi , offset num1

mov edi , offset num2

mov ebx ,offset result

mov ecx, sizeOfnum

call Extended\_Sub

mov edx, offset diffrenceStr

call writeString

mov esi , offset result

add esi , sizeofNum \* 4

mov ecx, sizeOfNum

l2:

sub esi , TYPE DWORD

mov eax , [esi]

call writeHex

loop l2

call crlf

exit

main ENDP

end main

**Question 3:**

**CODE:**

TITLE My First Program (Test.asm)

INCLUDE Irvine32.inc

.data

string BYTE "Num to find: " , 0

numFind BYTE "Integer Found! Index: " , 0

notFind BYTE "Integer not Found!" , 0

array1 BYTE 24, 45, 87, 22, 56, 33, 11, 10, 20, 46

.code

findInteger PROTO, numToFind: DWORD

main PROC

mov edx, offset string

call writeString

call readDec

mov esi, 0

invoke findInteger, eax

cmp eax, -1

JE \_mainend

mov edx, offset notFind

call writeString

\_mainend:

call DumpRegs

exit

main ENDP

findInteger PROC, toFind: DWORD

mov ecx, lengthOf array1

cmp ecx, esi

JE \_end

mov edx, toFind

movzx ecx, array1[esi]

cmp edx, ecx

JE \_if

add esi, 1

invoke findInteger, edx

JMP \_end

\_if:

mov edx, offset numFind

call writeString

mov eax, esi

call writeDec

mov eax, -1 ; supposing -1 as a condition if number is found

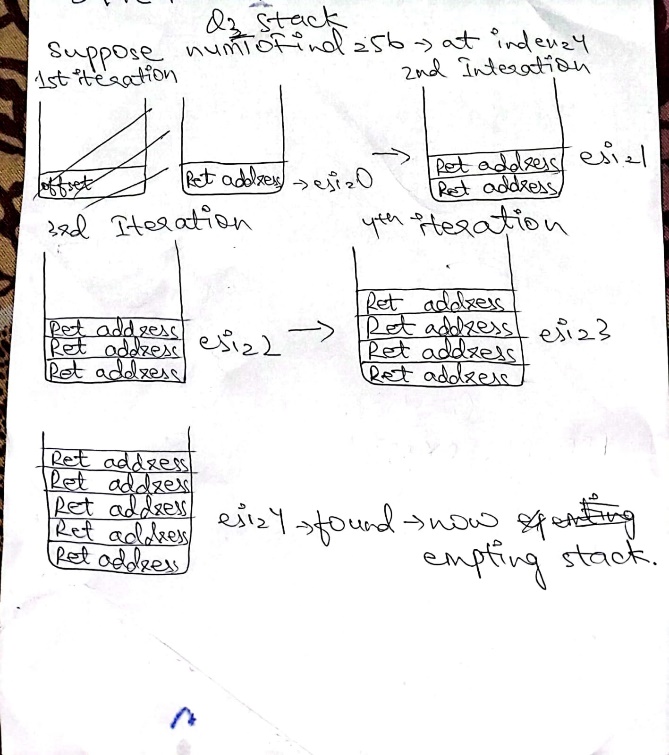
\_end:

ret

findInteger ENDP

END main

**Stack:**



**Question 4:**

**CODE:**

TITLE My First Program (Test.asm)

INCLUDE Irvine32.inc

.data

Find BYTE "consecutives 3 Found!" , 0

notFind BYTE "consecutives 3 not Found!" , 0

array1 BYTE 3, 3, 3, 4, 5, 6

array2 BYTE 1, 2, 3, 4, 5

array3 BYTE 1, 2, 3, 3, 3, 6, 5

array4 BYTE 2, 1, 6, 5, 4, 3, 3, 3

array5 BYTE 1, 2, 4, 5, 6, 7, 8

.code

FindThrees PROTO, arr: PTR BYTE , leng: DWORD

main PROC

invoke FindThrees, ADDR array1, sizeof array1

cmp eax, 1

JNE \_else

mov edx, offset Find

JMP \_end

\_else:

mov edx, offset notFind

\_end:

call writeString

invoke FindThrees, ADDR array2, sizeof array2

cmp eax, 1

JNE \_else2

mov edx, offset Find

JMP \_end2

\_else2:

mov edx, offset notFind

\_end2:

call crlf

call writeString

invoke FindThrees, ADDR array3, sizeof array3

cmp eax, 1

JNE \_else3

mov edx, offset Find

JMP \_end3

\_else3:

mov edx, offset notFind

\_end3:

call crlf

call writeString

call DumpRegs

exit

main ENDP

FindThrees PROC USES esi ebx ecx, arr: PTR BYTE, leng: DWORD

local count: DWORD

mov esi, arr

mov ecx, leng

l1:

mov ebx, 0

mov bl, [esi]

cmp ebx, 3

JNE \_elsel1

add count, 1

cmp count, 3

JE \_loopfinish

JMP \_loopend

\_elsel1:

mov count, 0

\_loopend:

add esi, 1

loop l1

\_loopfinish:

cmp count , 3

JNE \_elseCount

mov eax, 1

JMP \_endCount

\_elseCount:

mov eax, 0

\_endCount:

ret

FindThrees ENDP

END main

**Question 5:**

**CODE:**

INCLUDE IRVINE32.INC

.data

removeStr byte "###ABC" , 0

.code

str\_trims Proc , strRec: ptr byte , charToRemove: byte

local localStr[256]: byte

mov ecx, 256

mov edi , 0

l1:

mov localStr[edi] , 0

add edi, 1

loop l1

mov esi , strRec

mov edi , 0

\_loopStart:

mov bl , charToRemove

cmp [esi] , bl

JE \_dontDo

mov bl , [esi]

mov localStr[edi] , bl

add edi , 1

\_dontDo:

add esi , 1

mov bl , [esi]

cmp bl , 0

JE \_loopFinish

jmp \_loopStart

\_loopFinish:

mov edi , 0

mov esi , strRec

\_ModifiedStr:

cmp localStr[edi], 0

JE \_modifiedStrFinish

mov dl , localStr[edi]

mov [esi] , dl

add esi , 1

add edi , 1

jmp \_modifiedStr

\_modifiedStrFinish:

mov bl , 0

mov [esi] , bl

ret

str\_trims endp

main proc

mov edx, offset removeStr

call writeString

mov al , removeStr[0]

INVOKE str\_trims , addr removeStr , al

mov edx, offset removeStr

call crlf

call writeString

call dumpRegs

exit

Main Endp

End main

**Question 6:**

**CODE:**

INCLUDE IRVINE32.INC

.data

dividend word 0d4a4h

divisor word 0ah

.code

recursiveFunc Proc

mov eax, 0

mov ebx, 0

mov edx, 0

movzx eax, dividend

mov bx, divisor

div bx

cmp ax, 5h

JBE \_finish

mov dividend, ax

call recursiveFunc

\_finish:

ret

recursiveFunc endp

main proc

call recursiveFunc

call dumpRegs

exit

Main Endp

End main

**Question 7:**

1. ADD AL, [BX + SI] -> 0200h
2. INC DX -> FFC2h
3. MOV AX, VAR + 6 ; OFFSET of VAR is 0002H. -> 8B46h
4. SUB CX, VAR2 ; OFFSET of VAR2 is 0008H -> 2B0Eh
5. MOV [SI+490], BX -> 899C9004h

**Question 8:**

1. B9 00 12 -> mov cx, 1200h
2. 8C 85 DC 01 -> mov [DI+ O1DC] , ax
3. 8B 87 56 78 -> mov ax, [bx + 7856h]
4. 28 1D -> sub [di] , bl

**Question 9:**

**CODE:**

INCLUDE IRVINE32.INC

.data

A word ?

B word ?

C1 word ?

enterA byte "Enter value of A: " , 0

enterB byte "Enter value of B: " , 0

enterC byte "Enter value of C: " , 0

.code

main proc

mov edx, offset enterA

call writeString

call readInt

mov A, ax

call crlf

mov edx, offset enterB

call writeString

call readInt

mov B, ax

call crlf

mov edx, offset enterC

call writeString

call readInt

mov C1, ax

; working for part a

mov ax, A

mov bx, 5

mul bx

sub ax, 7

mov A, ax

; working For part b

mov ax, A

sub ax, B

mov cx, B

add cx, 10

mul cx

mov B, ax

; working for part c

mov ax, 9

mov bx, A

imul bx

mov bx, 6

sub bx , ax

mov A, bx

;working for part d

mov ax, A

mov bx, A

mul bx

mov dx, ax

mov ax, B

mov bx, B

mul bx

add dx, ax

mov ax,C1

mov bx, C1

mul bx

cmp dx, ax

JE \_if

jmp \_elseif

\_if:

STC

\_elseif:

CLC

exit

Main Endp

End main

**Question 10:**

**CODE:**

INCLUDE IRVINE32.INC

.data

M sdword ?

N sdword ?

enterN byte "Enter N: " , 0

enterM byte "Enter M: " , 0

gcd byte "Gcd of M and N is: " , 0

.code

main proc

mov edx, offset enterM

call writeString

call readInt

mov M, eax

call crlf

mov edx, offset enterN

call writeString

call readInt

mov N, eax

\_repeat:

mov eax, M

cdq

mov ebx, N

idiv ebx

cmp edx, 0

JE \_finish

mov ebx, N

mov M, ebx

mov ebx, edx

mov N, edx

jmp \_repeat

\_finish:

mov edx, offset gcd

mov eax, N

call crlf

call writeString

call writeInt

exit

Main Endp

End main

**Question 11:**

**CODE:**

INCLUDE irvine32.inc

.data

results dword 67, 45 , 98 , 33

dword 70, 56, 87 , 44

dword 82 , 72 , 89 , 40

dword 80 , 67 , 95 , 50

dword 78 , 76 , 92 , 60

avg DWORD 4 DUP(0)

.code

main PROC

mov edi, 0

mov ecx, 5

mov esi, 0

mov eax, 0

calcAvg1:

add eax, results[esi]

add esi, 16

loop calcAvg1

mov ebx, 0

mov bl, 5

cdq

div ebx

mov avg[edi] , eax

add edi, 4

mov ecx, 5

mov esi, 4

mov eax, 0

calcAvg2:

add eax, results[esi]

add esi, 16

loop calcAvg2

mov ebx, 0

mov bl, 5

cdq

div ebx

mov avg[edi] , eax

add edi, 4

mov ecx, 5

mov esi, 8

mov eax, 0

calcAvg3:

add eax, results[esi]

add esi, 16

loop calcAvg3

mov ebx, 0

mov bl, 5

cdq

div ebx

mov avg[edi] , eax

add edi, 4

mov ecx, 5

mov esi, 12

mov eax, 0

calcAvg4:

add eax, results[esi]

call crlf

call writeDec

add esi, 16

loop calcAvg4

mov ebx, 0

mov bl, 5

cdq

div ebx

mov avg[edi] , eax

;printing avg array

mov ecx, 4

mov esi , 0

print:

mov eax, avg[esi]

call writeDec

call crlf

add esi, 4

loop print

exit

main ENDP

END main

**Question 12:**

**CODE:**

INCLUDE irvine32.inc

; TA Bhai logic dekh lena agar aap ko achi lage tou baki output tou warr gya ha

.data

sampDoc BYTE "this is a sample document ", 0

temp BYTE 100 DUP(" "), 0

testDoc1 BYTE "Test document 1 this is", 0

testDoc2 BYTE "Test document 2 that is", 0

testDoc3 BYTE "Test document 3 this is a", 0

testDoc4 BYTE "Test document 4 that is a", 0

testDoc5 BYTE "Test document 5 is a", 0

checkTemp BYTE 100 DUP(" "), 0

i DWORD 0

matched BYTE "Words matched: ", 0

found BYTE "Plag Found ", 0

foundNot BYTE "Plag don't detected", 0

.code

main PROC

; Initialize variables

mov edx, OFFSET temp

call WriteString

call Crlf

mov ebx, 0

mov edx, 0

mov al, ' '

mov ecx, LENGTHOF sampDoc

\_checkPlag:

push ecx

mov al, sampDoc[ebx]

mov temp[edx], al

cmp al, ' '

je \_checkWord

inc edx

jmp \_next

\_checkWord:

mov edx, OFFSET checkTemp

mov ebx, OFFSET testDoc1

call plagiarismFinder

pop ecx

\_next:

inc ebx

loop \_checkPlag

mov edx, OFFSET matched

call WriteString

mov eax, i

call WriteDec

call Crlf

cmp i, 3

je PlagiarismDetected

PlagiarismDetected:

mov edx, OFFSET found

call WriteString

call Crlf

jmp \_exit

\_exit:

exit

main ENDP

PlagiarismFinder PROC

mov al, ' '

mov ecx, LENGTHOF testDoc1

\_checkingWord:

cmp al, [ebx]

je SF1

mov al, [ebx]

mov [edx], al

inc edx

jmp Moving

SF1:

push ebx

push edx

mov ebx, 0

mov edx, 0

mov al, ' '

mov ecx, LENGTHOF testDoc1

\_loop3:

push ecx

cmp al, [ebx]

je SF2

mov al, [ebx]

mov [edx], al

mov al, ' '

inc edx

jmp \_moving

SF2:

push edx

mov dl, [checkTemp]

cmp dl, [temp]

jne \_nE

inc i

\_nE:

pop edx

mov edx, OFFSET checkTemp

call WriteString

call Crlf

mov edx, 0

mov ecx, 100

\_loop:

mov [checkTemp + edx], al

inc edx

loop \_loop

mov edx, 0

\_moving:

inc ebx

pop ecx

loop \_loop3

pop edx

pop ebx

moving:

inc ebx

pop ecx

dec cx

jne \_checkingword

ret

plagiarismFinder ENDP

END main