Q) The Str_ucase procedure converts a string to all uppercase characters. It returns no value. When you call it, pass the offset of a string:

```
INCLUDE Irvine32.inc
.data
_str BYTE "kinza",0
.code

ConvertToupper PROC USES eax, pstring :PTR BYTE
mov esi, pstring
L1:
mov al, [esi]
cmp al,0 ; null terminator
je L2
cmp al,'a'
jb L3
cmp al,'z'
ja L3
and BYTE PTR [esi],11011111b
L3:
inc esi
jmp L1
L2:
ret
ConvertToupper ENDP
```

```
main PROC
mov edi, OFFSET _str
INVOKE ConvertToupper, edi
mov edx, OFFSET _str
call WriteString
call Crlf
main ENDP
END main
```

```
KINZA
```

Qus 03: Declare and initialize an array with your (Actual) Id and Name stored in it. Use a loop with indexed addressing to swap the elements of array by half. Do not copy the elements to any other array. In the end display the elements from the array. **[Points: 15]**

Hint: Use the SIZEOF, TYPE, and LENGTHOF operators to make the program flexible
Example: (Use your own Id & Name)

1	9	k	-	1	2	3	4		A	Z	A	M		K	H	A	N

Expected Output

N	A	H	K		M	A	Z	A		1	2	3	4	-	k	9	1

This Past paper Q will be solved by the same string reverse code.
inputStr BYTE “23k-0842 KINZA AFZAL”

4. Create a Str_Reverse procedure to reverse strings.

```
TITLE Q4
INCLUDE Irvine32.inc
.data
inputStr BYTE "HELLO", 0
msg BYTE "Reversed string: ", 0
.code
main PROC
mov edx, OFFSET inputStr
call WriteString
call CrLf
push OFFSET inputStr
call Str_Reverse

mov edx, OFFSET msg
call WriteString
mov edx, OFFSET inputStr
call WriteString
call CrLf
exit
main ENDP
```

```
Str_Reverse PROC
    push ebp
    mov ebp, esp

    mov esi, [ebp+8]
    mov edi, esi

find_end:
    cmp BYTE PTR [edi], 0
    je reverse_start
    inc edi
    jmp find_end
```

```
reverse_start:
    dec edi             ; Move back to the last character
reverse_loop:
    cmp esi, edi        ; Check if pointers have crossed
    jge reverse_done    ; If yes, reversing is complete

    mov al, [esi]
    mov bl, [edi]
    mov [esi], bl
    mov [edi], al

    inc esi
    dec edi
    jmp reverse_loop

reverse_done:
    pop ebp
    ret 4
Str_Reverse ENDP

END main
```

```
HELLO
Reversed string: OLLEH
```

Q) IS PRIME

```
isPrime PROC
cmp eax, 2
je prime
jl notprime
mov [count], 0
mov ecx, 2 ;i
mov ebx, eax
L2:
mov eax, ecx
imul eax, ecx      ; Calculate ecx * ecx
cmp eax, ebx
jg check            ; If ecx^2 > number, it is prime
mov eax, ebx
cdq
div ecx
cmp edx, 0 ; if remainder is zero
jne next
inc [count]
next:
inc ecx
jmp L2
```

```
check:
cmp [count], 0
ja notprime
jmp prime

notprime:
mov eax, 1
jmp _ex
prime:
xor eax, eax ; this sets the zero flag

_ex:
ret
isPrime ENDP
END main
```

Q) Str Concat

```
INCLUDE Irvine32.inc
.data
sourceStr BYTE " LOVE",0
targetStr BYTE "COAL IS",SIZEOF sourceStr DUP(0)
.code
Str_concatenate PROC psource: PTR BYTE, ptarget: PTR BYTE
mov esi, psource
mov edi, ptarget
L1:
mov al, [edi]
cmp al, 0 ;null terminator
je startcopyingsource
inc edi
jmp L1
```

```
startcopyingsource:
mov al, [esi]
cmp al, 0; null terminator
je _end
mov [edi], al
inc esi
inc edi
jmp startcopyingsource
_end:
mov BYTE PTR [edi], 0
ret
Str_concatenate ENDP
main PROC
INVOKE Str_concatenate, ADDR sourceStr, ADDR targetStr
mov edx, OFFSET targetStr
call WriteString
main ENDP
END main
```

```
COAL IS LOVE
```

Q) Str_remove Procedure

Write a procedure named Str_remove that removes n characters from a string. Pass a pointer to the position in the string where the characters are to be removed. Pass an integer specifying the number of characters to remove. The following code, for example, shows how to remove “xxxx” from target:

.data
target BYTE "abcxxxxdefghijklmop",0
.code
INVOKE Str_remove, ADDR [target+3], 4

```
INCLUDE Irvine32.inc
.data
target BYTE "abcxxxxdefghijklmop",0
tgt BYTE SIZEOF target DUP(0)
.code
Strremove PROC psource: PTR BYTE, removechar: BYTE
mov esi, psource
mov edi, OFFSET tgt
mov al, removechar
movzx ecx, al
L1:
cmp ecx, 0
je _end
inc esi
dec ecx
jmp L1
_end:
```

```
_end:

L2:
    mov al,[esi]  ; Load current character from source
    cmp al, 0
    je L3
    mov [edi], al  ; Write it to the destination
    inc esi                ; Move to next source character
    inc edi                ; Move to next destination character
   jmp L2
```

```
L3:
 mov BYTE PTR [edi], 0
 mov edx, OFFSET tgt
 call WriteString
call Crlf
mov esi, psource
mov edi, OFFSET tgt
copy:
mov al, [edi]
cmp al, 0; null terminator
je _return
mov [esi], al
inc esi
inc edi
jmp copy
_return:
 ret
```

```
Strremove ENDP
main PROC
INVOKE Strremove, ADDR [target+3], 4
mov edx, OFFSET target
call WriteString
call Crlf
main ENDP
END main
```

```
defghijklmop
abcdefghijklmoplmop
```

Q) Count the sum of digits of a given number:

```
TITLE Count Digits
INCLUDE Irvine32.inc
.data
msg BYTE "The digits in n are: ",0
n DWORD 5000
sum DWORD 0
.code
main PROC
mov edx, 0
mov eax, n
mov ecx, 0
call Countdigits
mov eax, ecx
mov edx, OFFSET msg
call WriteString
call WriteDec
call Crlf
mov eax, [sum]
call WriteDec
exit
main ENDP
```

```
Countdigits PROC
cmp eax, 0
je basecase
mov ebx, 10
div ebx ; eax = eax/10
add [sum], edx
inc ecx ; ecx = count++
call Countdigits
basecase:
ret
Countdigits ENDP
END main
```

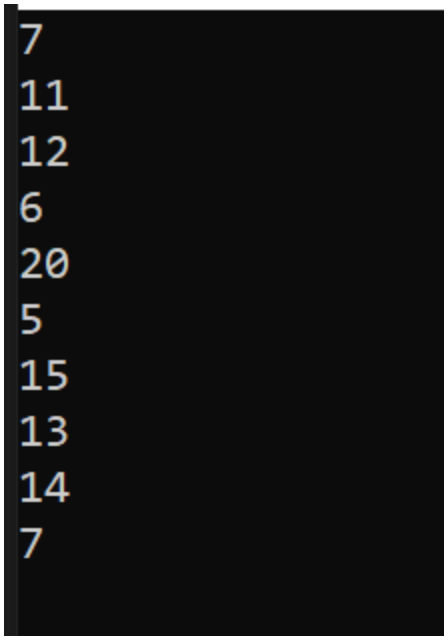
```
The digits in n are: 4
5
```

Q) FILL ARRAY USING RANDOM RANGE

```
TITLE FillArray Procedure
INCLUDE Irvine32.inc
.data
arr DWORD 10 DUP (?)
_count DWORD 10
_upperrange DWORD 15h
_lowerrange DWORD 5h
```

```
.code
FillArray PROC pArray: PTR DWORD, Count: DWORD, UpperRange: SDWORD, LowerRange: SDWORD
mov edi,pArray ; EDI points to the array
mov ecx,Count ; loop counter
mov edx,UpperRange
sub edx,LowerRange ; EDX = absolute range (0..n)
cld ; clear direction flag
L1:
mov eax,edx
call RandomRange
add eax,LowerRange ; bias the result
stosd ; store EAX into [edi]
loop L1
ret
FillArray ENDP
```

```
main PROC
INVOKE FillArray, ADDR arr, _count, _upperrange, _lowerrange
mov esi, OFFSET arr
mov eax, 0
mov ecx, 10
L1:
mov eax, [esi]
call WriteDec
call Crlf
add esi, 4
loop L1
main ENDP
END main
```


7
11
12
6
20
5
15
13
14
7

Questions 2 **[5] Points**

Write a recursive procedure to find a value in a large integer array. Ask the user to enter an integer value in the main program. You should pass user supplied value as parameter to the recursive function using the INVOKE directive. Also, draw labeled diagrams to show stack values at each iteration of this recursive function.

```
TITLE Q2
INCLUDE Irvine32.inc
.data
arr BYTE 10,20,30,40,50
msg BYTE "Enter a number you want to find in array: ",0
tosearch BYTE ?
flagfound BYTE 0
foundmsg BYTE "Value found at index: ",0
notfoundmsg BYTE "Value is not found",0

searchValue PROTO ptrArray:PTR BYTE, _size:DWORD, value:BYTE
```

```
.code
main PROC
mov eax, 0
mov edx, OFFSET msg
call WriteString
call ReadInt
mov tosearch, al
mov esi, OFFSET arr
mov ecx, LENGTHOF arr
INVOKE searchValue, esi, ecx, al
cmp flagfound, 1
je valueFound

mov edx, OFFSET notfoundmsg
call WriteString
jmp _exit

valueFound:
mov edx, OFFSET foundmsg
call WriteString
call WriteDec  ;Print the index stored in EAX

_exit:
exit
main ENDP
```

```
searchValue PROC ptrArray:PTR BYTE, _size:DWORD, value:BYTE
; Load parameters into registers
mov esi, ptrArray
mov ecx, _size
mov al, value

cmp ecx, 0
je notfound
mov bl, [esi]
cmp bl, al
je found
inc esi
dec ecx
INVOKE searchValue, esi, ecx, al
ret
found:
sub esi, OFFSET arr  ; calculate index (ESI - base address)
mov eax, esi        ; Store index in EAX
mov flagfound, 1    ; value found
ret
```

```
notfound:
mov flagfound, 0   ; set flag to 0
ret
searchValue ENDP
END main
```

```
Enter a number you want to find in array: 50
Value found at index: 4
```

Question 6 **[5] Points**

Create a variant of the Str_trim procedure that lets the caller remove all instances of a leading character from a string. For example, if you were to call it with a pointer to the string "###ABC" and pass it the # character, the resulting string would be "ABC".

Logic:

- esi is initialized to point to the source string (string1).
- edi is initialized to point to the destination within the same string (string1).
- al is loaded with the character to be removed (removeChar).
- The current character is loaded into dl from the source string (using esi).
- If the current character is null (cmp dl, 0), it means the end of the string is reached, and the loop exits.
- If the current character matches the removeChar (cmp dl, al), it is skipped by incrementing esi (source pointer).
- If it does not match, it is copied to the destination location (pointed to by edi), and both esi and edi are incremented.
- After processing the string, a null byte (0) is added at the position pointed to by edi to properly terminate the modified string.

```
INCLUDE Irvine32.inc

.data
original BYTE "Original String: ", 0
changed BYTE "Changed String: ", 0
string BYTE "###ABC", 0

.code
strtrim PROTO, string1:PTR BYTE, removeChar:BYTE

main PROC
lea edx, original
call WriteString
lea edx, string
call WriteString
call Crlf
```

```
INVOKE strtrim, ADDR string, '#'

lea edx, changed
call WriteString
lea edx, string
call WriteString
call Crlf

exit
main ENDP
strtrim PROC uses esi edi, string1:PTR BYTE, removeChar:BYTE
mov esi, string1
mov edi, string1
mov al, removeChar

trim_loop:
mov dl, [esi]
cmp dl, 0
je end_trim
cmp dl, al
jne copy_char
inc esi
jmp trim_loop
```

```
copy_char:
mov [edi], dl
inc esi
inc edi
jmp trim_loop

end_trim:
mov byte ptr [edi], 0
ret
strtrim ENDP
END main
```

```
Original String: ###ABC
Changed String: ABC
```

Q) Count occurrences of a specific character in a string

```
INCLUDE Irvine32.inc
.data
_string BYTE "abbaca",0
countchar BYTE 0
.code
Countoccurences PROC pstring: PTR BYTE,count: DWORD, char : BYTE
mov esi, pstring
mov bl, char
mov ecx, count
L1:
mov al, [esi]
cmp al, 0; null terminator
je _end
cmp al, bl
jne next
inc countchar
inc esi
loop L1
next:
inc esi
```

```
next:
inc esi
jmp L1
_end:
ret
Countoccurences ENDP
main PROC
mov ecx, LENGTHOF _string
dec ecx
INVOKE Countoccurences, ADDR _string , ecx, 'z'
mov eax, 0
mov al, [countchar]
call WriteDec
main ENDP
END main
```

Q) CALCULATING THE ROW SUM:

```
INCLUDE Irvine32.inc
.data
tableB BYTE 10,20,30,40,50
Rowsize = $ - tableB
        BYTE 60,70,80,90,10
        BYTE 10,20,30,40,50
rowindex = 0
.code
main PROC
mov ebx, OFFSET tableB
mov eax, rowindex
mov ecx, Rowsize
mul ecx
add ebx, eax
mov eax, 0
mov esi, 0;column index
```

```
L1:
movzx edx, BYTE PTR [ebx+esi]
add eax, edx
inc esi
loop L1
call WriteDec
call Crlf
exit
main ENDP
END main
```

Loop runs 5 times since ecx holds the rowsize which is equal to 5 elements.
Rn since the rowindex is 0 so it sums the first row :
10+20+30+40+50 = 150

```
150
```

Q) CALCULATING THE COLUMN SUM:

```
INCLUDE Irvine32.inc
.data
tableB BYTE 10,20,30,40,50
Rowsize = $ - tableB
       BYTE 60,70,80,90,10
       BYTE 10,20,30,40,50
rowindex = 0
.code
main PROC
mov ebx, OFFSET tableB
mov eax, rowindex
mov ecx, Rowsize
mul ecx
add ebx, eax
mov eax, 0
mov esi, 0;column index
L1:
movzx edx, BYTE PTR [ebx+esi]
add eax, edx
add ebx, Rowsize
loop L1
call WriteDec
call Crlf
exit
main ENDP
END main
```

Rn for 0th index column the sum of columns is: 10+60+10 = 80

```
INCLUDE Irvine32.inc
.data
tableB WORD 1000,2000,3000,4000,5000
Rowsize = 5
       WORD 6000,7000,8000,9000,1000
       WORD 1000,2000,3000,4000,5000
rowindex WORD 0
.code
main PROC
mov ebx, OFFSET tableB
movzx eax, rowindex
mov ecx, Rowsize
mul ecx
add ebx, eax
mov eax, 0
mov edx, 0
mov esi, 1;column index
L1:
mov dx, WORD PTR [ebx+ esi*TYPE tableB]
add eax, edx
add ebx, Rowsize*TYPE tableB
loop L1
call WriteDec
call Crlf
exit
main ENDP
END main
```

2000+7000+2000 = 11000 rn the answer is this.

```
INCLUDE Irvine32.inc
.data
tableB WORD 1000,2000,3000,4000,5000
Rowsize = 5
       WORD 6000,7000,8000,9000,1000
       WORD 1000,2000,3000,4000,5000
rowindex WORD 1
.code
main PROC
mov ecx, 0
mov eax, 0
mov ebx, OFFSET tableB
movzx eax, rowindex
mov ecx, Rowsize
mul ecx
shl eax, 1
add ebx, eax
mov eax, 0
mov edx, 0
mov esi, 0
mov ecx, 5
L1:
mov dx, WORD PTR [ebx+ esi*TYPE tableB]
add ax, dx
inc esi
loop L1
call WriteDec
call Crlf
exit
main ENDP
END main
```

Rn 6000+7000+8000+9000+1000=31000

Row sum for index 1 and row 2.

Q) FREQUENCY TABLE:

```
INCLUDE Irvine32.inc
.data
target BYTE "AAEBDCFBBC", 0
freqTable DWORD 256 DUP(0)
alphabet BYTE "ABCDEFGHIJKLMNOPQRSTUVWXYZ",0

.code
Get_frequencies PROC targ:DWORD, freq :DWORD
    cld
    mov esi, targ
    mov edi, freq
    mov ecx,0

count:
    mov al, [esi + ecx]
    test al, al
    jz done

    movzx eax, al       ;ascii val
    inc dword ptr [edi + eax*4]

    inc ecx
    jmp count
```

```
done:
    ret
Get_frequencies ENDP
```

```
main PROC
    INVOKE Get_frequencies, ADDR target, ADDR freqTable
    mov edx,offset target
    call writestring
    call crlf
    mov edi ,offset target
    mov ecx, 0
    mov esi, offset alphabet
    print:
        mov eax, [freqTable + ecx*4]
        cmp eax, 0
        je skip
        mov al, [esi]
        call WriteChar
        inc esi
        mov eax,':'
        call Writechar
        mov eax, [freqTable + ecx*4]
        call WriteDec
        mov al,' '
        call writechar
```

```
    skip:
        inc ecx
        inc edi
        cmp ecx, 256
        jl print

    exit
main ENDP
END main
```

```
AAEBDCFBBC
A:2 B:3 C:2 D:1 E:1 F:1
```

Q) PALINDROME

1. Take an input string (null-terminated).
2. Use two pointers:

- One starts at the beginning of the string.
- The other starts at the end of the string (traverse to find it).

3. Compare the characters at these pointers:

- If they don't match, the string is not a palindrome.
- If pointers meet or cross and all characters match, it's a palindrome.

4. Display the result.

```
INCLUDE Irvine32.inc

.data
    inputString BYTE "radar", 0        ; Input string (null-terminated)
    palindromeMsg BYTE "Palindrome", 0
    notPalindromeMsg BYTE "Not a Palindrome", 0

.code
main PROC
    ; Initialize Pointers
    lea esi, inputString              ; ESI points to the start of the string
    lea edi, inputString              ; EDI will be used to find the end

    ; Find the end of the string
find_end:
    mov al, [edi]                     ; Load the current character
    cmp al, 0                         ; Check for null terminator
    je check_palindrome               ; If null terminator, start checking
    inc edi                           ; Move to the next character
    jmp find_end
```

```
check_palindrome:
    dec edi                          ; Move EDI to the last valid character

compare_loop:
    cmp esi, edi                     ; Check if pointers have crossed
    jge is_palindrome                ; If ESI >= EDI, it's a palindrome

    mov al, [esi]                    ; Load character from start
    mov bl, [edi]                    ; Load character from end
    cmp al, bl                       ; Compare characters
    jne not_palindrome               ; If not equal, it's not a palindrome

    inc esi                          ; Move start pointer forward
    dec edi                          ; Move end pointer backward
    jmp compare_loop                 ; Repeat comparison
```

```
is_palindrome:
    ; Display "Palindrome"
    mov edx, OFFSET palindromeMsg
    call WriteString
    jmp exit_program

not_palindrome:
    ; Display "Not a Palindrome"
    mov edx, OFFSET notPalindromeMsg
    call WriteString

exit_program:
    call WaitMsg
    ret
main ENDP

END main
```

Task#2

Give an assembly language program to move -128 in BX and expand EBX. Using shift and rotate instruction.

```
INCLUDE Irvine32.inc
.code
main PROC
mov bx, -128
movsx ebx, bx
shl ebx, 16
sar ebx, 16
mov eax, ebx
call DumpRegs
exit
main ENDP
END main
```

```
EAX=FFFFFF80  EBX=FFFFFF80  ECX=005D10AA  EDX=005D10AA
ESI=005D10AA  EDI=005D10AA  EBP=0095F88C  ESP=0095F880
EIP=005D3674  EFL=00000282  CF=0  SF=1  ZF=0  OF=0  AF=0  PF=0


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (proc
```

Task#5

Implement the following C++ expression in assembly language, using 32-bit **signed** operands:

val1 = (val2 / val3) * (val1 / val2);

```
TITLE Expression
INCLUDE Irvine32.inc
.data
   val1 SDWORD 2
   val2 SDWORD 2
   val3 SDWORD 1
   quotient2 SDWORD ?
   quotient1 SDWORD ?

.code
main PROC
  mov eax, [val1]
  cdq
  mov ebx, [val2]
  idiv ebx
  mov [quotient2], eax

   mov eax, [val2]
   cdq
  mov ebx, [val3]
  idiv ebx
  mov [quotient1], eax

  mov eax, [quotient1]
  mov ebx, [quotient2]
  imul ebx
  mov [val1], eax
  call WriteDec

call DumpRegs
exit
main ENDP
END main
```

```
2
  EAX=00000002  EBX=00000001  ECX=00B410AA  EDX=00000000
  ESI=00B410AA  EDI=00B410AA  EBP=00EFFCDC  ESP=00EFFCD0
  EIP=00B436A2  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  P


C:\Users\k230842\source\repos\COALlab10\Debug\COALlab10.exe (
```

Task#8

Write a program that performs simple encryption by rotating each plaintext byte a varying number of positions in different directions. For example, in the following array that represents the encryption key, a negative value indicates a rotation to the left and a positive value indicates a rotation to the right. The integer in each position indicates the magnitude of the rotation:

key BYTE 2, 4, 1, 0, 3, 5, 2, 4, 4, 6

Your program should loop through a plaintext message and align the key to the first 10 bytes of the message. Rotate each plaintext byte by the amount indicated by its matching key array value. Then align the key to the next 10 bytes of the message and repeat the process.

```
INCLUDE Irvine32.inc
.data
key BYTE 2, 4, 1, 0, 3, 5, 2, 4, 4, 6
plaintext BYTE "Coal Finale", 0
encrypted BYTE SIZEOF plaintext dup(0)
.code
main PROC
mov esi, OFFSET plaintext
mov edi, OFFSET encrypted
mov ecx, SIZEOF plaintext
mov ebx, 0
encryptionLoop:
mov al, byte ptr [esi]
cmp al, 0
je done
mov dl, byte ptr [key + ebx]
call rotateByte
mov [edi], al
inc esi
inc edi
inc ebx
cmp ebx, 10
jl encryptionLoop
```

```
mov ebx, 0
jmp encryptionLoop

done:
mov edx, OFFSET encrypted
call writestring
exit
main ENDP
```

```
rotateByte proc
mov cl, dl
cmp dl, 0
jge rotateRight
neg dl

mov cl, dl
shl al, cl
rol al, cl
ret
rotateRight:
mov cl, dl
shr al, cl
mov cl, dl
rol al, cl
ret
rotateByte ENDP

END main
```

```
@``l @h``@d
```

Q) FIND OCCURRENCES OF SUBSTRING IN A STRING:

Algorithm FindSubstringFrequency

Input: main string, substring

Output: Frequency of substring occurrences

Initialize count to 0
Set pointers for main string and substring
Set loop index to 0
Load the first character of the substring into a register (e.g., dh)

While (current index < length of main string):
  If (character at current index == first character of substring):
    Save current index and loop counter
    Compare each character of substring with main string starting from current index
    If all characters match:
      Increment count
    Restore saved index and loop counter if mismatch occurs
  Increment index to move to the next character

Return count
End Algorithm

```
INCLUDE Irvine32.inc

.data
string BYTE "ronarld donar donarld bollarld",0
substring BYTE "rld",0
stringlen DWORD lengthof string-1
substrlen DWORD lengthof substring-1
count DWORD 0
.code
find_substring_freq PROC uses esi edi ecx eax,_string:ptr byte, _substr: ptr byte
mov ecx, stringlen
```

```
mov ebx, _string
mov edi,_substr
mov esi,0
mov dh,[edi]
L1:
    mov dl,dh
    cmp dl,[ebx+esi]
    mov eax,esi
    je check

    back:
    inc esi
    mov edi,_substr

    loop L1
    ret
```

```
check:
    push esi
    push ecx
    mov ecx, substrlen
    L2:

    mov dl,[ebx+esi]
    cmp dl,[edi]
    jne skip
    inc esi
    inc edi
    loop L2
    ; if this loop is over then it means we found the string
    inc count
```

```
skip:
    pop ecx
    pop esi
    jmp back

find_substring_freq ENDP

main PROC
invoke find_substring_freq, ADDR string, ADDR substring
mov eax, count
call writeDec

exit
main ENDP
END main
```

```
3
```

```
TITLE Frequency Table
INCLUDE Irvine32.inc

.data
target BYTE "A BROWN BEAR", 0          ; Input string
freqTable DWORD 256 DUP(0)            ; Frequency table initialized to 0
alphabet BYTE "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 0 ; Alphabet string for display

.code
Get_frequencies PROC targ: DWORD, freq: DWORD
    cld                                ; Clear direction flag
    mov esi, targ                      ; ESI points to the input string
    mov edi, freq                      ; EDI points to the frequency table
    mov ecx, 0                         ; ECX is the loop counter

count:
    mov al, [esi + ecx]                ; Get the current character from the string
    test al, al                        ; Check if it is the null terminator
    jz done                            ; If yes, exit the loop

    movzx eax, al                      ; Zero-extend ASCII value into EAX
    shl eax, 2                         ; Multiply by 4 to calculate DWORD offset
    inc dword ptr [edi + eax]          ; Increment the corresponding frequency count

    inc ecx                            ; Move to the next character in the string
    jmp count                          ; Repeat the loop
```

```
done:
    ret
Get_frequencies ENDP

main PROC
    INVOKE Get_frequencies, ADDR target, ADDR freqTable ; Calculate frequencies

    ; Print the original string
    mov edx, OFFSET target
    call WriteString
    call Crlf

    ; Display the frequency table
    mov ecx, 0                          ; Initialize loop counter for A-Z
    mov esi, OFFSET alphabet            ; Point to the alphabet string
```

```
print:
    cmp ecx, 26                     ; Check if we have processed all letters
    jge done_display                ; If yes, exit the loop

    mov eax, ecx                    ; Copy ECX (index) to EAX
    add eax, 65                     ; Add 65 to get ASCII value of the letter
    shl eax, 2                      ; Multiply by 4 to get the frequency table offset
    mov ebx, [freqTable + eax]      ; Load the frequency from the table
    cmp ebx, 0                      ; Check if the frequency is zero
    je skip                         ; If zero, skip this letter
```

```
    mov al, [esi + ecx]             ; Load the current letter
    call WriteChar                  ; Print the letter

    mov al, ':'                     ; Print a colon separator
    call WriteChar

    mov eax, ebx                    ; Load the frequency value into EAX
    call WriteDec                   ; Print the frequency

    mov al, ' '                     ; Print a space
    call WriteChar

skip:
    inc ecx                         ; Move to the next letter
    jmp print                       ; Repeat the loop
done_display:
    call Crlf                       ; Print a new line for neatness
    exit
main ENDP
END main
```

```
A BROWN BEAR
A:2 B:2 E:1 N:1 O:1 R:2 W:1
```

```c
#include <stdio.h>

void bubbleSort(int array[], int size) {
    int temp;
    for (int cx1 = size - 1; cx1 > 0; cx1--) {
        for (int cx2 = 0; cx2 < cx1; cx2++) {
            if (array[cx2] > array[cx2 + 1]) {
                // Exchange array[cx2] and array[cx2 + 1]
                temp = array[cx2];
                array[cx2] = array[cx2 + 1];
                array[cx2 + 1] = temp;
            }
        }
    }
}
```

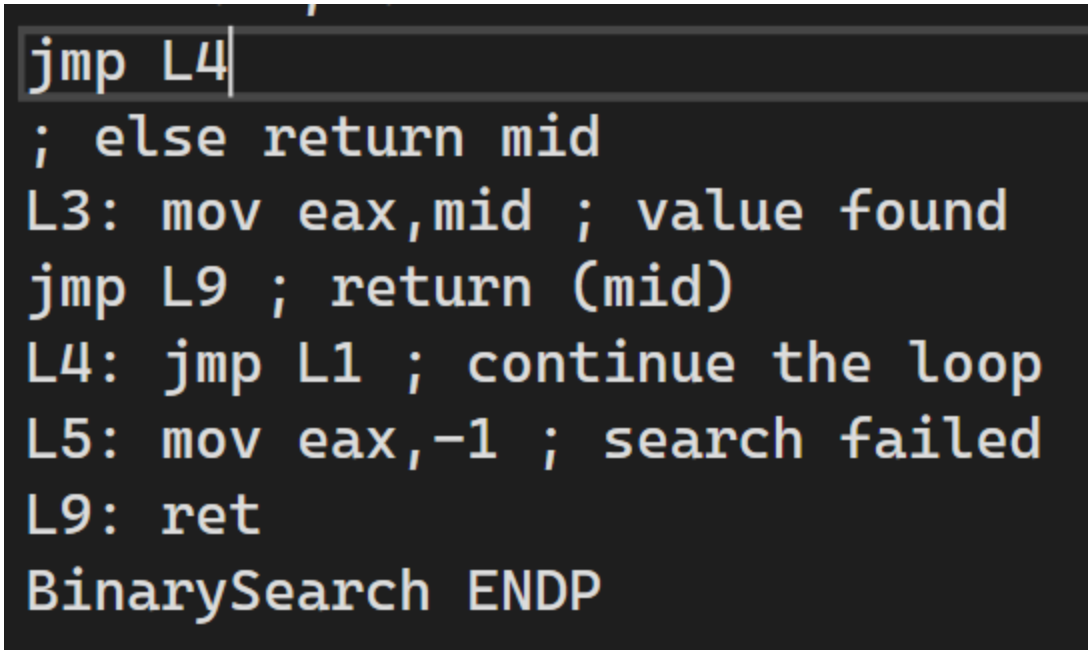
```
;-----------------------------------------------------
BubbleSort PROC USES eax ecx esi,
pArray:PTR DWORD, ; pointer to array
Count:DWORD ; array size
; Sort an array of 32-bit signed integers in ascending
; order, using the bubble sort algorithm.
; Receives: pointer to array, array size
; Returns: nothing
;-----------------------------------------------------
mov ecx,Count
dec ecx ; decrement count by 1
L1:
push ecx ; save outer loop count
mov esi,pArray ; point to first value
L2: mov eax,[esi] ; get array value
cmp [esi+4],eax ; compare a pair of values
jg L3 ; if [ESI] <= [ESI+4], no exchange
xchg eax,[esi+4] ; exchange the pair
mov [esi],eax
L3:
add esi,4 ; move both pointers forward
loop L2 ; inner loop
pop ecx ; retrieve outer loop count
loop L1 ; else repeat outer loop
L4:
ret
BubbleSort ENDP
```

```c
// Binary search function
int binarySearch(int array[], int size, int target) {
    int start = 0, end = size - 1;

    while (start <= end) {
        int mid = (start + end) / 2; // Calculate mid index

        if (array[mid] == target) {
            return mid; // Target found
        } else if (array[mid] < target) {
            start = mid + 1; // Search in the right half
        } else {
            end = mid - 1; // Search in the left half
        }
    }

    return -1; // Target not found
}
```

```
BinarySearch PROC USES ebx edx esi edi,
pArray:PTR DWORD, ; pointer to array
Count:DWORD, ; array size
searchVal:DWORD ; search value
LOCAL first:DWORD, ; first position
last:DWORD, ; last position
mid:DWORD ; midpoint

mov first,0 ; first = 0
mov eax,Count ; last = (count - 1)
dec eax
mov last,eax
mov edi,searchVal ; EDI = searchVal
mov ebx,pArray ; EBX points to the array
L1: ; while first <= last
mov eax,first
cmp eax,last
jg L5 ; exit search
; mid = (last + first) / 2
```

```
; mid = (last + first) / 2
mov eax,last
add eax,first
shr eax,1
mov mid,eax
; EDX = values[mid]
mov esi,mid
shl esi,2 ; scale mid value by 4
mov edx,[ebx+esi] ; EDX = values[mid]
; if ( EDX < searchval(EDI) )
cmp edx,edi
jge L2
; first = mid + 1
mov eax,mid
inc eax
mov first,eax
jmp L4
```

```
; else if( EDX > searchVal(EDI) )
L2: cmp edx,edi ; optional
jle L3
; last = mid - 1
mov eax,mid
dec eax
mov last,eax
jmp L4
; else return mid
L3: mov eax,mid ; value found
jmp L9 ; return (mid)
```



ADDRESSING:

1. MOV ESI, OFFSET array

MOV EAX, [ESI + EBX*TYPE array]
Inc ebx

2. MOV esi, OFFSET array

MOV AL, [esi]
Inc esi

3. array DB 10, 20, 30, 40 ; Define an array with 4 elements (10, 20, 30, 40)

Mov al, [array+1] ; al = 20
Mov al, [array] ; al = 10
Mov al, [array+2] ; al = 30

Q) Reverse words in a string.

Algorithm:

1. **Initialization**:
   a. Define the input string: "Assembly language and Computer".
   b. Define an empty output buffer to store the reversed result.
   c. Initialize a temporary buffer for individual words and set the stack pointer.
2. **Split Words**:
   a. Start iterating through the input string, one character at a time.
   b. If the character is not a space or null terminator:
      - Add the character to the temporary buffer.
      - Keep track of the word length.
        c. If a space is encountered:
      - Terminate the current word in the temporary buffer with a null character (\0).
      - Push the address of the word onto the stack.
      - Reset the buffer pointer for the next word.
3. **Push Last Word**:
   a. If the null terminator of the input string is reached, terminate the final word.
   b. Push the address of the final word onto the stack.
4. **Reverse Word Order Using Stack**:
   a. Start popping words from the stack one by one.
   b. For each popped word, copy its characters into the output buffer.
   c. Append a space after each word, except for the last one.
5. **End of Reversal**:
   a. Terminate the output buffer with a null character to mark the end of the string.
   b. Restore the stack pointer to its original state.
6. **Output the Result**:
   a. Display the reversed string stored in the output buffer.
7. **Expected Result**: “Computer and Assembly language”.

```
INCLUDE Irvine32.inc

.data
inputString BYTE "Assembly language and Computer", 0 ; Input string
outputString BYTE SIZEOF inputString DUP(?)       ; Output string buffer
tempBuffer BYTE SIZEOF inputString DUP(?)         ; Temporary buffer for words
```

```
.code
main PROC
    ; Display the original string
    mov edx, OFFSET inputString
    call WriteString
    call Crlf
    lea edx, inputString      ; Load address of input string into edx
    lea esi, outputString     ; Load address of output string into esi
    call ReverseWords         ; Reverse words and store result in outputString

    ; Display the reversed string
    mov edx, OFFSET outputString
    call WriteString
    call Crlf

    exit
main ENDP
```

```
ReverseWords PROC
    LOCAL wordStart: DWORD   ; Local variable to keep track of current index
    LOCAL wordEnd: DWORD     ; End address of the current word
    LOCAL wordLength: DWORD  ; Length of the current word
    LOCAL tempIndex: DWORD   ; Index for tempBuffer

    ; Initialize necessary registers
    lea ebx, tempBuffer      ; Load address of tempBuffer into ebx
    lea esi, inputString     ; Load address of inputString into esi
    xor edx, edx             ; Clear edx (used to store current character)
    xor ecx, ecx             ; Clear ecx (used as index for tempBuffer)

    ; Initialize word processing
ReverseLoop:
    mov al, [esi]            ; Load the current byte of the input string
    inc esi                  ; Increment input pointer
    cmp al, 0                ; Check for null terminator
    je ReverseComplete       ; If null terminator, finish

    cmp al, ' '              ; Check for space (word delimiter)
    je ProcessWord           ; If space, process the current word

    ; Otherwise, add character to temporary buffer
    mov [ebx + ecx], al
    inc ecx                  ; Move to next position in tempBuffer
    jmp ReverseLoop          ; Repeat the loop
```

```
ProcessWord:
    ; Null-terminate the current word in the tempBuffer
    mov byte ptr [ebx + ecx], 0

    ; Push the address of the word onto the stack
    lea eax, [ebx]            ; Load address of the current word
    push eax

    ; Reset tempBuffer for the next word
    xor ecx, ecx              ; Reset the index for tempBuffer

    ; Continue processing the next word
    jmp ReverseLoop
```

```
ReverseComplete:
    ; Pop each word from the stack and append to the output string
ReverseStack:
    cmp esp, ebp              ; Check if the stack is empty
    je EndReverse             ; If empty, finish
    pop eax                   ; Pop the address of the word from the stack
    lea esi, [eax]            ; Load the address of the word into esi

WriteWord:
    mov al, [esi]             ; Load the current byte from the word
    inc esi                   ; Move to the next character
    cmp al, 0                 ; Check if it is null terminator
    je AddSpace               ; If null, go to add space
    mov [outputString + edx], al ; Store the character in output string
    inc edx                   ; Move to the next position in output string
    jmp WriteWord             ; Repeat for next character

AddSpace:
    cmp esp, ebp              ; Check if it is the last word
    je EndReverse             ; If it's the last word, no space needed
    mov byte ptr [outputString + edx], ' ' ; Add space between words
    inc edx                   ; Move to next position in output string
    jmp ReverseStack          ; Continue popping words from stack
```

```
EndReverse:
    mov byte ptr [outputString + edx], 0   ; Null-terminate the output string
    ret
ReverseWords ENDP

END main
```

Given that EBX points to the following array ewords, write x86 code snippet to process this array and print the starting offse of each word starting with letter e or E.

```
ewords byte “The eagle eyed snake eagerly attacked Easter eggs”, 0
```

```
INCLUDE Irvine32.inc

.data
ewords byte "The eagle eyed snake eagerly attacked Easter eggs", 0

.code
main PROC
    ; Initialize registers
    mov esi, OFFSET ewords
    mov eax, esi
    call WriteHex
    call Crlf
    mov ecx, LENGTHOF ewords

L1:
; Check if the current character is 'E' or 'e'
    mov al, [esi]
    cmp al, 0       ; Check for null terminator
    je Done         ; Exit loop if end of string is reached
```

```
L1:
; Check if the current character is 'E' or 'e'
    mov al, [esi]
    cmp al, 0       ; Check for null terminator
    je Done         ; Exit loop if end of string is reached

    cmp al, 'E'
    je CheckPreviousChar
    cmp al, 'e'
    je CheckPreviousChar
    jmp Continue          ; If not 'E' or 'e', continue to next character
```

Idea is to check if a particular character is ‘e’ or ‘E’.
Then its previous character should be space or one exceptional case is where the first character is ‘e’ or ‘E’ so that's why we compare if its offset is equal to offset of string.

```
CheckPreviousChar:
    ; Check if the previous character is a space or the start of the string
    lea edx, [esi - 1]    ; Address of the previous character
    mov dl, [edx]         ; Load the previous character into DL
    cmp esi, OFFSET ewords ; Check if we are at the first character
    je PrintOffset        ; If yes, the word starts here
    cmp dl, ' '           ; Is the previous character a space?
    jne Continue          ; If not, continue to the next character
```

```
PrintOffset:
    ; sub esi, OFFSET ewords for printing index
     mov eax, esi          ; Move the offset into EAX
     call WriteHex        ; Print the offset as a decimal value
     call Crlf             ; Print a newline

Continue:
     inc esi
     loop L1

Done:
     call DumpRegs
     exit
main ENDP
END main
```

```
005F6000
005F6004
005F600A
005F6015
005F6026
005F602D
```

Qus 06: Consider you have two 16-bit memory spaces. First memory space is used to mark the attendance and other is used to mark participation of a student in the class. Use loop and shift operation take 16-times input from the user for both (attendance and participation). Calculate and display the participation of student out of 16 classes. **[Points: 10]**

```
INCLUDE Irvine32.inc

.DATA
    attendance WORD 0          ; 16-bit memory for attendance
    participation WORD 0       ; 16-bit memory for participation
    totalParticipation DWORD 0 ; To store the count of participation
    promptAttendance BYTE "Enter attendance (0 or 1): ", 0
    promptParticipation BYTE "Enter participation (0 or 1): ", 0
    message BYTE "Total Participation in attended classes: ", 0
    invalidmsg BYTE "Invalid input! Please enter 0 or 1 only.", 0
```

```
.CODE
main PROC
    MOV AX, 0
    MOV attendance, AX
    MOV participation, AX
    mov ecx, 0
    ; Loop 16 times for input
    MOV CX, 4
inputLoop:
    ; Prompt for attendance input
    MOV edx, OFFSET promptAttendance
    CALL WriteString
    CALL ReadInt
    CMP EAX, 0           ; Validate input (0 or 1)
    JL handleInvalid
    CMP EAX, 1
    JG handleInvalid
    SHL attendance, 1      ; Shift attendance left
    OR attendance, AX      ; Add input bit to attendance
```

```
; Prompt for participation input
MOV edx, OFFSET promptParticipation
CALL WriteString
CALL ReadInt
CMP EAX, 0              ; Validate input (0 or 1)
JL handleInvalid
CMP EAX, 1
JG handleInvalid
SHL participation, 1    ; Shift participation left
OR participation, AX    ; Add input bit to participation

LOOP inputLoop
```

```
countParticipation:
    SHR AX, 1             ; Shift right to isolate LSB
    JNC skipIncrement     ; If no carry, skip increment
    INC EBX               ; Increment counter
skipIncrement:
    TEST AX, AX    ; Check if AX is 0
    JNZ countParticipation    ; Continue until all bits are checked

    MOV totalParticipation, EBX
    MOV edx, OFFSET message
    CALL WriteString
    MOV EAX, totalParticipation
    CALL WriteDec   ; Display the participation count
    CALL WaitMsg
    EXIT
```

```
handleInvalid:
    MOV edx, OFFSET invalidmsg
    CALL WriteString
    CALL CrLf
    JMP inputLoop ; Retry input without decrementing C

main ENDP
END main
```

```
Enter attendance (0 or 1): 1
Enter participation (0 or 1): 1
Enter attendance (0 or 1): 1
Enter participation (0 or 1): 1
Enter attendance (0 or 1): 1
Enter participation (0 or 1): 1
Enter attendance (0 or 1): 1
Enter participation (0 or 1): 1
Total Participation in attended classes: 4Press any key to continue...
```

Data transmission systems and file subsystems often use a form of error detection that relies on calculating the parity (even or odd) of blocks of data. Suppose you are given an integer that requires 32 bits to store. You are asked to find whether its binary representation has an odd or even number of 1's. Write a program to read an integer (should accept both positive and negative numbers) from the user and outputs whether it contains an odd or even number 1's. Your program should also print the number of 1's in the binary representation.

```
TITLE Count number of 1's
INCLUDE Irvine32.inc

.DATA
    promptMsg BYTE "Enter an integer: ", 0
    evenMsg BYTE "The binary representation has an even number of 1s.", 0
    oddMsg BYTE "The binary representation has an odd number of 1s.", 0
    onesCountMsg BYTE "Number of 1s: ", 0
```

```
.CODE
main PROC
    ; Prompt the user for input
    MOV edx, OFFSET promptMsg
    CALL WriteString
    CALL ReadInt
    MOV EAX, EAX     ; Store the input in EAX for processing

    ; Count the number of 1s in the binary representation
    XOR ECX, ECX ; ECX will hold the count of 1s

countOnesLoop:
    SHR EAX, 1            ; Shift right, LSB goes to carry flag
    JNC skipIncrement     ; If CF = 0, skip increment
    INC ECX               ; Increment count if CF = 1
skipIncrement:
    TEST EAX, EAX          ; Check if EAX is 0
    JNZ countOnesLoop      ; Continue until all bits are processed
```

```
; Print the count of 1s
MOV edx, OFFSET onesCountMsg
CALL WriteString
MOV EAX, ECX   ; Load count of 1s into EAX
CALL WriteDec
CALL CrLf
```

```
    ; Check if the number of 1s is even or odd
    TEST ECX, 1               ; Check if the count is odd or even
    JZ printEven
    MOV edx, OFFSET oddMsg    ; Odd case
    CALL WriteString
    JMP exitProgram

printEven:
    MOV edx, OFFSET evenMsg   ; Even case
    CALL WriteString

exitProgram:
    CALL CrLf
    CALL WaitMsg
    EXIT
main ENDP
END main
```

```
Enter an integer: 39
Number of 1s: 4
The binary representation has an even number of 1s.
Press any key to continue...
```

```
Enter an integer: 7
Number of 1s: 3
The binary representation has an odd number of 1s.
Press any key to continue...
```

Write an assembly language program with a loop and indexed addressing that calculates the sum of all the gaps between successive array elements. The array elements are DWORDS, sequenced in non-decreasing order. For example, the array {0, 2, 5, 9, 10} has differences of 2, 3, 4, and 1 respectively, whose sum equals 10.

Take the absolute difference between the successive elements .
Differences:

- 2−0=22 - 0 = 22−0=2
- 5−2=35 - 2 = 35−2=3
- 9−5=49 - 5 = 49−5=4
- 10−9=110 - 9 = 110−9=1

Sum: 2+3+4+1=102 + 3 + 4 + 1 = 102+3+4+1=10

```
INCLUDE Irvine32.inc

.DATA
    array DWORD 0, 2, 5, 9, 10   ; Array elements
    arraySize DWORD 5            ; Number of elements in the array
    sumMsg BYTE "Sum of absolute differences: ", 0
    sum DWORD 0                  ; To store the sum of absolute differences
```

```
.CODE
main PROC
    mov ebx, 0
    mov ecx, 0
    mov eax, 0
    MOV ESI, OFFSET array        ; Point to the start of the array
    dec arraysize
calculateDiff:
    CMP EBX, arraysize     ; Check if we reached the second last element
    JGE displaySum               ; If yes, jump to display the result

    MOV EAX, [ESI + EBX * 4]
    ADD EBX, 1                   ; Increment index
    MOV EDX, [ESI + EBX * 4]

    SUB EDX, EAX    ; Calculate the difference (next - current)
```

```
    ; Take the absolute value of the difference
    CMP EDX, 0                  ; Check if the difference is negative
    JGE addToSum                ; If EDX >= 0, jump to add to sum
    NEG EDX                     ; If EDX < 0, negate it to get the absolute value

addToSum:
    ADD ECX, EDX                ; Add the absolute difference to the sum
    JMP calculateDiff           ; Repeat for the next element
```

```
displaySum:
    MOV sum, ECX

    ; Print the sum of absolute differences
    MOV EDX, OFFSET sumMsg
    CALL WriteString
    MOV EAX, sum
    CALL WriteDec
    CALL CrLf

    CALL WaitMsg
    EXIT
main ENDP
END main
```

```
Sum of absolute differences: 10
Press any key to continue...
```

Q) Write an Assembly Language Program having a procedure CAPITALCASE to find all capital letters in a string and store only Capital Case Letters in a DWORD array CCASE. In this problem, you are supposed to use recursion performed through stack frames discussed in the class.

For example, if the INPUT string passed is: FAST National University, Karachi
The Expected Output will be: FAST N U K

```
Include irvine32.inc
.data
STR1 BYTE "FAST National University, Karachi.",0
CCASE BYTE 100 DUP(0)
.code
MAIN PROC
mov esi, OFFSET STR1
mov edi, OFFSET CCASE
CALL CAPITALCASE
mov edx, OFFSET CCASE
call WriteString
exit
main ENDP
```

```
CAPITALCASE PROC
cmp BYTE PTR [ESI], ' '
JE _IF
cmp BYTE PTR [esi], 'A'
JNAE _ENDIF
cmp BYTE PTR [esi], 'Z'
JNBE _ENDIF
_IF:
mov bl, [esi]
mov [edi],bl
inc edi
_ENDIF:
CMP BYTE PTR [esi],0
JE L2
inc esi
CALL CAPITALCASE
L2:
ret
CAPITALCASE ENDP
END main
```

```
FAST N U K
```

Q) Write an assembly language procedure MAXIMUM that is called from the MAIN procedure to find the maximum MAX among 10 elements of an array ARRAY1. The arguments are passed by reference to the procedure MAXIMUM using registers. The result is also returned in a register. Also, write the corresponding data definition directives.

```
Include irvine32.inc
.DATA
arr DWORD 0,10,2,3,4,5,6,7,8,9

.CODE
MAXIMUM PROC USES ECX ESI
mov eax, [esi]
L1:
cmp eax, [esi]
JG L2
mov eax, [esi]
L2:
add esi, 4
LOOP L1
ret
MAXIMUM ENDP
```

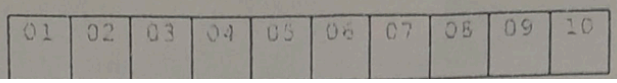
```
main PROC
mov esi, OFFSET arr
mov ecx, LENGTHOF arr
call MAXIMUM
call WriteInt
exit
main ENDP
END main
```

```
+10
```

Q) Multiplying an array.

```
TITLE Multiplying an array
INCLUDE Irvine32.inc
.data
array DWORD 1,2,3,4,5,6,7,8,9,10
multiplier DWORD 10
.code
main PROC
    cld
    mov esi,OFFSET array
    mov edi,esi
    mov ecx,LENGTHOF array

L1: lodsd               ; copy [ESI] into EAX
    mul multiplier      ; multiply by a value
    stosd               ; store EAX at [EDI]
    loop L1
    exit
main ENDP
END main
```

Using string primitive instructions, replace each element of a given array by its mathematical square. Assume any valid type for array1.




```
TITLE Multiplying an array
INCLUDE Irvine32.inc
.data
array DWORD 1,2,3,4,5,6,7,8,9,10
multiplier DWORD 10
.code
main PROC
    cld
    mov esi,OFFSET array
    mov edi,esi
    mov ecx,LENGTHOF array

L1: lodsd               ; copy [ESI] into EAX
    mul eax  ; multiply by a value
    stosd               ; store EAX at [EDI]
    loop L1
    mov esi, OFFSET array
    mov ecx, LENGTHOF array
printarr:
    mov eax, [esi]
    call WriteDec
    add esi, 4
    call Crlf
    loop printarr
    exit
main ENDP
END main
```

```
1
4
9
16
25
36
49
64
81
100
```

```
TITLE FIBIONACCI SERIES
INCLUDE Irvine32.inc
.data
Fib DWORD 1,1, 10 DUP(0)
.code
main PROC
mov esi, OFFSET Fib
mov edi, OFFSET [Fib+4]
mov ecx, 10
L1:
mov eax, [esi]
mov edx, [edi]
add eax, edx
mov [edi+4],eax
add esi, 4
add edi,4
loop L1
mov eax, 0
mov esi, OFFSET Fib
mov ecx, LENGTHOF Fib
printarr:
mov eax, [esi]
add esi, 4
call WriteDec
call Crlf
loop printarr
exit
main ENDP
END main
```

```
INCLUDE Irvine32.inc

.data
    inputString BYTE "Assembly language and Computer", 0
    outputBuffer BYTE SIZEOF inputString DUP(0)
    tempBuffer BYTE 50 DUP(0)
```

```
.code
main PROC
    ; Initialize registers
    lea esi, inputString         ; Load address of inputString into ESI
    lea edi, outputBuffer        ; Load address of outputBuffer into EDI
    lea ebx, tempBuffer          ; Load address of tempBuffer into EBX
    mov ecx, 0                ; Clear ECX (used as index for tempBuffer)

reverseLoop:
    mov al, [esi]                ; Load the current character from inputString
    inc esi                       ; Increment the pointer to the next character
    cmp al, 0                     ; Check for the null terminator (end of string
    je finalizeWord               ; If null terminator, finalize the last word

    cmp al, ' '                   ; Check if the current character is a space
    je processWord                ; If space, process the current word

    ; If it's a regular character, add it to the temporary buffer
    mov [ebx + ecx], al          ; Store the character in tempBuffer
    inc ecx                       ; Increment the index for tempBuffer
    jmp reverseLoop               ; Repeat the loop
```

```
processWord:
   ; Null-terminate the current word in tempBuffer
   mov byte ptr [ebx + ecx], 0   ; Null-terminate the word in tempBuffer

   ; Push the address of the word onto the stack
   lea edx, [ebx]                ; Load the address of the current word into EDX
   push edx                      ; Push the address onto the stack

   ; Reset the tempBuffer for the next word
   mov ecx, 0                  ; Reset the index for tempBuffer

   ; Continue with the next character
   jmp reverseLoop
```

```
finalizeWord:
    ; Null-terminate the last word if necessary
    mov byte ptr [ebx + ecx], 0    ; Null-terminate the last word

    ; Push the address of the last word onto the stack
    lea edx, [ebx]                 ; Load the address of the current word into EDX
    push edx                      ; Push the address onto the stack
```

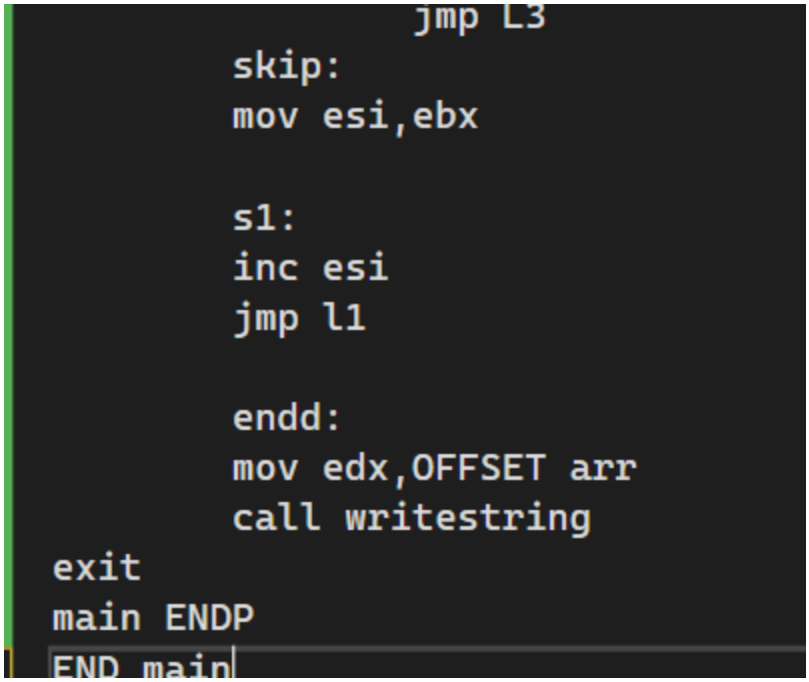
```
reverseWords:
    ; Pop the address of a word from the stack
    pop eax                        ; Pop the address of the word into EAX
    lea ebx, [eax]                 ; Load the word's address into EBX

    ; Copy the word into the output buffer
copyWord:
    mov al, [ebx]                  ; Load the current character from the word
    cmp al, 0                       ; Check if it's the null terminator
    je checkNextWord               ; If null terminator, go to next word
    mov [edi], al                  ; Store the character in outputBuffer
    inc edi                        ; Increment output buffer pointer
    inc ebx                        ; Move to the next character of the word
    jmp copyWord                   ; Repeat for the next character
```

```
checkNextWord:
    ; Add a space after the word if it's not the last word
    ; Since the stack is empty after the last word, we check if the stack is empty
    cmp esp, OFFSET inputString    ; Check if the stack is empty (no more words left)
    je doneReversing               ; If last word, skip adding space
    mov byte ptr [edi], ' '         ; Add a space after the word
    inc edi

    ; Repeat for the next word
    jmp reverseWords
```

```
doneReversing:
    ; Null-terminate the output string
    mov byte ptr [edi], 0

    ; Output the reversed string
    mov edx,OFFSET outputBuffer           ; Load address of outputBuffer
    call WriteString               ; Display the result

    exit
main ENDP
END main
```

```
TITLE REPLACE assembly WITH ASM
INCLUDE Irvine32.inc
.data
    arr BYTE "this is assembly. it assembly is lame",0
    find BYTE "assembly",0
    replace BYTE "ASM",0
.code
main PROC
    mov eax,0
    mov ebx,0
    mov esi,OFFSET arr
    l1:
        mov al,[esi]
        cmp esi,OFFSET arr+LENGTHOF arr-LENGTHOF find    ;DONT CHECK TILL LAST
        je endd
        mov ebx,esi
        mov edi, OFFSET find
        mov ecx, LENGTHOF find-1    ; -1 v imp here
        cld
        repe cmpsb
        jne skip
```

```
copy:
    mov edi,OFFSET replace
    mov ecx,LENGTHOF replace-1
    l2:
        mov al,[edi]
        mov [ebx],al
        inc edi
        inc ebx
        loop l2
    mov esi,ebx    ;SAVE ESI TO NEXT POSITION AFTER 'ASM'
    dec esi

    ;NOW EMPTY SPACES SHIFT (EBX+5)
    mov ecx,lengthof find-lengthof replace   ;ECX=5
    L3:
        mov al,[ebx+ecx]
        mov [ebx],al
        inc ebx
        cmp al,0     ;DO EXIT AFTER COPYING NULL TERMINATOR ig it works otherwise
        je s1
        jmp L3
```


jmp L3
skip:
mov esi,ebx
s1:
inc esi
jmp l1
endd:
mov edx,OFFSET arr
call writestring
exit
main ENDP
END main

Question 3 **[5] Points**

Write an assembly language program to copy the characters of a string to a target string. The characters are stored in such a way that only a single instance of any character in the string is stored. Initialize a source string to: "This is the source string".

Logic:

Initialize string 1 to “This is the source string”.
Declare string 2.
Copy the characters from string 1 to string 2.
Only different characters should be copied. No duplicates allowed.
So before copying, check if the character already exists.

```
INCLUDE Irvine32.inc
.data
string1 BYTE "This is the source string", 0
string2 BYTE 20 DUP(?)
msg1 BYTE "Copied String after removing duplicates:", 0

.code
main PROC
mov esi, OFFSET string1
mov edi, OFFSET string2
mov ecx, 0
loop1:
mov al, [esi]
cmp al, 0
je done
```

```
mov ebx, OFFSET string2
mov edx, ecx

check_duplicates:
cmp al, [ebx]
je found_duplicate
inc ebx
dec edx
jns check_duplicates
```

```
mov [edi + ecx], al
inc ecx

found_duplicate:
inc esi
jmp loop1

done:
mov BYTE PTR [edi + ecx], 0
mov edx, OFFSET msg1
call WriteString
mov edx, OFFSET string2
call WriteString
call Crlf

exit
main ENDP
END main
```

```
Copied String after removing duplicates:This teourcng
```

Question 1. **[5] Points**

Write a recursive procedure in x86 assembly language that divides a number by another number and stops when dividend is less than or equal to 5h. Consider dividend = D4A4h and divisor = Ah. The Intel IA 32 version of this program is required.

Step-by-Step Execution:

1. **Initial Call**:
    - EAX = 0D4A4h (54436)
    - Division : 54436 ÷ 10
    - **Quotient**: 5443, **Remainder**: 6
    - Push return address.
    - Recursive call with EAX = 5443.
2. **First Recursive Call**:
    - EAX = 5443
    - Division : 5443 ÷ 10
    - **Quotient**: 544, **Remainder**: 3
    - Push return address.
    - Recursive call with EAX = 544.
3. **Second Recursive Call**:
    - EAX = 544
    - Division : 544 ÷ 10
    - **Quotient**: 54, **Remainder**: 4
    - Push return address.
    - Recursive call with EAX = 54.
4. **Third Recursive Call**:
    - EAX = 54
    - Division : 54 ÷ 10
    - **Quotient**: 5, **Remainder**: 4
    - Push return address.
    - Recursive call with EAX = 5.
5. **Base Case Reached**:
    - EAX = 5
    - The condition cmp eax, 5h is true ($5 \leq 5$).
    - Exit recursion and return.

Stack Level	EAX (Dividend)	Quotient	Return Address
Base Case	5	-	(Final Return)
3rd Call	54	5	Address of Step 4
2nd Call	544	54	Address of Step 3
1st Call	5443	544	Address of Step 2
Initial Call	54436	5443	Address of Step 1

```
TITLE Division of Integers
INCLUDE Irvine32.inc
.data
dividend DWORD 0D4A4h
divisor DWORD 0Ah
quotient DWORD ?
.code
main PROC
mov eax, 0
mov ecx, 0
mov eax, dividend
mov ecx, divisor
call DivIntegers
mov eax, quotient
call WriteDec
call Crlf
exit
main ENDP
DivIntegers PROC
cmp eax, 5h
jle L2
xor edx, edx
div ecx
mov quotient, eax
call DivIntegers
L2:
ret
DivIntegers ENDP
END main
```

```
5

C:\Users\
To automa
```

Q) Shifting the Elements in an Array

Using a loop and indexed addressing, write code that rotates the members of a 32-bit integer array forward one position. The value at the end of the array must wrap around to the first position. For example, the array [10,20,30,40] would be transformed into [40,10,20,30].

Here i have used an additional register ebx to resolve the constant value error.

ebx is used instead of ecx-1.

```
TITLE Rotate 32-bit Array forward
INCLUDE Irvine32.inc
.data
arr DWORD 10, 20, 30, 40
arrsize = LENGTHOF arr
arrt = TYPE arr
.code
main PROC
mov esi, OFFSET arr
mov ecx, arrsize
mov ebx, ecx
sub ebx, 1
mov eax, [esi+ebx * 4]
dec ecx
mov ebx, 0
```

```
      shiftloop:
      mov ebx, ecx
      sub ebx, 1
      mov edx, [esi + ebx * 4]
      mov [esi + ecx * 4], edx
      dec ecx
      jnz shiftloop

mov [esi], eax
```

```
        mov ecx, arrsize
        mov esi, OFFSET arr
        printloop:
        mov eax, [esi]
        call WriteInt
        call Crlf
        add esi, 4
        loop printloop

 exit
 main ENDP
 END main
```

```
+40
+10
+20
+30
```

Algorithm for Checking Valid Parentheses in Assembly:

1. Initialization:
    - Calculate the length of the expression string.
    - Initialize ECX with the length of expression to serve as the loop counter.
    - Set ESI to 0 to use as an index for accessing each character in expression.
    - Set EDI to 0 to use as a counter to track the balance of parentheses.
2. Loop Through Each Character in the Expression:
    - Load the character at the ESI index from expression into AL.
    - Check if the character is an opening parenthesis '(':
        - If true, jump to the save section.
        - In save, push EAX (which contains '(') onto the stack, and increment EDI (balance counter) by 1. Then, jump to next to continue to the next character.
    - If the character is a closing parenthesis ')':
        - If true, jump to _test.
        - In _test, check if EDI (balance counter) is 0 (indicating no matching opening parenthesis).
            - If EDI is 0, jump to the error section.
            - Otherwise, pop the stack to retrieve the last pushed '(', decrement EDI by 1 (indicating one matching pair is closed). Then, jump to next to continue to the next character.
    - If the character is neither '(' nor ')', jump to next to skip it.
3. Continue to the Next Character:
    - Increment ESI to move to the next character in expression.
    - Decrement ECX (loop counter) and repeat the loop until ECX reaches 0 (all characters processed).
4. Final Balance Check:
    - After the loop, check if EDI is 0.
        - If EDI is not 0, jump to the error section because there are unmatched opening parentheses.
        - If EDI is 0, all parentheses are balanced, and the program can proceed to display a valid message.
5. Display Result:
    - If all parentheses are balanced, load the validMessage into EDX and call WriteString to print "Parentheses are balanced".
    - If there is an imbalance (unmatched parentheses), load the errorMessage into EDX and call WriteString to print "Parentheses are not balanced".
6. End the Program:
    - Print a newline using Crlf and exit the program.

```
TITLE Valid Parentheses Check (ValidParentheses.asm)
INCLUDE Irvine32.inc

.data
    expression BYTE "((2+3)-(4+4))", 0  ; Expression to check
    _length = ($ - expression) - 1       ; Length of the expression string
    validMessage BYTE "Parentheses are balanced", 0
    errorMessage BYTE "Parentheses are not balanced", 0

.code
main PROC
    ; Initialize
    mov ecx, _length       ; Set loop counter to length of expression
    mov esi, 0             ; Index for accessing each character in expression
    mov edi, 0             ; Initialize stack balance counter
```

```
L1:
    mov al, expression[esi]      ; Load character from expression

    cmp al, '('        ; Check if character is '('
    je save            ; If '(', jump to save

    cmp al, ')'       ; Check if character is ')'
    je _test          ; If ')', jump to test

    jmp next        ; Skip other characters

save:
    push eax        ; Push '(' onto the stack
    inc edi         ; Increase balance counter
    jmp next

_test:
    cmp edi, 0      ; Check if there's an unmatched ')'
    je error        ; If balance counter is zero, report error

    pop eax         ; Pop the top of the stack
    dec edi         ; Decrease balance counter
    cmp al, '('     ; Check if matching '(' was popped
    je next         ; If match, continue
    jmp error       ; Otherwise, error

next:
    inc esi     ; Move to the next character
    loop L1     ; Repeat until end of expression
```

```
    ; Final check to see if all parentheses were matched
    cmp edi, 0   ; If counter is not zero, report error
    jne error

    ; Display valid message
    mov edx, OFFSET validMessage
    call WriteString
    jmp endProg

error:
    ; Display error message
    mov edx, OFFSET errorMessage
    call WriteString

endProg:
    call Crlf
    exit
main ENDP
END main
```

```
Parentheses are balanced
```