Q1)

```
Include Irvine32.inc
Include Macros.inc

.data
arr sdword 30, -40, 20, 65, 80, 45
j sdword ?
k sdword ?

.code
main proc

mov ecx, LENGTHOF arr
mov ebx, 0

mov j, 20
mov k, 50
push OFFSET arr
push sizeof arr
push j
push k
call arraySum
call printResults

mov j, 35
mov k, 90
Push offset arr
push sizeof arr
push j
push k
call arraySum
Call printResults
exit
main endp

PrintResult proc
mwrite "The sum of all elements is: "
call writeint
call crlf
ret
PrintResult endp.
```

```
arraySum  PROC  uses  EBX  ECX  EDX  ESI
local first : sdword , last : sdword , sizeArray : sdword

mov esi , [ebp + 20]

mov eax , [ebp + 16]
mov sizeArray , eax

mov eax , [ebp + 12]
mov first , eax

mov eax , [ebp + 8]
mov last , eax

mov eax , 0
mov edx , 0
mov ecx , sizeArray

sumInRange :

    mov ebx , [esi + edx * 4]
    cmp ebx , first
    jge CheckIfWithinRange
    jmp ContinueLoop.

    CheckIfWithinRange :

        cmp ebx , last
        jle AddIt
        jmp ContinueLoop

    AddIt :

        add eax , ebx

    ContinueLoop :

        inc ecx
        sub ecx , 4
        inc edx
        loop sumInRange

  ret 16

arraySum  endp

end main .
```

Q2)

```
Include Irvine32.inc
Include Macros.inc

.data
    arr dword  60, 40, 17, 45, 7

.code
    main  proc
          mov ecx, lengthof arr
          mov ebx, 0
          ~~getInput~~
          call selectionSort
          mov ecx, lengthof arr
          mov ebx, 0
          printSortedArr:
              mov eax, [arr + ebx * 4]
              call Writedec
              mWrite " "
              inc ebx
          loop-SortedArr
          exit
    main  endp


    swap  proc
          push ebp
          mov ebp, esp
          mov edx, [esp + 8]
          push edx
          mov eax, [arr + edx * 4]
          mov edx, [ebp + 12]
          xchg eax, [arr + edx * 4]
          pop edx
          mov [arr + edx * 4], eax
          pop ebp
          ret 8
    swap  endp.
```

```
SelectionSort proc
local largest : dword , i : dword , j : dword
mov ecx , lengthof arr
mov largest , 0
dec ecx
mov i , ecx
mov j , ecx
outerloop:
        mov ebx , i
        mov largest , ebx
        push ecx
        mov edx , i
        mov j , edx
        innerLoop:
            dec j
            mov edx , j
            mov eax , [arr + edx * 4]
            mov edx , largest
            mov ebx , [arr + edx * 4]
            cmp eax , ebx
            jg MarkNewMax
            jmp continueLoop
            MarkNewMax:
                mov edx , j
                mov largest , edx
            continueLoop:
        loop innerLoop
        push i
        push largest
        call swap
        pop ecx
        dec i
loop outerloop
ret
SelectionSort endp

END MAIN
```

Q3.

```
Include Irvine32.inc
Include Macros.inc

.data
arr byte 10 DUP(?)

.code
main proc

mov ecx, lengthof arr
mov ebx, 0
getInput:
        mWrite "Enter value #"
        Call Readint
        mov [arr + ebx], al
        inc ebx
loop getInput

mov esi, offset arr
mov ebx, lengthof arr
Call BubbleSort
mov ebx, 0
mov ecx, lengthof arr
printArray:
        mov al, [arr + ebx]
        call Writedec
        inc ebx
loop printArray
exit
main endp.

BubbleSort proc
    mov edi, esi
    mov ecx, ebx
    dec ecx
    mov ebx, 0
    mov eax, 0
```

```
outerloop:
        push ecx
        mov esi, edi
        innerloop
            mov al, [esi]
            mov bl, [esi + 1]
            cmp al, bl
            jg swapElements
            continueLoop:
                inc esi
        loop innerloop
        pop ecx
    loop outerloop.
    jmp endProgram.

    swapElements:
        mov al, [esi]
        mov bl, [esi+1]
        xchg al, bl
        mov [esi], al
        mov [esi + 1], bl
        jmp continueLoop

    endProgram:
    ret 8
BubbleSort ENDP
ENDMAIN.
```

Q4.

```
Include Irvine32.inc
Include Macros.inc

.data
N dword ?

.code
main proc
 mWrite "Enter the number: "
 call Readint
 mov N , eax.
 Call Factorial
 mWrite "Factorial = "
 call Writedec
 exit
main endp.

factorial proc
    mov eax, 1
    cmp N , 0
    jle endProgram
    mov ecx, N
    Calculate:
        mov edx, 0
        mul ecx.
    loop Calculate.
    end_Program:
    ret
factorial endp.

end main.
```

05.

```
Include Irvine32.inc
    Include Macros.inc

    .data

        character byte ?
        binaryCode dword 0000 0000b
        oneCount dword 0

    .code

        main proc

            mov eax, 0
            mov edx, 0
            mWrite "TYPE A CHARACTER"
            call ReadChar
            call WriteChar ; to show on console
            mov character, al
            mov ah, 0
            movzx ebx, al
            mov binaryCode, ebx
            mov eax, binaryCode
            call countOnes
            call crlf
            mWrite "THE ASCII CODE OF "
            mov al, character
            call WriteChar
            mWrite "in binary is:"
            mov eax, binaryCode
            mov ebx, 1
            call WriteBinB
            call crlf
            mWrite "The NUMBER OF 1 BIT IS:"
            mov eax, OneCount
            call WriteDec
            exit
    main endp.
```

```
countOnes    proc

        mov   eax , binaryCode
        mov   ebx , 2

        loop1:
            mov   edx , 0
            div   ebx
            cmp   edx , 1
            jz    incrementCount
            cmp   eax , 0
            ~~cmp   eax , 0~~
            jz    endloop
                                → jmp continueLoop.
            incrementCount :
                    inc   oneCount

            continueLoop :
        loop   loop1

        endloop:
            ret

countOnes   endp.

END MAIN
```

Q6.

```
Include Irvine32.inc
Include Macros.inc

countMatches PROTO,
        S1 : ptr sdword
        S2 : ptr sdword
        len-Arr : dword.

.data
        arr1   sdword   5, 6, 7, 8, 9
        arr2   sdword   5, 6, 7, 9, 8

.code
        main proc.
            INVOKE  countMatches, ADDR arr1, ADDR arr2, lengthof arr1
            mWrite "Number of common elements is: "
            call Writedec
            exit
        main endp.

countMatches PROC uses EBX ECX EDX ESI EDI,
        S1 : ptr sdword, S2 : ptr sdword, lengthArr : dword.
            mov esi, S1
            mov edi, S2
            mov ebx, 0
            mov eax, 0
            mov ecx, lengthArr
            dec ecx
            compare-Elements :
                mov edx, [esi + ebx * 4]
                cmp edx, [edi + ebx * 4]
                jz incrementCount
                jmp ContinueLoop.
                incrementCount :
                        inc eax
                continueLoop :
                        inc ebx
            loop Compare_Elements
            ret
    CountMatches endp.

ENDMAIN.
```

Q7

```
include Irvine32.inc
include Macros.inc

.data
    num1  qword  ?
    num2  qword  ?
    diff  dword  3  dup (?)

.code
    main  proc

        mov esi, offset num1
        mov edi, offset num2.
        mov ebx, offset diff
        mov ecx, 2
        call Extended_sub.
        mov ecx, lengthof diff
        mov ebx, ecx
        dec ebx.
        mWrite "Result: "
        printarr:

            mov eax, [diff + ebx * 4]
            call ~~Writedec~~ WriteHex
            dec ebx.
        loop printarr

        exit
    main endp.

    Extended_sub proc.
        clc
        subtract.
            mov eax, [esi]
            sbb eax, [edi]
            pushfd
            mov [ebx], eax
            add esi, 4
            add edi, 4.
            add ebx, 4
            popfd
        loop subtract

        sbb word ptr [ebx], 0
        ret
    Extended_sub endp
END MAIN.
```

Q8.

```
Include Irvine32.inc
Include macros.inc
.data
   num1   qword  ?
   num2   qword  ?
   sum    dword  3 dup (?)
.code
   main proc
   mov esi, offset num1
   mov edi, offset num2
   mov ebx, offset sum
   mov ecx, 2
   Call Extended_Add
   mov ecx, lengthof sum
   mov ebx, ecx
   dec ebx
   mWrite " Result = "
   printArr:
            mov eax, [sum + ebx * 4]
            Call WriteHex
            dec ebx
   loop printArr
   exit
main endp

Extended_Add  proc
     clc
     add:
        mov eax, [esi]
        adc eax,[edi]
        pushfd
        mov [ebx], eax
        add esi, 4
        add edi, 4
        add ebx, 4
        popfd
     loop add

     adc word ptr [ebx], 0
     ret
Extended_Add endp
End main
```

Qa.

```
Include Irvine32.inc
Include macros.inc

.data
    A dword ?
    B dword ?
    gcd dword ?

.code
    main proc
      mov eax, 5    ; (5, 20)
      mov A, eax
      mov eax, 20
      mov B, eax.
      call getGCD.
      call printResult.

      mov eax, 24.   ; (24, 18)
      mov A, eax
      mov eax, 18
      mov B, eax.
      call getGCD.
      call printResult.

      mov eax, 432   ;  (432, 226)
      mov A, eax
      mov eax, 226
      mov B, eax
      call getGCD.
      call printResult
      exit
    main endp.

    PrintResult proc
      mWrite "The gcd is"
      mov eax, gcd
      call Writedec
      call crlf
      ret
    PrintResult endp
```

```
getGCD  proc
    cmp A , 0
    jz firsthand
    cmp B , 0
    jz secondhand
    mov eax , A
    cmp eax , B
    jz thirdhand.
    jg firstrecurse
    jL secondrecurse

    firsthand :
            mov eax , B
            mov gcd , eax
            ret
            jmp endprogram.
    secondhand :
            mov  eax , A
            mov  gcd , eax
            ret
            jmp endprogram
    third hand :
            mov  eax , A
            mov  gcd , eax
            ret
            jmp endProgram.
    firstRecurse :
            mov  eax , A
            sub  eax , B
            mov  A , eax
            Call getGCD.
            jmp endProgram
Secondrecurse :
            mov eax , B
            sub eax , A
            mov B , eax
            call getGCD.
endProgram :
            ret
           getGCD  endp
           END MAIN.
```

Q10.

```
Include Irvine32.inc
Include Macros.inc

CountNearMatches proto,
        S1 : ptr sdword,
        S2 : ptr sdword,
        lengthArr : dword,
        diff_max : sdword

.data
    arr1 sdword 10, 20, 30, 40, 50
    arr2 sdword 5, 16, 28, 31, 49

.code
    main proc

        invoke CountNearMatches, ADDR arr1, ADDR arr2, lengthof arr1, 4

        mWrite " Result = "
        Call Writedec
        exit
    main endp.

    CountNearMatches proc uses EBX ECX EDX ESI EDI,
            S1 : ptr sdword, S2 : ptr sdword, lengthArr : dword, diff_max : sdword

            mov eax, 0
            mov esi, S1
            mov edi, S2
            mov eax, lengthArr
            mov ebx, 0

            CompareEl :
                    mov edx, [esi + ebx * 4]
                    sub eax, [edi + ebx * 4]
                    cmp edx, diff_max
                    jle incrementCount
                    jmp Continueloop.
                    incrementCount :
                            inc eax.
                    continueLoop :
                            inc ebx
            loop CompareEl
            ret
    CountNearMatches endp
    ENDMAIN
```