# LAB 05 Syed Hanzala Ali 24K0025

## Q1 (a)

```
INCLUDE Irvine32.inc
.data
.code
main PROC
   mov AL, 7Fh        ; AL = 7Fh (127 in decimal)
   add AL, 1          ; AL = 7Fh + 1 = 80h (128 in decimal)
; Flags after execution:
; Zero flag (ZF) = 0
; Sign flag (SF) = 1
; Carry flag (CF) = 1
; Overflow flag (OF) = 1
    exit
main ENDP
END main
```

## Q1 (b)

```
INCLUDE Irvine32.inc
.data
.code
main PROC
   mov AL, 7Fh        ; AL = 7Fh (127 in decimal)
   sub AL, 80h        ; AL = 7Fh - 80h = FFh (-1 in decimal)
; Flags after execution:
; Zero flag (ZF) = 0
; Sign flag (SF) = 1
; Carry flag (CF) = 0
; Overflow flag (OF) = 0
    exit
main ENDP
END main
```

## Q2

```
INCLUDE Irvine32.inc
.data
   myByte BYTE 12h
   myWord WORD 1234h
   myDword DWORD 12345678h
.code
main PROC
   mov ESI, OFFSET myByte
   mov AX, WORD PTR [myDword + 2]
   mov BX, TYPE myByte
   exit
main ENDP
END main
```

## Q3

```
INCLUDE Irvine32.inc
.data
   arr1 BYTE 10,20,30,40
   arr2 WORD 100h,200h,300h
   arr3 DWORD 5 DUP(0)
.code
main PROC
   mov AX, LENGTHOF arr1
   mov BX, SIZEOF arr2
   mov CX, LENGTHOF arr3
   exit
main ENDP
END main
```

## Q4 (a)

```
INCLUDE Irvine32.inc
.data
   arrayB BYTE 11h, 22h, 33h
   arrayW WORD 4444h, 5555h, 6666h
.code
main PROC
   mov ESI, OFFSET arrayB
   mov AL, [ESI]
   inc ESI
   mov AL, [ESI]
   inc ESI
   mov AL, [ESI]
   exit
main ENDP
END main
```

## Q4(b)

```
INCLUDE Irvine32.inc
.data
   arrayB BYTE 11h, 22h, 33h
   arrayW WORD 4444h, 5555h, 6666h
.code
main PROC
   mov ESI, OFFSET arrayW
   mov AX, [ESI]
   add ESI, 2
   mov AX, [ESI]
   add ESI, 2
   mov AX, [ESI]
   exit
main ENDP
END main
```

## Q4(c)

The increment of ESI is different in both cases because of the size of the data type being accessed in each array. For arrayB, which contains byte elements, ESI is incremented by 1 after accessing each element because each element is 1 byte in size. For arrayW, which contains word elements, ESI is incremented by 2 after accessing each element because each element is 2 bytes in size. Therefore, the increment of ESI depends on the size of the data type being accessed in the arrays.

## Q5

```
INCLUDE Irvine32.inc
.data
   arrayD DWORD 1000h,2000h,3000h,4000h
.code
main PROC
  mov EAX, [arrayD + 1 * TYPE DWORD]
  mov EBX, [arrayD + 3 * TYPE DWORD]
   exit
main ENDP
END main
```

## Q5 (c)

In this example, the TYPE operator is used to determine the size of each element in the array. It multiplies the index by the size of the data type (in this case, DWORD, which is 4 bytes) to calculate the correct byte offset for accessing elements within the array. The TYPE operator helps in correctly calculating the memory address for each element when using indexed addressing with scale factors, ensuring that the program accesses the right element based on its size. This is crucial for correctly handling arrays of different data types like DWORDs, which occupy 4 bytes each.