University of Engineering and Technology

**COAL Lab Assignment 4**

**Submitted By:**

Hasan Ahmed

(2019-CS-145)

**Submitted To:**

Sir Hassan

**Question 1:**

**Code:**

INCLUDE Irvine32.inc

.data

oneByte BYTE 78h

oneWord WORD 1234h

oneDword DWORD 12345678h

.code

main PROC

mov eax,99999999h ; EAX = 99999999h

call DumpRegs

mov al,oneByte ; EAX = 00000078h

call DumpRegs

mov ax,oneWord ; EAX = 00001234h

call DumpRegs

mov eax,oneDword ; EAX = 12345678h

call DumpRegs

mov ax,0 ; EAX = 12340000h

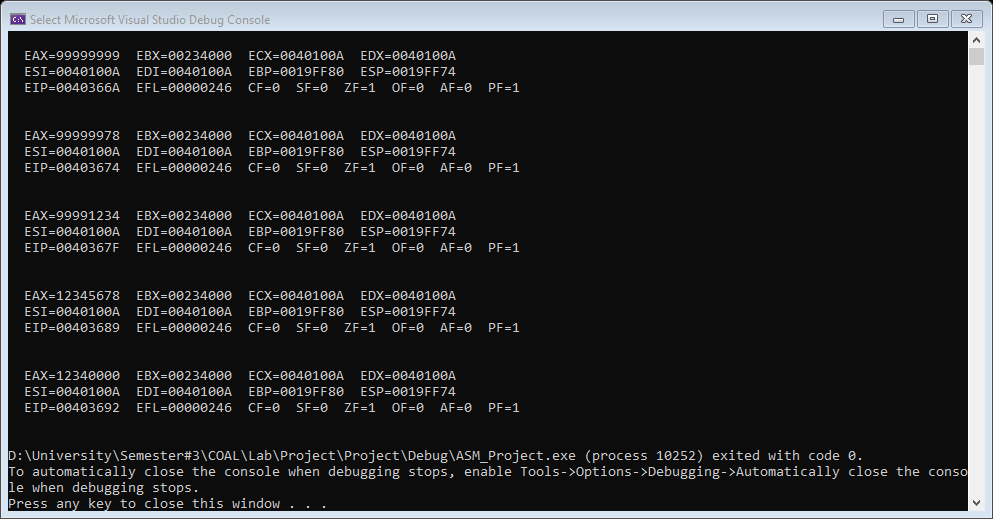
call DumpRegs

exit

main ENDP

END main

**Output:**



**Explanation:**

In data part, we have declared 3 variables using following,

* BYTE of 8-bit
* WORD of 16-bit
* DWORD 32-bit

In 1st step we have moved a value in eax and have called Dumregs which is used to print the values of all registers and flags. In next step, we have moved variable name “oneByte” in al because oneByte is initialized as BYTE and al only accepts 8-bit integers etc and again we have called Dumregs to print all the values of registers. In next step, we have moved the variable named “oneWord” in ax as WORD is of 16-bits and ax accepts only 16-bit integers etc. Similarly, we have again called Dumregs to print all the values of registers. In next step, we have moved variable named “oneDword” in eax as DWORD is of 32-bits and DWORD accepts 32-bits integers. And again, we have called Dumregs to print all the values of registers. At end we have passed 0 to ax which will make all lower values of 16-bit 0.

**Question 2**

**Code:**

INCLUDE Irvine32.inc

.data

oneDword Dword 12345678h

oneByte Byte 11h

.code

main PROC

mov eax, oneDword ; EAX = 123456h

call DumpRegs

movzx eax, oneByte ; EAX = 000011h

call DumpRegs

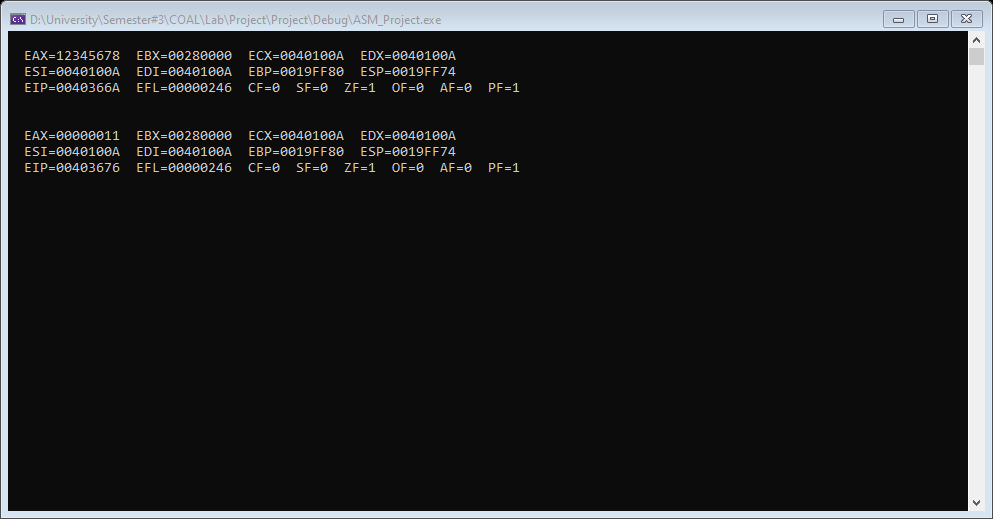
call ReadInt

exit

main ENDP

END main

**Output:**

****

**Explanation:**

In the .data part we have declared 2 variables. “oneByte” is declared as BYTE and and “oneDword” as DWORD. In .data part in the 1st step we have moved oneDword in eax. We can’t use movzx here because it does not work for 32-bits. In the next step we have called Dumregs which is used to print the values of all registers. movzx is basically used to copy smaller values to larger values and it can only be used with unsigned integers. In next step, we have passed 8-bit value which is declared as BYTE in eax and called Dumregs which worked because of movzx.

**Question 3:**

**Code:**

INCLUDE Irvine32.inc

.data

oneDword SDword 00101111100111001001001010111100b

oneByte SBYTE 10001001b

.code

main PROC

mov eax, oneDword ; EAX = 00101111100111001001001010111100

movsx eax, oneByte ; EAX = 11111111111111111111111110001001

call Dumregs

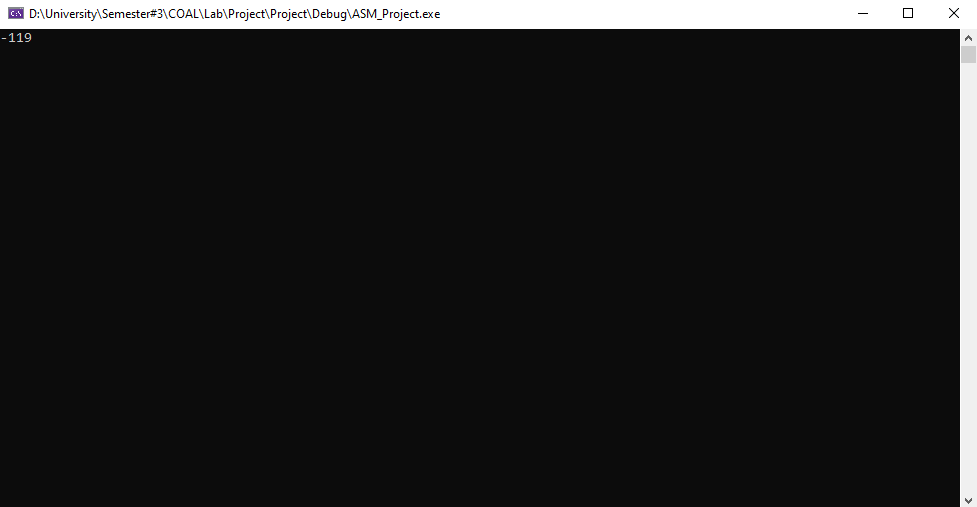
call writeInt

exit

main ENDP

END main

**Output:**

****

**Explanation:**

In the .data part we have declared “oneDword” as SDWORD and “oneByte” as SBYTE. In SBYTE s stands for signed, meaning it is used for signed integers. Similarly, movsx which works same as moxzv except that it is used for signed integers. The left most value indicates the sign of the value i.e. 1 is for negative and 0 is for positive. In .code part we have first moved oneDword to eax. In next step, we have moved oneByte which is declared as SBYTE in eax using movsx whose function is described above. After that call dumpregs and writeint to print all the registers and the value of eax respectively.

**Question 4:**

**Code:**

INCLUDE Irvine32.inc

.data

a dword 1111h

b dword 2222h

.code

main PROC

Mov Eax,a

Mov Ebx,b

Call DumpRegs

Xchg Eax,Ebx

Call DumpRegs

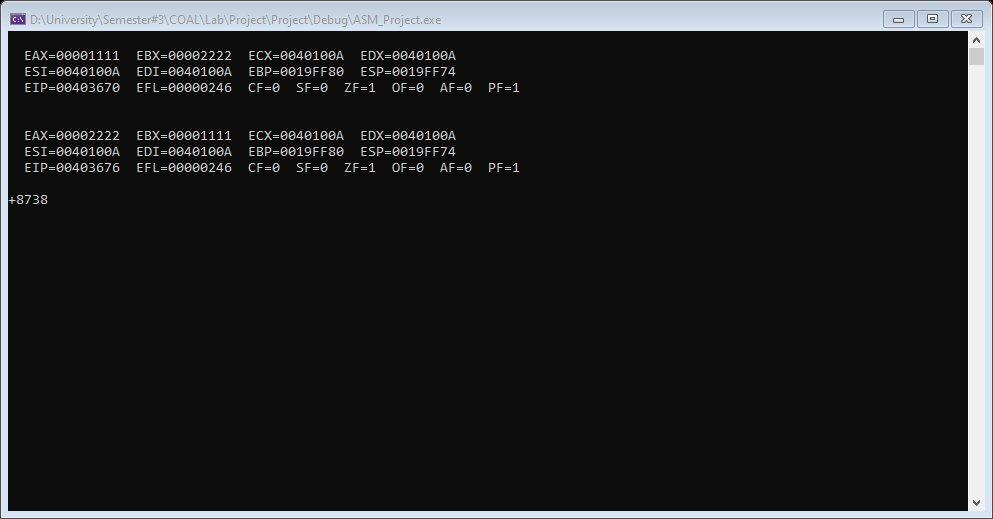
Call writeInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

In .data, 2 variables i.e. a & b are declared as DWORD. In .code a and b are moved to eax and ebx respectively. After that Dumpregs is called to print all the registers. XCHNG is used to exchange the contents of 2 operands. It works same as mov except that it does not accept immediate operands. After that the eax and ebx registers are exchanged using xchng and again Dumpregs and writeint is used to print all the registers and eax register respectively.

**Question 5:**

**Code:**

INCLUDE Irvine32.inc

.data

a dword 1111h

.code

main PROC

Mov Eax,a

Call DumpRegs

Inc Eax

Call DumpRegs

Dec Eax

Call DumpRegs

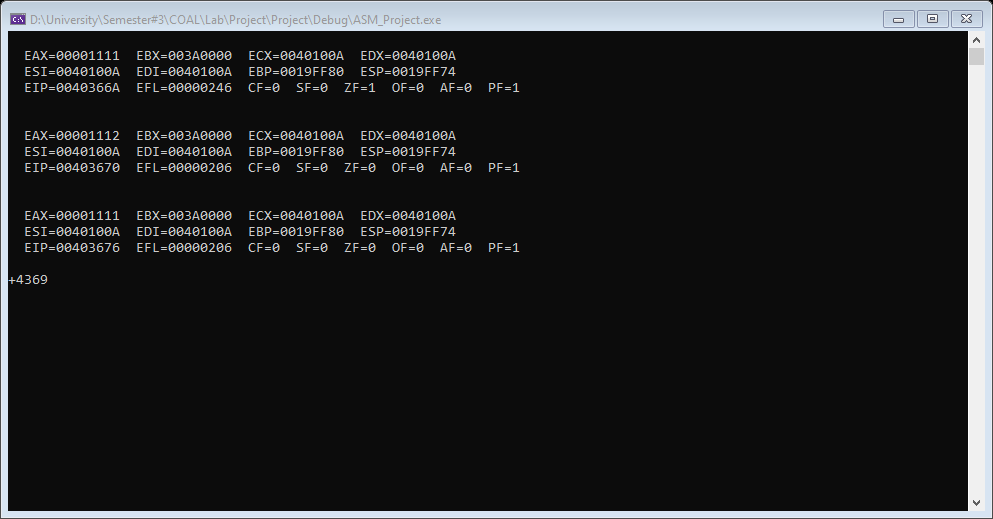
Call writeInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

In the 1st part i.e. .data a variable a is declared as DWORD. In the next part i.e. .code 1st a is moved in eax and then Dumpregs is called to print all the registers. Inc is used to increment 1 in the value passed in the eax and similarly dec is used to decrement 1 value. In the code next there is an increment of 1 in eax. After that again Dumpregs is called and then there is a decrement of 1 in eax and Dumpregs and writeint is called.

**Question 6:**

**Code:**

INCLUDE Irvine32.inc

.data

.code

main PROC

Mov Eax,0h

add Eax,10h

Call DumpRegs

Sub Eax, 10h

Call DumpRegs

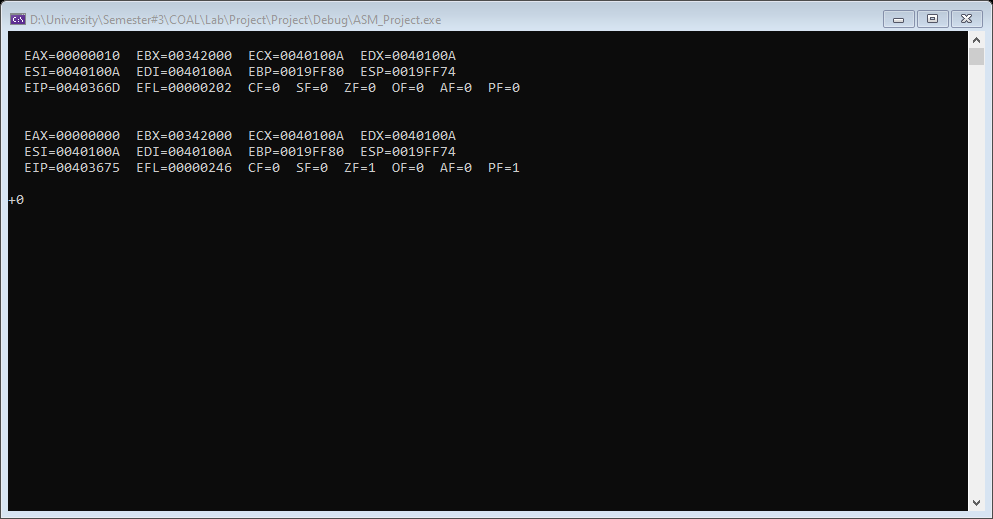
Call writeInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

In the above program 1st 0 is passed to eax and then 10 is added to eax. After that Dumpregs is called. Then 10 is subtracted from 10 again Dumpregs is called which shows that the output is 0. Which shows that zero flag is set because it is set when the arithmetic operation is 0.

**Question 7:**

**Code:**

INCLUDE Irvine32.inc

.data

a BYTE 255

b BYTE 1

.code

main PROC

Call DumpRegs

Mov al,a

add al,b

Call DumpRegs

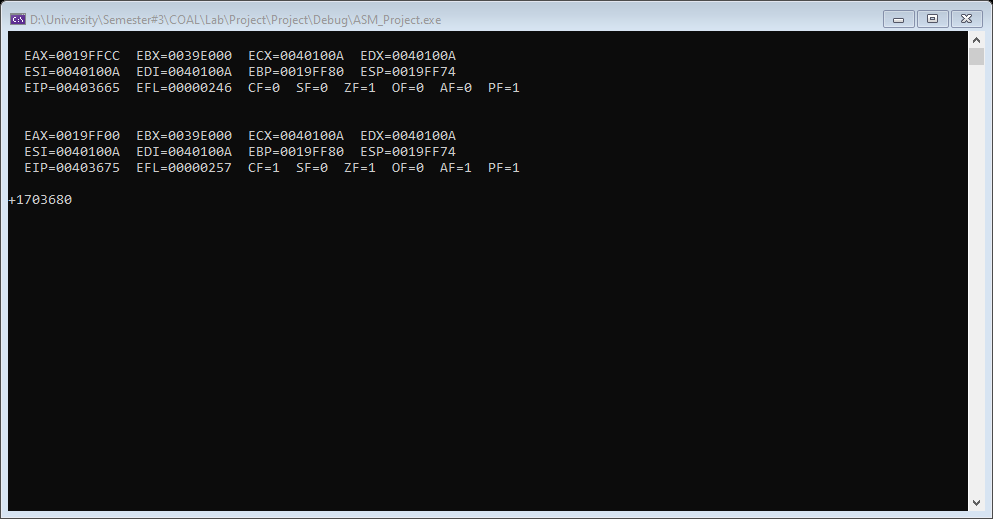
Call writeInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

In this program 2 variables are declared as BYTE i.e. a and b. In .data part a is moved in al which has a value of 255 (the last decimal value) and then b that has a value of 1 is added in it. When such thing happens al is overflowed because it cannot store a value greater than 255 and 1 is stored in carry flag.

**Question 8:**

**Code:**

INCLUDE Irvine32.inc

.data

.code

main PROC

Call DumpRegs

Mov Eax, 5

Sub Eax, 10

Call DumpRegs

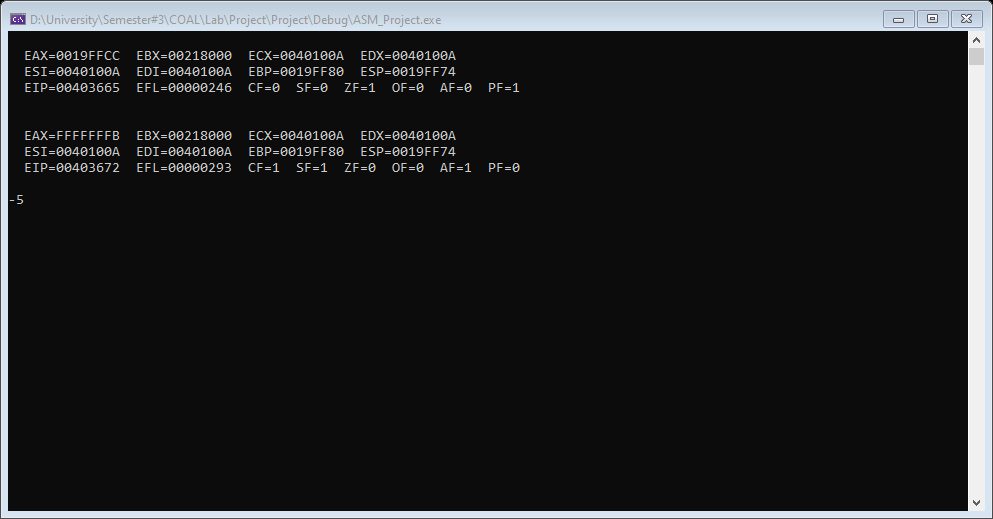
Call ReadInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

In this program 1st 5 is moved in eax register. After that 10 is subtracted from eax which gives a negative value. Sign flag is used when the result of a signed value is negative. In this program as the answer is a negative integer it will cause the flag to be 1.

**Question 9:**

**Code:**

INCLUDE Irvine32.inc

.data

.code

main PROC

Call DumpRegs

Mov AL, -128

Sub AL, 1

Call DumpRegs

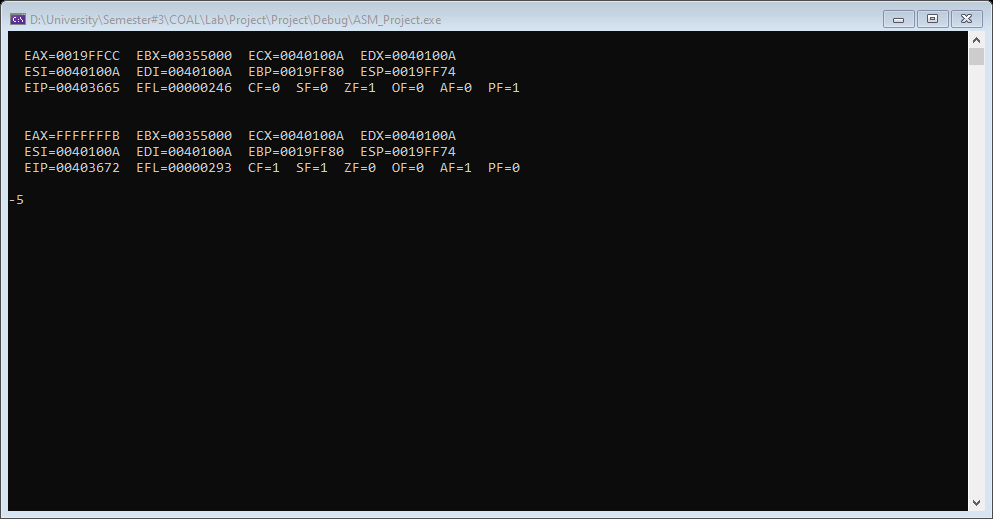
Call ReadInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

The overflow flag is set when a result of signed arithmetic operation overflows or underflows. An operation overflows when any value is added in the maximum value, similarly an operation, underflows when a value is subtracted from the least value. In the above code the least value i.e. 128 is moved in al and then 1 is subtracted with it which will set the overflow flag because the value is supposed to become 129 which is not possible to be stored in al.

**Question 10:**

**Code:**

INCLUDE Irvine32.inc

.data

a BYTE 010000b

b BYTE 000100b

d BYTE 000001b

.code

main PROC

Mov AL,a

Add AL,b

Call DumpRegs

Add AL,d

Call DumpRegs

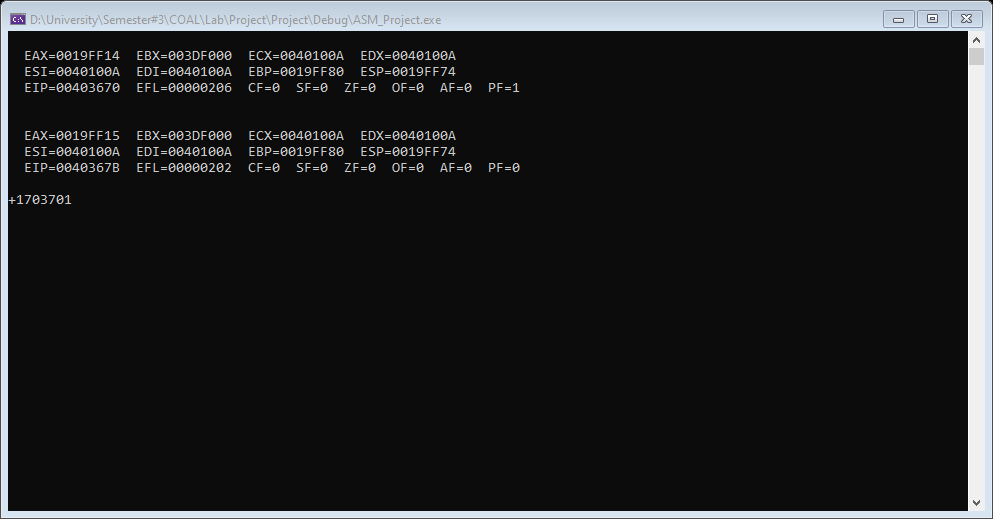
Call ReadInt

Exit

main ENDP

END main

**Output:**

****

**Explanation:**

The Parity flag (PF) is set when the least significant byte of the destination has an even number of 1 bit, immediately after an arithmetic or Boolean instruction has executed. In this code 3 variables a, b and c are declared as BYTE. In .code part a is moved in al and then b is added to al and then dumpregs is called. Similarly again d is added in al and again dumpregs and and writeint is called to display the registers and flags.