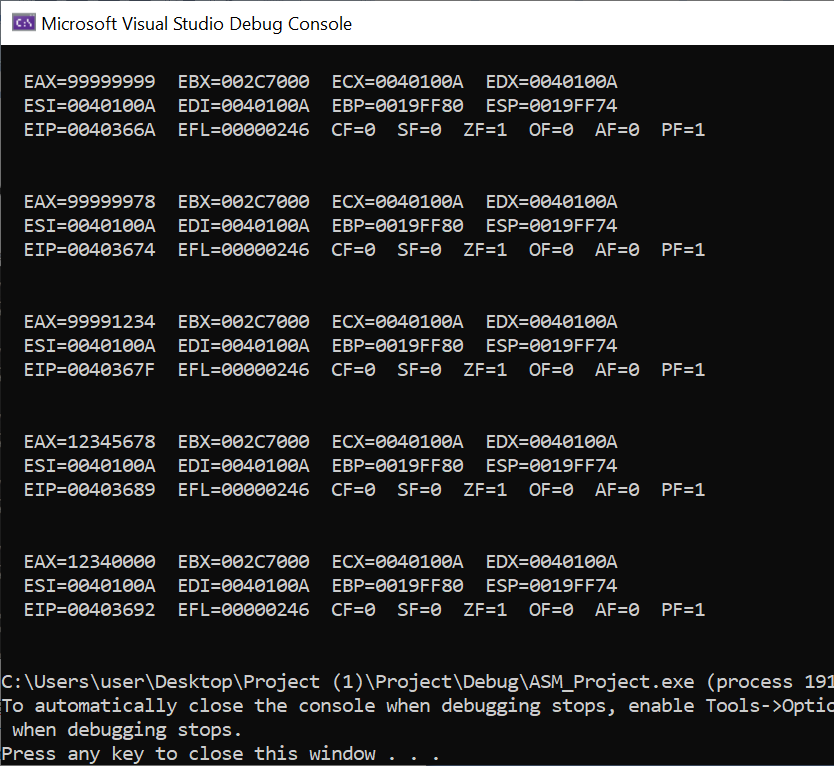
**Example – OVERLAPPING**



INCLUDE Irvine32.inc

.data

oneByte BYTE 78h

oneWord WORD 1234h

oneDword DWORD 12345678h

.code

main PROC

mov eax,99999999h

call DumpRegs

mov al,oneByte

call DumpRegs

mov ax,oneWord

call DumpRegs

mov eax,oneDword

call DumpRegs

mov ax,0

call DumpRegs

exit

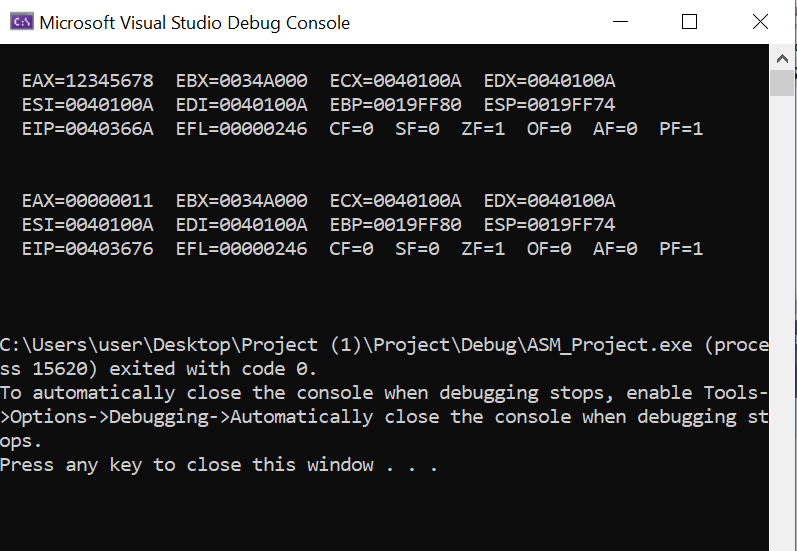
main ENDP

END main

**Explanation:**

Firstly there are 3 variables being initialized oneByte, oneWord, oneDword . Then the hexadecimal value 99999999 is being stored in the register “eax”. After this calling “DumpRegs” gave us the values stored in all the registers and flags . Here we can see that “eax” has 99999999 stored in it while the other registers and flags have some predefined/imaginary values stored . Now in the line “mov al,oneByte” the value of oneByte variable is stored in al of eax, as it is only of 8 bits and is the lowest byte of ax so only that part will change and value of eax will become 99999978 .Moving forward in the line “mov ax,oneWord” the value of oneWord variable is stored in ax and value of eax will become 99991234.Then “mov eax,oneDword” will store 12345678 in eax . And “mov ax,0” will store 0 in ax part ie. The last 16 bits of eax(32-bits) , so the value of eax will be altered to 12340000.

**Example – MOVZX**



INCLUDE Irvine32.inc

.data

oneDword Dword 12345678h

oneByte Byte 11h

.code

main PROC

mov eax, oneDword

call DumpRegs

movzx eax, oneByte

call DumpRegs

call ReadInt

exit

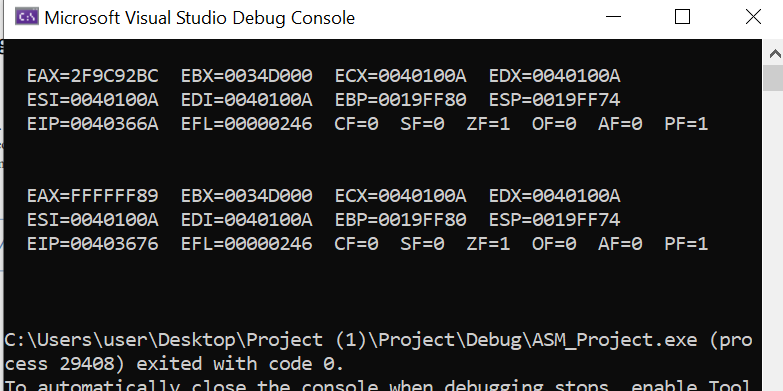
main ENDP

END main

**Explanation:**

Firstly there are variables oneDword, oneByte being initialized with 12345678h, 11h respectively .In “mov eax, oneDword” 12345678h is stored in eax. Then “movzx eax, oneByte” will store oneByte (an 8 bit operand) to the register eax( a 32-bit register) with zero extend, so eax will have 00000011 stored in it.Then calling DumpRegs will give us the values stored in all the registers and flags .

**Example – MOVSX**



INCLUDE Irvine32.inc

.data

oneDword SDword 00101111100111001001001010111100b

oneByte SBYTE 10001001b

.code

main PROC

mov eax, oneDword

; EAX = 00101111100111001001001010111100

call DumpRegs

movsx eax, oneByte

; EAX = 11111111111111111111111110001001

call DumpRegs

call ReadInt

exit

main ENDP

END main

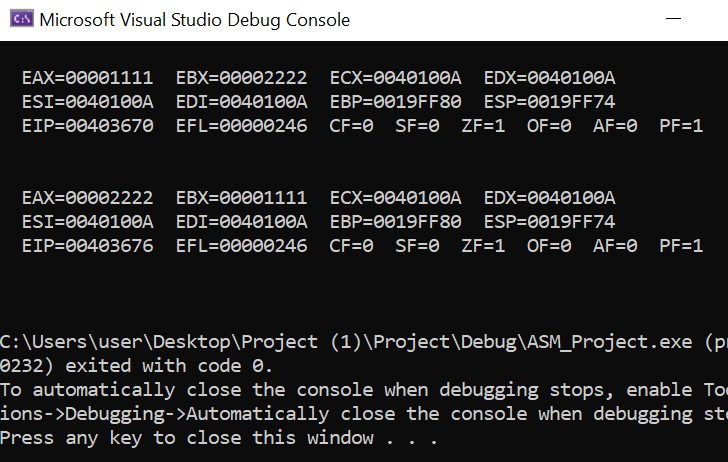
**Explanation:**

Firstly signed variables oneDword, oneByte are initialized by 00101111100111001001001010111100b, 10001001b respectively. In “mov eax, oneDword” 00101111100111001001001010111100b is stored in eax. As DumpRegs shows the values in hexadecimal so 0010 becomes 2, 11111 becomes F, 1001 becomes 9, 1100 becomes C, 1001 becomes 9, 0010 becomes 2, 1011 becomes B and 1100 becomes C . So , eax becomes 2F9C92BC . Then in “movsx eax, oneByte” as oneByte is a negative number indicated by

->10001001b , here movsx will store 10001001 by extended 1s which means eax has 11111111111111111111111110001001. And converting that to hexadecimal it becomes (1111)F 1111(F) (1111)F (1111)F (1111)F (1111)F (1000)8 (1001)9 .

**Example – Operand Exchange**

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 ReadInt

Exit

main ENDP

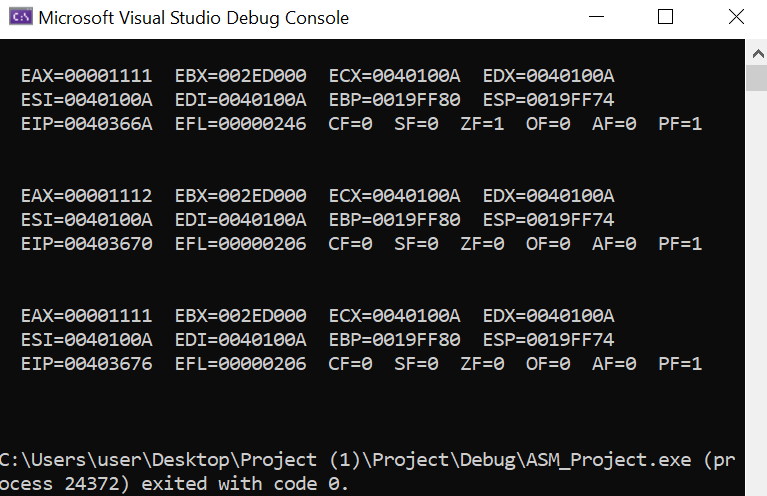
END main

**Explanation:**

Variables( dword) “a” and “b” are initialitialized by 1111h, and 2222h. In “Mov Eax,a” value of a is stored in eax register with extended 0s as it is unsigned ie. EAX=00001111. Then in “Mov Ebx,b” value b is stored in ebx register with extended 0s as it is unsigned ie. EBX=00002222 . Then using “Xchg” in line “Xchg Eax,Ebx” the values of registers EAX and EBX are exchanged . This means EAX becomes 00002222 and EBX becomes 00001111. We can observe the alteration of values in registers by using “Call DumpRegs”.

**Example – Increment Decrement**

INCLUDE Irvine32.inc



.data

a dword 1111h

.code

main PROC

Mov Eax,a

Call DumpRegs

Inc Eax

Call DumpRegs

Dec Eax

Call DumpRegs

Call ReadInt

Exit

main ENDP

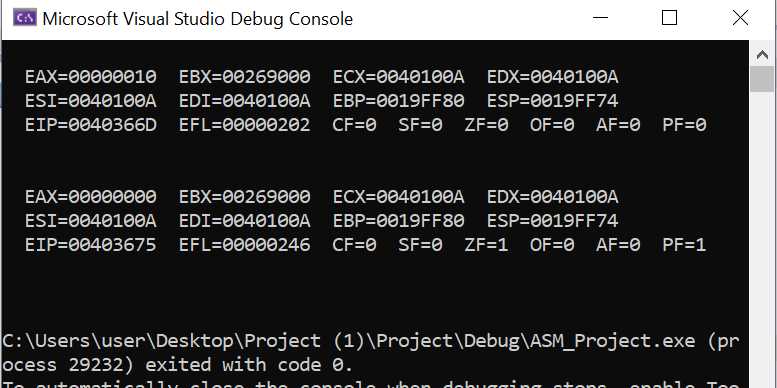
END main

**Explanation:**

Variable ( dword) “a” is initialitialized with 1111h . In “Mov Eax,a” value of a is stored in eax register with extended 0s as it is unsigned ie. EAX=00001111. . Then using “Inc ” in line “Inc Eax” the value of registers EAX is increased by one ie. EAX=00001111+00000001 =00001112. Then in line “Dec Eax” ,“Dec” is used to decrease the value of Eax by one ie. EAX=00001112-00000001 =00001111. Console is displaying the alterations in the values of the registers by using “Call DumpRegs” which give us the values stored in all the registers and flags.

**Example – Zero Flag**

INCLUDE Irvine32.inc



.data

.code

main PROC

Mov Eax,0h

add Eax,10h

Call DumpRegs

Sub Eax, 10h

Call DumpRegs

Call ReadInt

Exit

main ENDP

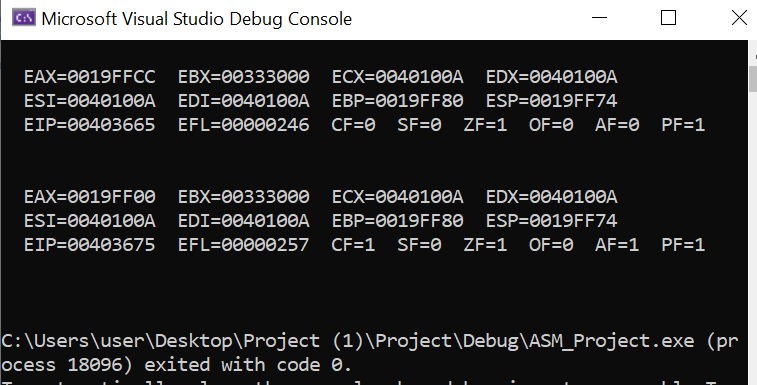
END main

**Explanation:**

In “Mov Eax,0h” 00000000 is stored in eax. Then in “add Eax,10h” 00000010 is added to EAX so EAX becomes EAX=00000000+00000010=00000010 .Here “Call DumpRegs” (which give us the values stored in all the registers and flags) will show the state of “Zero Flag” is “0”. Moving forward in line “Sub Eax, 10h ” 00000010 is substracted from EAX so EAX becomes EAX=00000010-00000010=00000000. Here as the value of EAX becomes 0 after substraction the state of “Zero Flag” becomes “1” which means it is set.

**Example – Carry Flag**

INCLUDE Irvine32.inc



.data

a BYTE 255

b BYTE 1

.code

main PROC

Call DumpRegs

Mov al,a

add al,b

Call DumpRegs

Call ReadInt

Exit

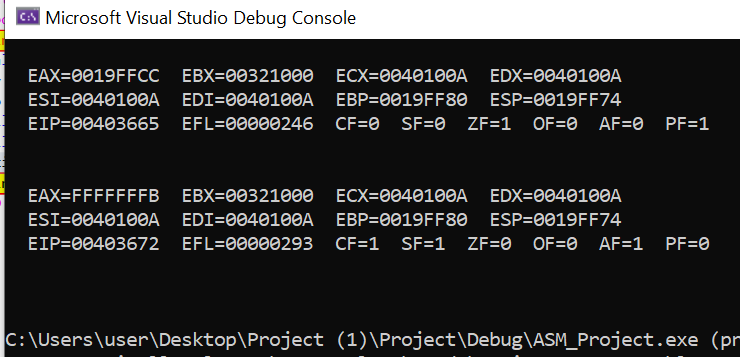
main ENDP

END main

**Explanation:**

Firstly , “a” and “b” are initialized by 255 and 1, respectively. Line “Call DumpRegs” shows the values of all the registers and states of all the registers. Then “Mov al,a” stores a(255) in “al” and then “add al,b” will add b(1) in “al” ,as AL cannot store a value greater than 255 so it will overflow. Due to this overflow the state of “Carry Flag” is set to 1. The line “Call DumpRegs” (which give us the values stored in all the registers and flags) is showing the state of “Carry Flag”.

**Example – Sign Flag**



INCLUDE Irvine32.inc

.data

.code

main PROC

Call DumpRegs

Mov Eax, 5

Sub Eax, 10

Call DumpRegs

Call ReadInt

Exit

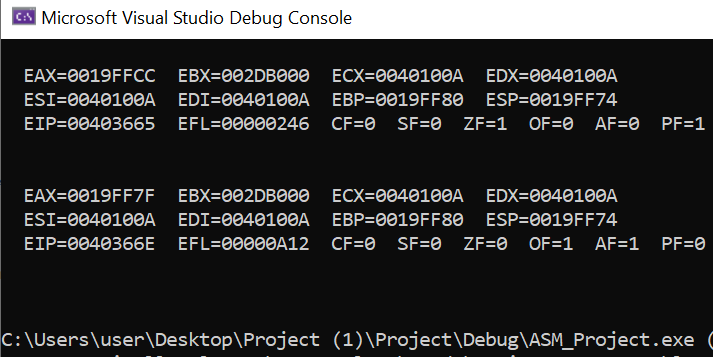
main ENDP

END main

**Explanation:**

In “Mov Eax, 5” 5 is stored in EAX. Then “Sub Eax, 10” substracts 10 from EAX , as 10 is greater than 5 EAX becomes negative and as the result of the arithmetic operation is negative hence the state of “Sign Flag” becomes “1”. We can observe the above alterations by calling “Call DumpRegs” (which give us the values stored in all the registers and flags).

**Example – Overflow Flag (Signed Underflow)**



INCLUDE Irvine32.inc

.data

.code

main PROC

Call DumpRegs

Mov AL, -128

Sub AL, 1

Call DumpRegs

Call ReadInt

Exit

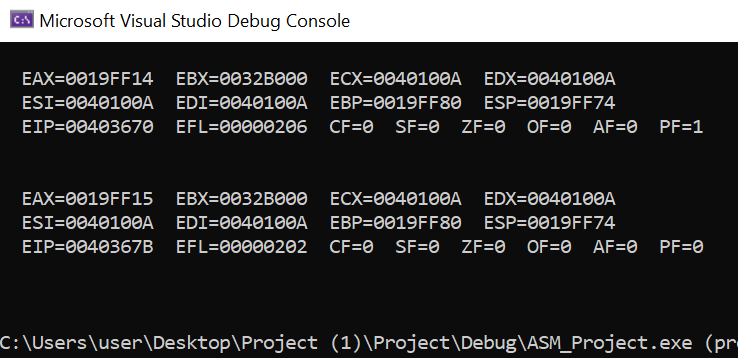
main ENDP

END main

**Explanation:**

In “Mov AL, -128” -128 is stored in AL and then in “Sub AL, 1” 1 is substracted from “AL” and as “AL” cannot store a value smaller than “-128” it results in underflow . And then due to this underflow the state of “Overflow Flag” is set to “1”. As shown on the console by calling “DumpRegs” (which give us the values stored in all the registers and flags).

**Example – Parity Flags**



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

**Explanation:**

Firstly a, b, c are initialized with 010000b, 000100b and 000001b respectively . In “Mov AL,a” value of a(010000b) is stored in al ,here no. Of 1s are odd .Then in “Add AL,b” b is added in al ,here no. of 1s become even , hence the state of the “Parity Flag” becomes “1”. Then in “Add AL,d” we add d(000001) in al and due to this the no. Of 1s becomes odd , as a result the state of the “ Parity flag” becomes “0”. As shown on the console by calling “DumpRegs” (which give us the values stored in all the registers and flags).