LAB 09

Integer Arithmetic

STUDENT NAME _____ ROLL NO _____ SEC _____

SIGNATURE & DATE _____

MARKS AWARDED: /

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Lab Session 09: Integer Arithmetic

Learning Objectives

a. Shift & rotate Instructions
b. Multiplication and Division
c. Extended Addition and Subtraction

Shift and Rotate Instructions

The 8086-based processors provide a complete set of instructions for shifting and rotating bits.

- **Shift Instructions:**

Shift instructions move bits a specified number of places to the right or left. The last in the direction of the shift goes into the carry flag, and the first bit is filled with 0 or with the previous value of the first bit.

- **SHL Instruction**

This instruction performs a logical left shift on the destination operand, filling the lowest bit with 0. The highest bit is moved to the Carry flag, and the bit that was in the Carry flag is discarded.

Syntax : SHL destination,count

The following lists the types of operands permitted by this instruction:

SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL

Example:

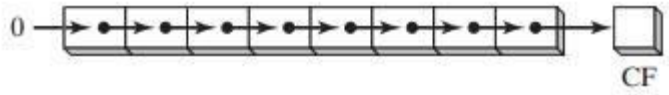
```
mov bl,8Fh          ;BL=10001111b
SHL bl,1            ;CF=1, BL=00011110b

mov al,10000000b    ;AL=10000000b
SHL al,2            ;CF=0, AL=00000000b
```

Bit Multiplication Example: SHL can perform multiplication by powers of 2. Shifting any operand left by *n* bits multiplies the operand by 2^n. For example, shifting the integer 5 left by 1 bit yields the product of $5 \times 2^1 = 10$:

```
mov dl,5                    ;DL=00000101b       =5
SHL dl,1                    ;CF=0, DL=00001010b =10
```

- **SHR Instruction**

The SHR (shift right) instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0. The lowest bit is copied into the Carry flag, and the bit that was previously in the Carry flag is lost.



Examples:

```
mov al,0D0h                 ; AL = 11010000b
shr al,1                    ; AL = 01101000b, CF = 0

mov al,00000010b
shr al,2                    ; AL = 00000000b, CF = 1
```

Bitwise Division

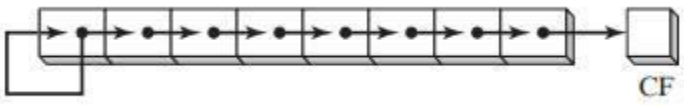
Bitwise Division Logically shifting an unsigned integer right by *n* bits divides the operand by 2^n. In the following statements, we divide 32 by 2^1, producing 16:

```
mov dl,32                   ;DL=00100000b       =32
SHR dl,1                    ;DL=00010000b, CF=0 =16
```

- **SAL and SAR Instructions.**

The SAL (shift arithmetic left) instruction works the same as the SHL instruction.

The SAR (shift arithmetic right) works like:



The following example shows how SAR duplicates the sign bit. AL is negative before and after it is shifted to the right:

```
mov al, 0F0h                ; AL = 11110000b (-16)
sar al,1                    ; AL = 11111000b (-8), CF = 0
```

Sign division:

```
mov dl,-128          ; DL = 10000000b
sar dl,3             ; DL = 11110000b
```

Sign-Extend AX into EAX:

```
mov ax,-128          ; EAX = ????FF80h
shl eax,16           ; EAX = FF800000h
sar eax,16           ; EAX = FFFFFF80h
```

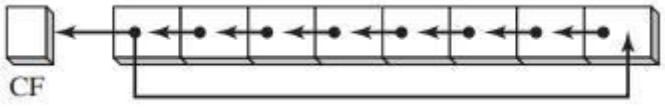
Instruction	CL	Initial Contents		Final Contents		
		Decimal	Binary	Decimal	Binary	CF
SHR AL,1		250	11111010	125	01111101	0
SHR AL,CL	3	250	11111010	31	00011111	0
SHL AL,1		23	00010111	46	00101110	0
SHL BL,CL	2	23	00010111	92	01011100	0
SAL BL,1		+23	00010111	+46	00101110	0
SAL DL,CL	4	+3	00000011	+48	00110000	0
SAR AL,1		-126	10000010	-63	11000001	0
SAR AL,CL	2	-126	10000010	-32	11100000	1

- **Rotate Instructions:**

Rotate instructions also move bits a specified number of places to the right or left. For each bit rotated the last bit in the direction of the rotate operation moves into the first bit position at the other end of the operand. With some variations, the carry bit is used as an additional bit of the operand. **RCR** (Rotate Carry Right) and **RCL** (Rotate Carry Left) instructions carry values from the first register to the second by passing the leftmost or rightmost bit through the carry flag.

- **ROL Instruction**

The ROL (rotate left) instruction shifts each bit to the left. The highest bit is copied into the Carry flag and the lowest bit position. The instruction format is the same as for SHL:



Example:

```
mov al,40h           ; AL = 01000000b
rol al,1             ; AL = 10000000b, CF = 0
```

```
rol al,1                ; AL = 00000001b, CF = 1
rol al,1                ; AL = 00000010b, CF = 0
```

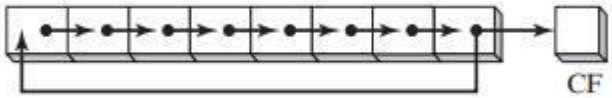
Exchanging Groups of Bits You can use ROL to exchange the upper (bits 4–7) and lower (bits 0–3) halves of a byte. For example, 26h rotated four bits in either direction becomes 62h:

```
mov al,26h
rol al,4                ; AL = 62h
```

- **ROR Instruction**

The ROR (rotate right) instruction shifts each bit to the right and copies the lowest bit into the Carry flag and the highest bit position.

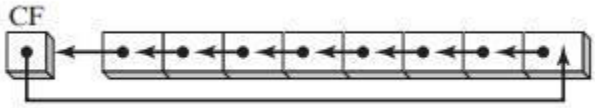
Example:




```
mov al,01h              ; AL = 00000001b
ror al,1                ; AL = 10000000b, CF = 1
ror al,1                ; AL = 01000000b, CF = 0
```

- **RCL Instructions**

The RCL (rotate carry left) instruction shifts each bit to the left, copies the Carry flag to the LSB, and copies the MSB into the Carry flag:



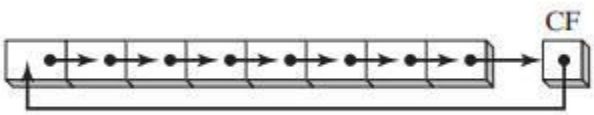
Example:

```
clc                     ;CF=0
mov bl,88h              ; CF, BL = 0 10001000b
rcl bl,1                ; CF,BL = 1 00010000b
```

- **RCR Instruction:**

The RCR (rotate carry right) instruction shifts each bit to the right, copies the Carry flag into the MSB, and copies the LSB into the Carry flag

Example:



```
stc                 ;CF=1
mov ah,10h          ; AH, CF = 00010000 1
rcr ah,1            ; AH, CF = 10001000 0
```

Instruction	CL	Initial Contents		Final Contents	
		CF	Binary	Binary	CF
ROR AL,1		0	11111010	01111101	0
ROR AL,CL	3	1	11111010	01011111	0
ROL AL,1		0	00010111	00101110	0
ROL BL,CL	2	1	00010111	01011100	0
RCL BL,1		0	00010111	00101110	0
RCL DL,CL	4	1	00000011	00111000	0
RCR AL,1		1	10000010	11000001	0
RCR AL,CL	2	0	10000010	00100000	1

APPLICATIONS:

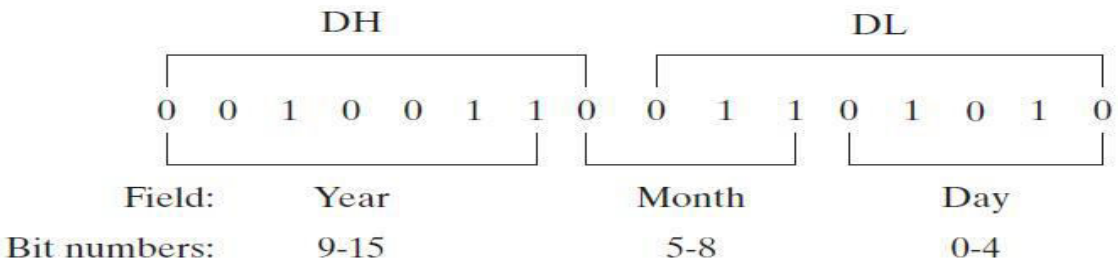
1. Binary Multiplication

$$EAX * 36 = EAX * (2^5 + 2^2)$$
$$= EAX * (32 + 4)$$
$$= (EAX * 32) + (EAX * 4)$$

```
.code
mov eax,123
mov ebx,eax
shl eax,5       ; mult by 25
shl ebx,2       ; mult by 22
add eax,ebx     ; add the products
```

```
    01111011      123
×   00100100      36
  01111011        123 SHL 2
+ 01111011        123 SHL 5
0001000101001100  4428
```

2. Isolating Data Fields



The following code example extracts the day number field of a date stamp integer by making a copy of DL and masking off bits not belonging to the field:

```
mov  al,dl                    ; make a copy of DL
and  al,00011111b             ; clear bits 5-7
mov  day,al                   ; save in day
```

To extract the month number field, we shift bits 5 through 8 into the low part of AL before masking off all other bits. AL is then copied into a variable:

```
mov  ax,dx                    ; make a copy of DX
shr  ax,5                     ; shift right 5 bits
and  al,00001111b             ; clear bits 4-7
mov  month,al                 ; save in month
```

The year number (bits 9 through 15) field is completely within the DH register. We copy it to AL and shift right by 1 bit:

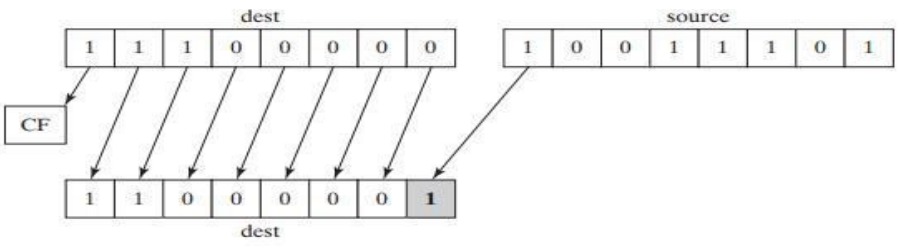
```
mov  al,dh                    ; make a copy of DH
shr  al,1                     ; shift right one position
mov  ah,0                     ; clear AH to zeros
add  ax,1980                  ; year is relative to 1980
mov  year,ax                  ; save in year
```

- **SHLD Instruction**

The SHLD (shift left double) instruction shifts a destination operand a given number of bits to the left. The bit positions opened up by the shift are filled by the most significant bits of the source operand.

Format:

SHLD reg16, reg16, CL/imm8
SHLD mem16, reg16, CL/imm8
SHLD reg32, reg32, CL/imm8
SHLD mem32, reg32, CL/imm8

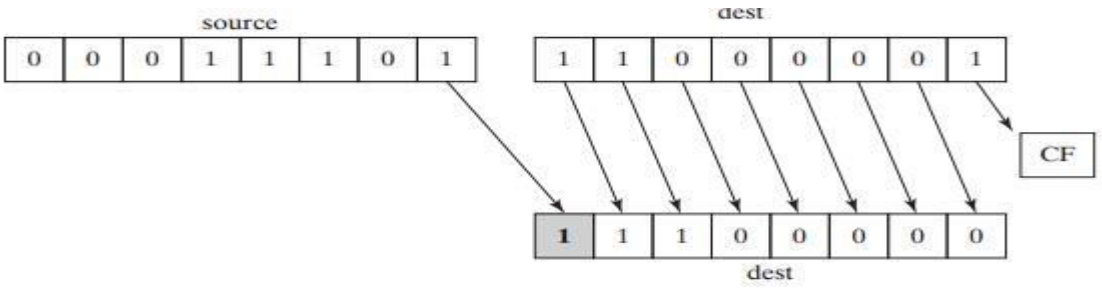


Example:

```
.data
        a WORD 9BA6h
.code
        mov ax, 0AC36h
        shld a, ax, 4                    ;a=BA6Ah
```

- **SHRD Instruction**

The SHRD (shift right double) instruction shifts a destination operand a given number of bits to the right. The bit positions opened by the shift are filled by the least significant bits of the source operand.



Example:

```
.code
        mov ax,234Bh
        mov dx,7654h
        shrd ax,dx,4                     ;ax=4234h
```

- **MUL Instruction**

The **MUL** instruction is for unsigned multiplication. Operands are treated as unsigned numbers. The three formats accept register and memory operands, but not immediate operands. The Carry flag is clear (CF = 0) because AH (the upper half of the product) equals zero. Syntax:

```
MUL reg/mem8
MUL reg/mem16
MUL reg/mem32
```

- The table represents MUL operands

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

EXAMPLE # 01:

```
INCLUDE Irvine32.inc

.code
main PROC
mov eax,0
mov ebx,0
mov al,5h
mov bl,10h
mul bl                          ; AX = 0050h, CF = 0

call crlf
call dumpregs
exit
main ENDP
END main
```

EXAMPLE # 02:

```
.data
        val1 WORD 2000h
        val2 WORD 0100h
.code
        mov ax,val1             ; AX = 2000h
        mul val2                ; DX:AX = 00200000h, CF = 0
```

EXAMPLE # 03:

```
        mov eax,12345h
        mov ebx,1000h
        mul ebx                 ; EDX:EAX = 0000000012345000h, CF = 0
```

- **IMUL Instruction**

The **IMUL** instruction is for signed multiplication. Operands are treated as signed numbers and result is positive or negative depending on the signs of the operands.
The x86 instruction set supports three formats for the IMUL instruction: one operand, two operands, and three operands.

- **One-Operand Formats:**

```
IMUL reg/mem8           ; AX = AL * reg/mem8
IMUL reg/mem16          ; DX:AX = AX * reg/mem16
IMUL reg/mem32          ; EDX:EAX = EAX * reg/mem32
```

- **Two-Operand Formats**

IMUL reg16, reg/mem16
IMUL reg16, imm8
IMUL reg16, imm16

- **Three-Operand Formats**

IMUL reg16, reg/mem16, imm8
IMUL reg16, reg/mem16, imm16
IMUL reg32, reg/mem32, imm8
IMUL reg32, reg/mem32, imm32

Example:

The following instructions multiply 48 by 4, producing +192 in AX. Although the product is correct, AH is not a sign extension of AL, so the Overflow flag is set:

```
mov al,48
mov bl,4
imul bl                    ;AX = 00C0h, OF = 1
```

The following instructions multiply -4 by 4, producing -16 in AX. AH is a sign extension of AL so the Overflow flag is clear:

```
.code
main PROC
mov eax,0
mov ebx,0
mov edx,0
mov ax,-2
mov bx,4                   ; EDX:EAX = FFFFFFF8h, OF = 0
imul bx
call crlf
call dumpregs
```

The following instructions demonstrate two-operand formats:

EXAMPLE :

```
INCLUDE Irvine32.inc
.data
        word1 SWORD 4
        dword1 SDWORD 4
.code
main PROC
mov eax,0
mov ebx,0
        mov ax,-4          ;AX=-4
        mov bx,2           ;BX=2
```

```
        call dumpregs
        imul bx,ax                          ;BX=-8
        call dumpregs
        imul bx,2                           ;BX=-16
        call dumpregs
        imul bx,word1                       ;BX=-64
        mov eax,-16                     ;
        mov ebx,2
        call dumpregs
        imul ebx,eax
        call dumpregs
        imul ebx,2
        call dumpregs
        imul ebx,dword1
        call dumpregs
exit
main ENDP
END main
```

The following instructions demonstrate three-operand formats:

Example:

```
INCLUDE Irvine32.inc
.data
        word1 SWORD 4
        dword1 SDWORD 4
.code
main PROC
mov ebx,0
imul bx,word1,-2
call dumpregs
imul ebx,dword1,-5
call dumpregs
exit
main ENDP
END main
```

- **DIV Instruction**

The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit unsigned integer division. The single register or memory operand is the divisor. The formats are

```
DIV reg/mem8
DIV reg/mem16
```

DIV reg/mem32

The following table shows the relationship between the dividend, divisor, quotient, and remainder:

Dividend	Divisor	Quotient	Remainder
AX	reg/mem8	AL	AH
DX:AX	reg/mem16	AX	DX
EDX:EAX	reg/mem32	EAX	EDX

Example:

```
mov ax,0083h        ; dividend
mov bl,2            ; divisor
div bl              ; AL = 41h, AH = 01h

mov dx,0            ; clear dividend, high
mov ax,8003h        ; dividend, low
mov cx,100h         ; divisor
div cx              ; AX = 0080h, DX = 0003h
```

Sign Extension Instructions(CBW,CWD,CDQ):

Dividends of signed integer division instructions must often be sign-extended before the division takes place. Intel provides three useful sign extension instructions: CBW, CWD, and CDQ.

The CBW instruction (convert byte to word) extends the sign bit of AL into AH, preserving the number's sign. In the next example, 9Bh (in AL) and FF9Bh (in AX) both equal −101 decimal:

EXAMPLE:

```
.data
        byteVal SBYTE -101          ; 9Bh
.code
        mov al,byteVal              ; AL = 9Bh
        cbw                         ; AX = FF9Bh
```

The CWD (convert word to doubleword) instruction extends the sign bit of AX into DX:

```
.data
        wordVal SWORD -101          ; FF9Bh
.code
        mov ax,wordVal              ; AX = FF9Bh
        cwd                         ; DX:AX = FFFFFF9Bh
```

The CDQ (convert doubleword to quadword) instruction extends the sign bit of EAX into EDX:

```
.data
        dwordVal SDWORD -101            ; FFFFFF9Bh
.code
        mov eax,dwordVal
        cdq                             ; EDX:EAX = FFFFFFFFFFFFF9Bh
```

- **IDIV Instruction**

The IDIV (signed divide) instruction performs signed integer division, using the same operands as DIV.

Example: The following instructions divide -48 by 5.

```
.data
        byteVal SBYTE -48               ; D0 hexadecimal
.code
        mov al,byteVal                  ; lower half of dividend
        cbw                             ; extend AL into AH
        mov bl,+5                       ; divisor
        idiv bl                         ;AL=-9,AH=-3
```

- ***ADC Instructions:***

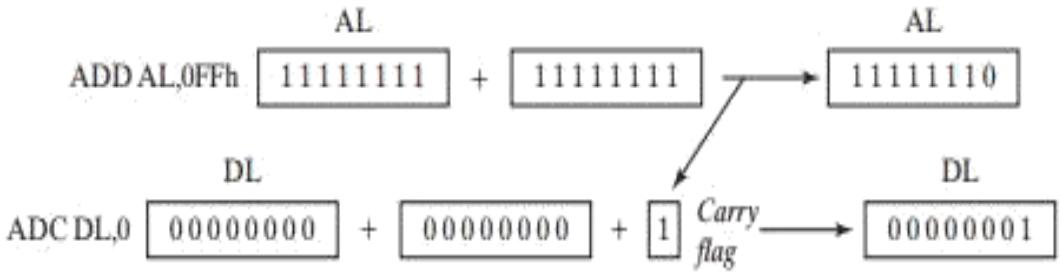
The ADC (add with carry) instruction adds both a source operand and the contents of the Carry flag to a destination operand.

Syntax: *ADC Destination, source*

ADC reg,reg
ADC mem,reg
ADC reg,mem
ADC mem,imm
ADC reg,imm

EXAMPLE # 01:

```
mov dl,0
mov al,0FFh
add al,0FFh                             ; AL = FEh
adc dl,0                                ; DL/AL = 01FEh
```



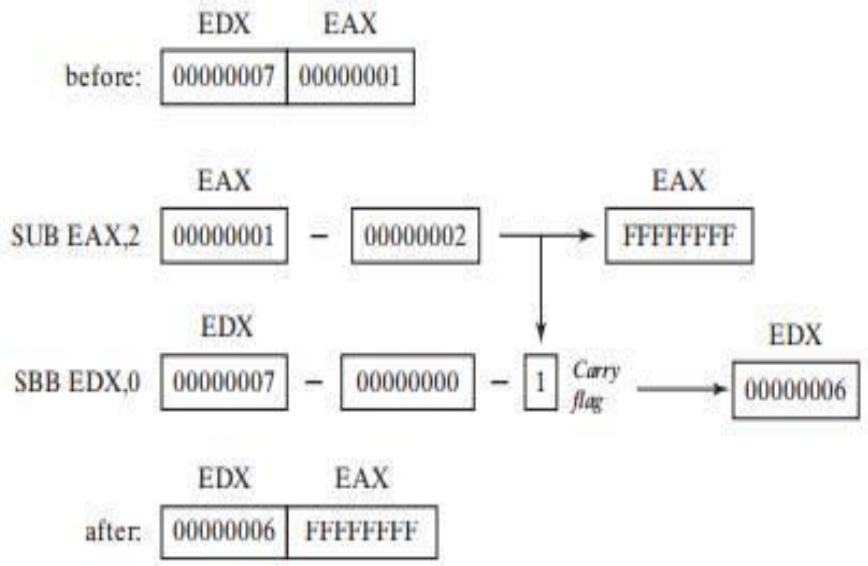
- **SBB Instructions:**

The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand.

Syntax: *SBB Destination, source*

EXAMPLE:

```
mov edx,7          ; upper half
mov eax,1          ; lower half
sub eax,2          ; subtract 2
sbb edx,0          ; subtract upper half
```



ACTIVITY:

Task#1

Write ASM instructions that calculate EAX * 21 using binary multiplication.
Hint: $21 = 2^4 + 2^2 + 2^0$.

Task#2

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

Task#3

The time stamp field of a file directory entry uses bits 0 through 4 for the seconds, bits 5 through 10 for the minutes, and bits 11 through 15 for the hours. Write instructions that extract the minutes and copy the value to a byte variable named **bMinutes**.

Task#4

Write a series of instructions that shift the lowest bit of AX into the highest bit of BX without using the SHRD instruction. Next, perform the same operation using SHRD.

Task#5

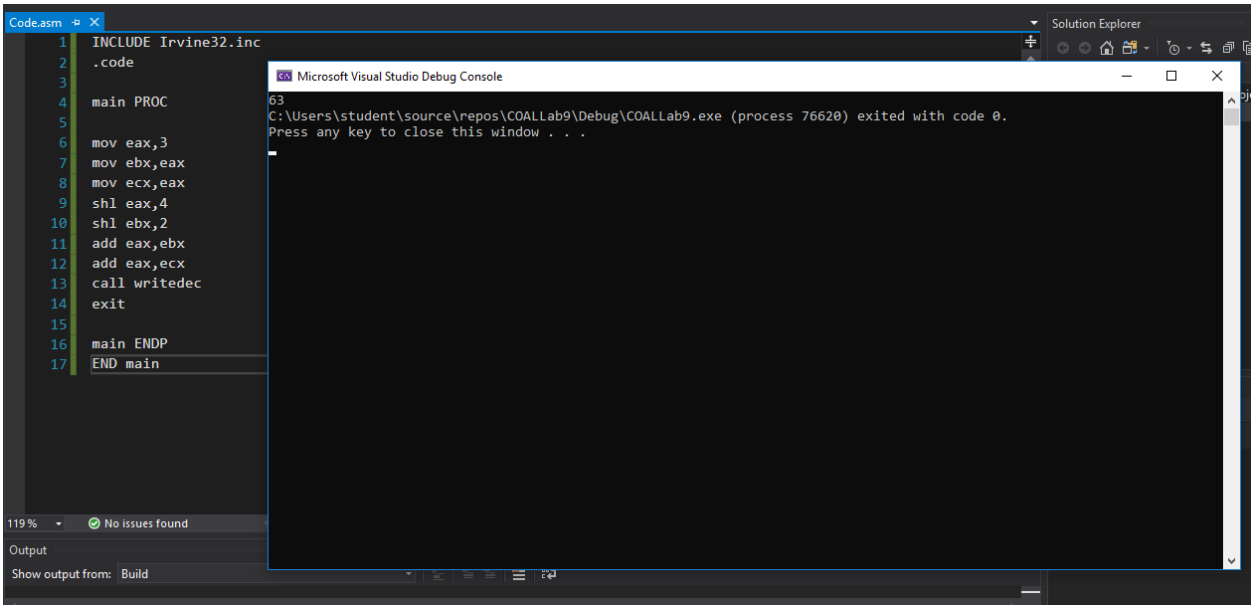
Implement the following C++ expression in assembly language, using 32-bit signed operands:

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

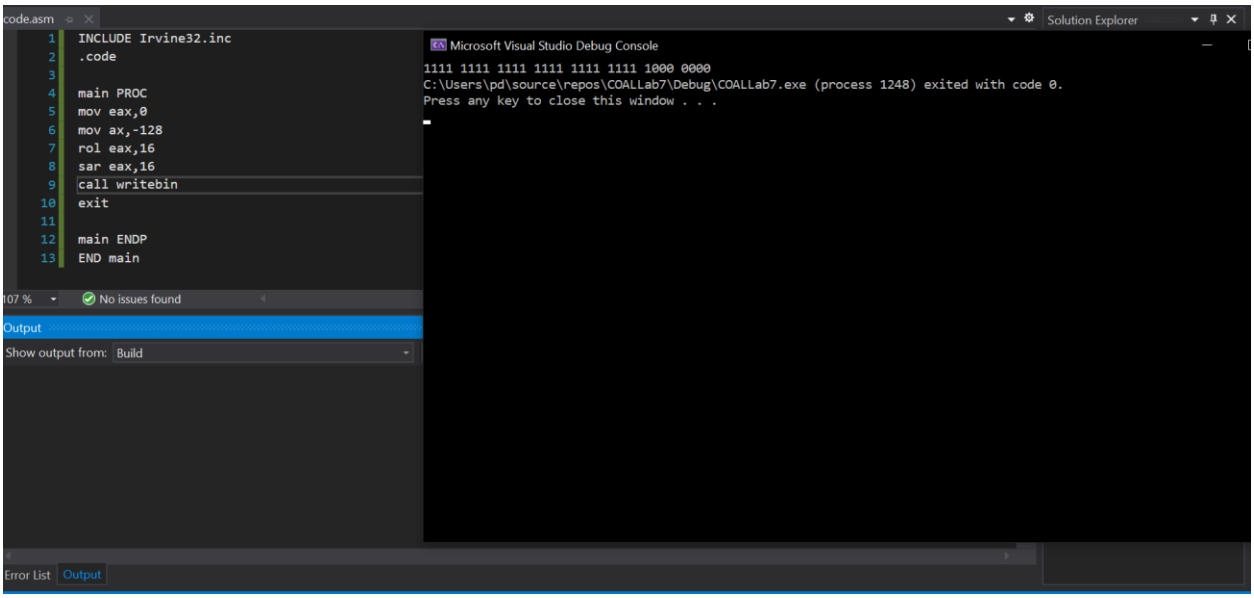
Task#6

Create a procedure **Extended_Add** procedure to add two 64-bit (8-byte) integers.

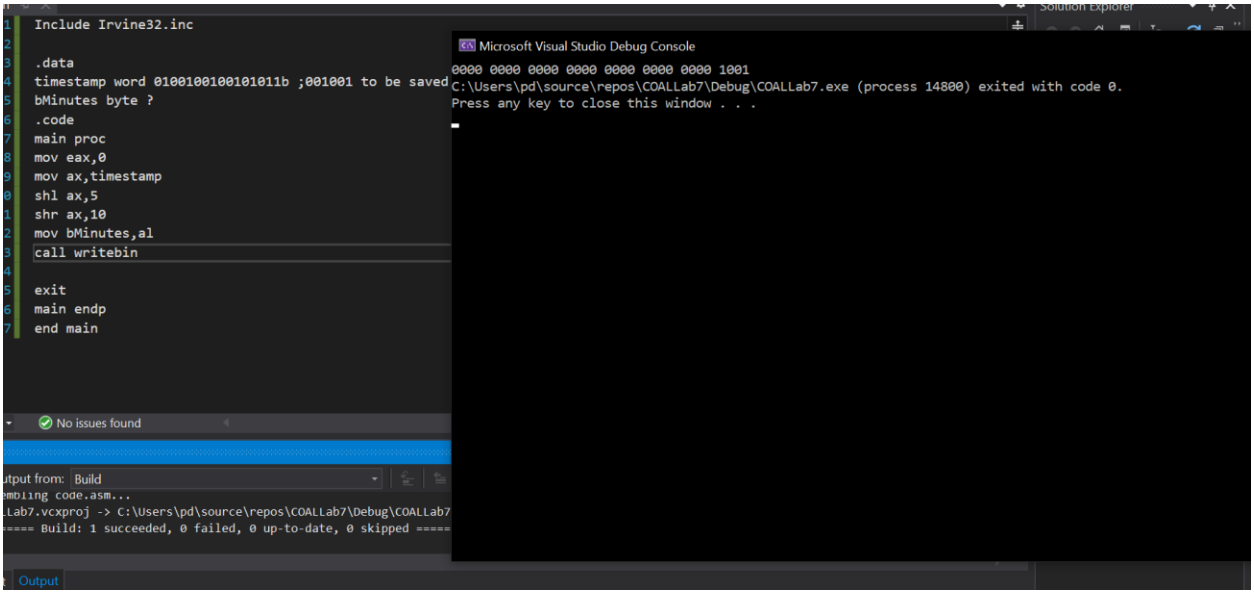
TASK 1:


Code.asm
INCLUDE Irvine32.inc
.code
main PROC
mov eax,3
mov ebx,eax
mov ecx,eax
shl eax,4
shl ebx,2
add eax,ebx
add eax,ecx
call writedec
exit
main ENDP
END main
Solution Explorer
Microsoft Visual Studio Debug Console
63
C:\Users\student\source\repos\COALLab9\Debug\COALLab9.exe (process 76620) exited with code 0.
Press any key to close this window . . .
119 %
No issues found
Output
Show output from: Build

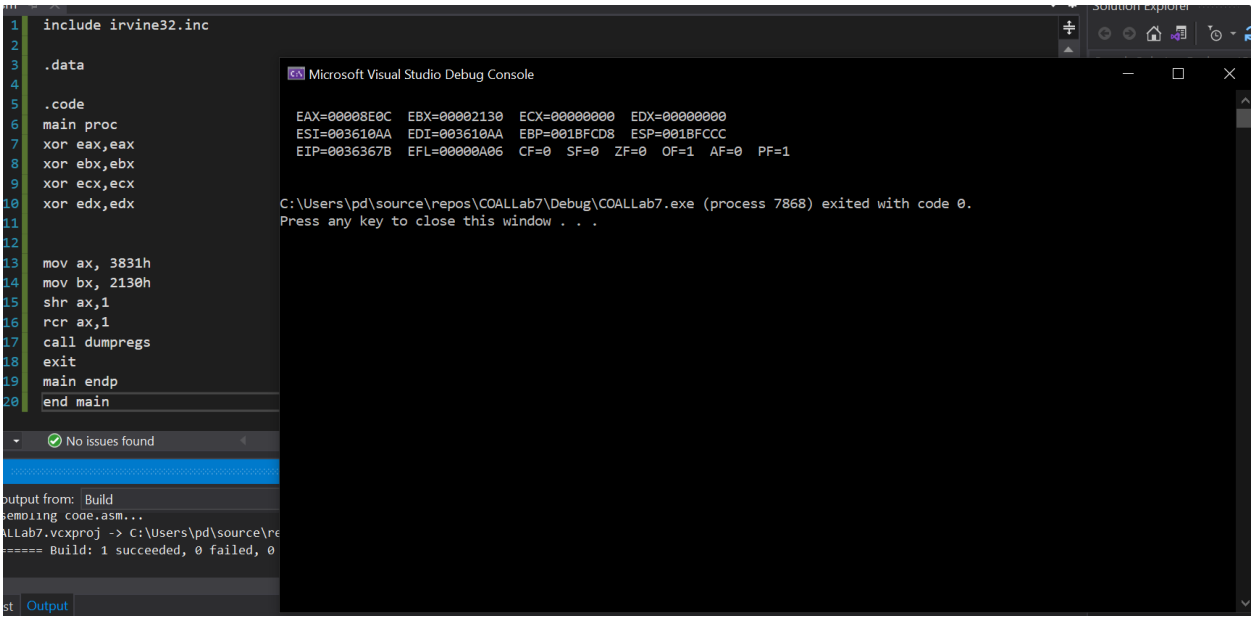
TASK 2:


code.asm
INCLUDE Irvine32.inc
.code
main PROC
mov eax,0
mov ax,-128
rol eax,16
sar eax,16
call writebin
exit
main ENDP
END main
Solution Explorer
Microsoft Visual Studio Debug Console
1111 1111 1111 1111 1111 1111 1000 0000
C:\Users\pd\source\repos\COALLab7\Debug\COALLab7.exe (process 1248) exited with code 0.
Press any key to close this window . . .
107 %
No issues found
Output
Show output from: Build
Error List
Output

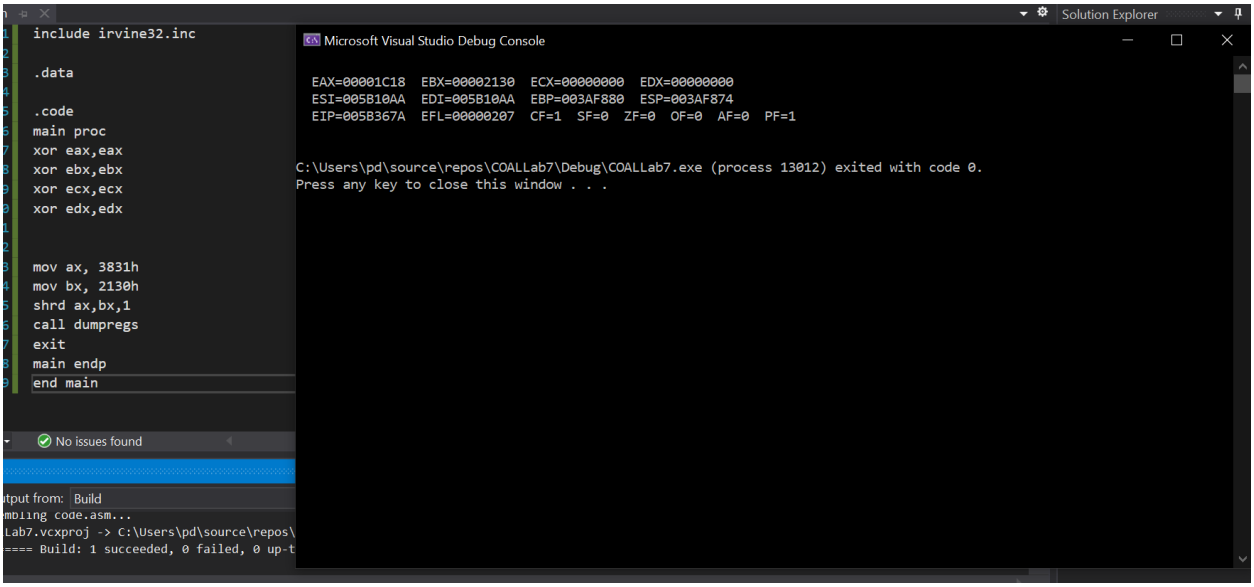
TASK 3:



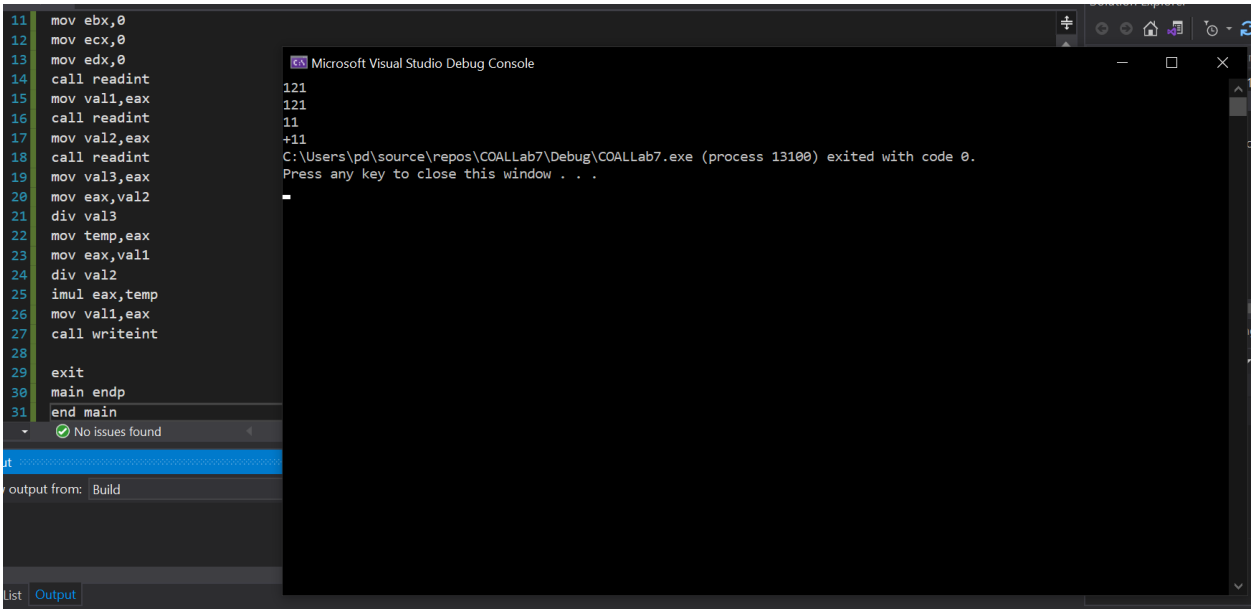
TASK 4 (A):



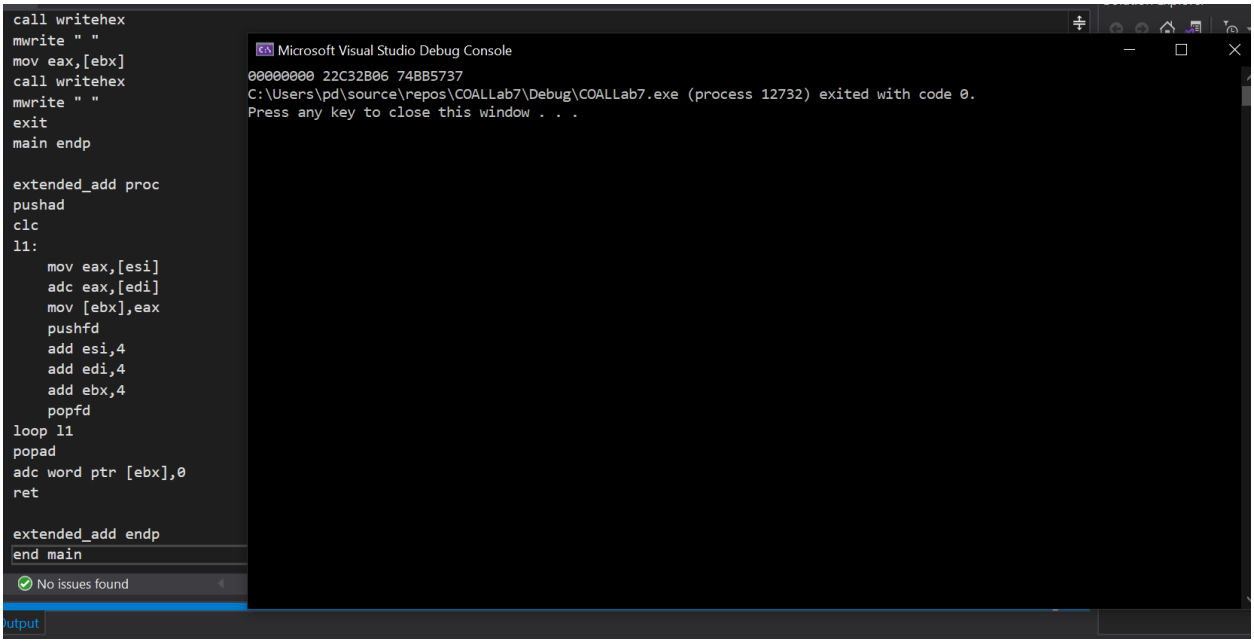
TASK 4(B):



TASK 5:



TASK 6:


call writehex
mwrite " "
mov eax,[ebx]
call writehex
mwrite " "
exit
main endp
extended_add proc
pushad
clc
l1:
mov eax,[esi]
adc eax,[edi]
mov [ebx],eax
pushfd
add esi,4
add edi,4
add ebx,4
popfd
loop l1
popad
adc word ptr [ebx],0
ret
extended_add endp
end main
No issues found
Microsoft Visual Studio Debug Console
00000000 22C32B06 74BB5737
C:\Users\pd\source\repos\COALLab7\Debug\COALLab7.exe (process 12732) exited with code 0.
Press any key to close this window . . .