**LAB 09**

**INTEGER ARITHMETIC**



Mohammad Yehya K213309 G

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_

STUDENT NAME ROLL NO SEC

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SIGNATURE & DATE



EL

2

0

0

3

C

O

M

P

O

R

G

&

A

S

S

E

M

B

L

Y

L

A

N

G

U

A

G

E

L

A

B

L

A

B

O

R

A

T

O

R

Y

M

A

N

U

A

L



F

a

l

l

2

0

2

2



I

n

s

t

r

u

c

t

o

r

:

Ruk

hsa

r

Ali

# MARKS AWARDED: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

**(NUCES), KARACHI**

Version: 1.0

Prepared by: Rukhsar Ali th

Date: 07 Nov 2022

**Lab Session 09: Integer Arithmetic**

## Learning Objectives

1. Shift & rotate Instructions
2. Multiplication and Division
3. Extended Addition and Subtraction

**Shift and Rotate Instructions**

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

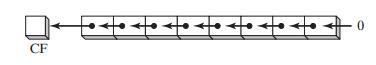
##  Shift Instructions:

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*  *SHL bl,1* | *;BL=10001111b*  *;CF=1, BL=00011110b* |
|  | *mov al,10000000b*  *SHL al,2* | *;AL=10000000b*  *;CF=0, AL=00000000b* |

Page | 1

***Bit Multiplication Example:*** SHL can perform multiplication by powers of 2. Shifting any *n* operand left by *n* bits multiplies the operand by 2 *.* For example, shifting the integer 5 left by

1 bit yields the product of 

*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.

|  |  |  |
| --- | --- | --- |
|  |  | |
| *mov al,0D0h* *shr al,1* | *; AL = 11010000b*    *; AL = 01101000b, CF = 0* |
|  | *mov al,00000010b* *shr al,2* | *; AL = 00000000b, CF = 1* |

Examples:

**Bitwise Division**



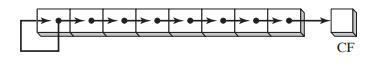
*mov dl,32* *;DL=00100000b* *=32*

*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:

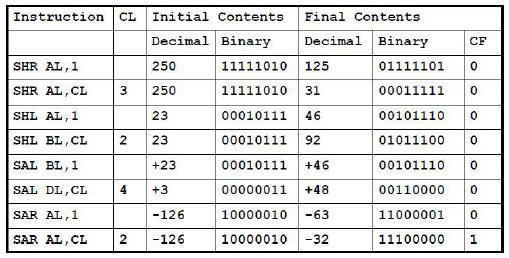


The following example shows how SAR duplicates the sign bit. AL is negative before and after it is shifted to the right:

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

**Sign division:**

|  |  |
| --- | --- |
| mov dl,-128 sar dl,3 | ; DL = 10000000b    ; DL = 11110000b |
| **Sign-Extend AX into EAX:**    mov ax,-128 shl eax,16 sar eax,16 | ; EAX = ????FF80h    ; EAX = FF800000h    ; EAX = FFFFFF80h |

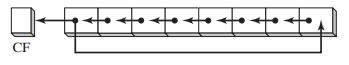


##  Rotate Instructions:

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

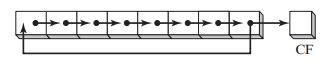
rol al,1 ; AL = 00000010b, CF = 0



|  |  |  |
| --- | --- | --- |
|  | 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.

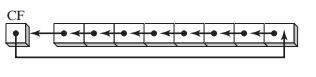


Example:

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

##  RCL Instructions

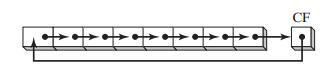
The RCL (rotate carry left) instruction shifts each bit to the left, copies the Carry flag to the LSB, and copies the MSB into the Carry flag:



## Example:

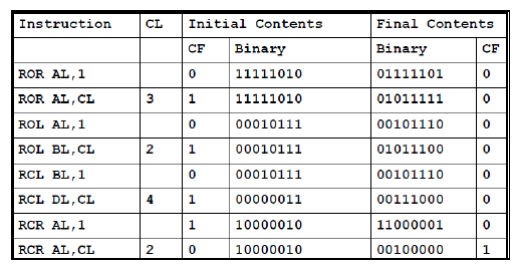
|  |  |  |
| --- | --- | --- |
|  | clc mov bl,88h  rcl bl,1 | ;CF=0    ; CF, BL = 0 10001000b    ; CF,BL = 1 00010000b |

##  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

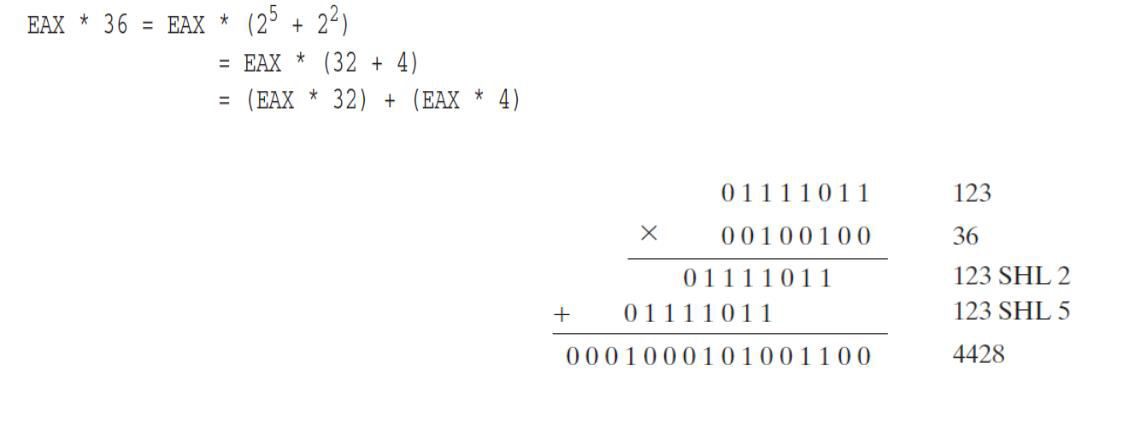
Example:

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



### APPLICATIONS:

add eax,ebx ; add the products



**1.**

**Binary Multiplication**

.code

mov eax,123

mov ebx,eax

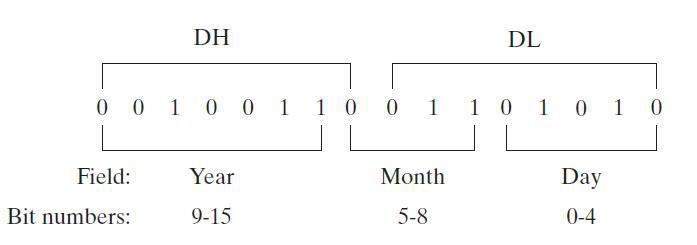
shl eax,5

; mult by 25

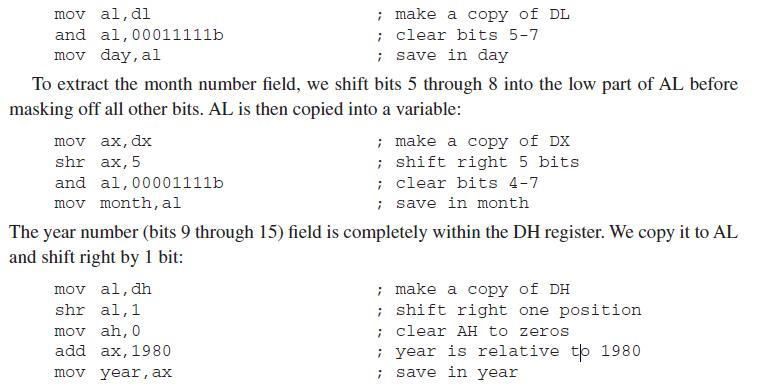
shl ebx,2

; mult by 22

**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:



##  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

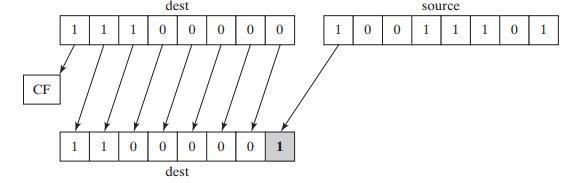
operand.

Format:

SHLD reg16, reg16, CL/imm8

SHLD mem16, reg16, CL/imm8

SHLD reg32, reg32, CL/imm8

SHLD mem32, reg32, CL/imm8

Page | 6

Example:

.data

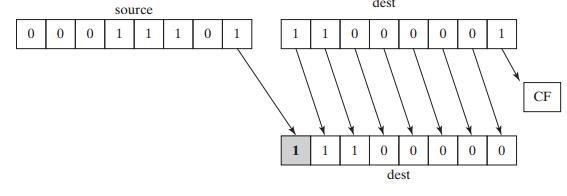
a WORD 9BA6h

|  |  |  |
| --- | --- | --- |
| .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

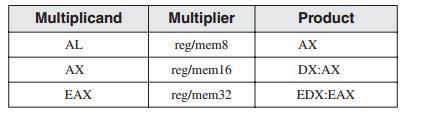
The **MUL** instruction is for unsigned multiplication. Operands are treated as unsigned numbers. The three formats accept register and memory operands, but not immediate operands. The Carry flag is clear (CF = 0) because AH (the upper half of the product) equals zero. Syntax:

MUL reg/mem8

MUL reg/mem16

MUL reg/mem32

 The table represents MUL operands



Page | 7

## 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  IMUL reg/mem16  IMUL reg/mem32 | ; AX = AL \* reg/mem8    ; DX:AX = AX \* reg/mem16    ; EDX:EAX = EAX \* reg/mem32 |

Page | 8

* **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  ;BX=2 |

mov bx,2

Page | 9

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

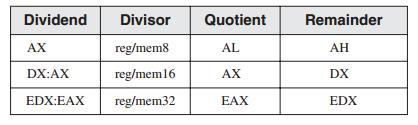
DIV reg/mem8

DIV reg/mem16

Page | 10

DIV reg/mem32

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



Example:

|  |  |  |
| --- | --- | --- |
|  | | *; dividend*  *; divisor*  *; AL = 41h, AH = 01h* |
|  | *mov ax,0083h* *mov bl,2* *div bl* |
|  | *mov dx,0* *mov ax,8003h* *mov cx,100h* *div cx* | *; clear dividend, high*  *; dividend, low*  *; divisor*  *; 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 cwd | ; AX = FF9Bh  ; DX:AX = FFFFFF9Bh |

Page | 11

**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 = FFFFFFFFFFFFFF9Bh |

##  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 cbw mov bl,+5 idiv bl | ; lower half of dividend  ; extend AL into AH  ; divisor  ;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*

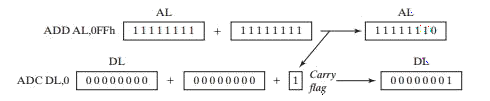
**EXAMPLE # 01:**

mov dl,0 mov al,0FFh

add al,0FFh ; AL = FEh

adc dl,0 ; DL/AL = 01FEh

Page | 12



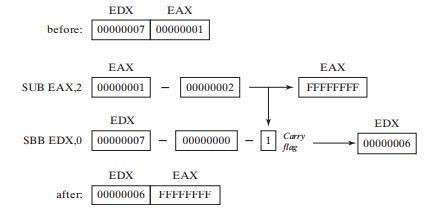
##  SBB Instructions:

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* *mov eax,1* *sub eax,2* *sbb edx,0* | *; upper half*  *; lower half*  *; subtract 2*  *; subtract upper half* |



Page | 13

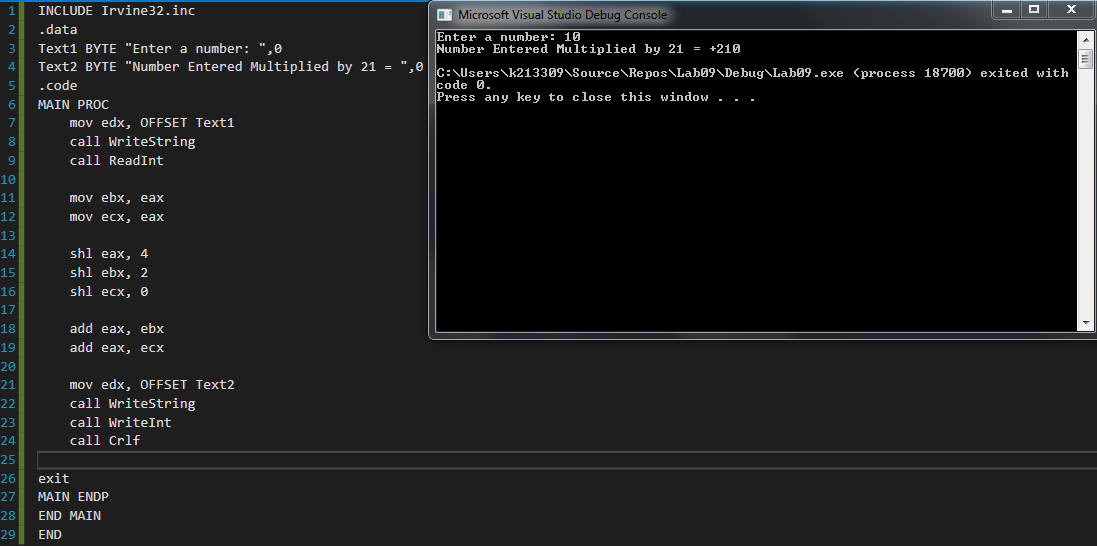
**ACTIVITY:**

## Task#1

Write ASM instructions that calculate EAX \* 21 using binary multiplication.

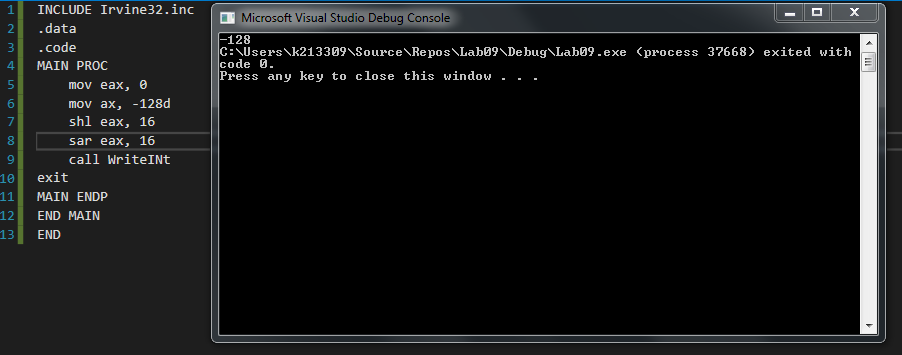
4 2 0

Hint: 21 = 2 + 2 + 2 .



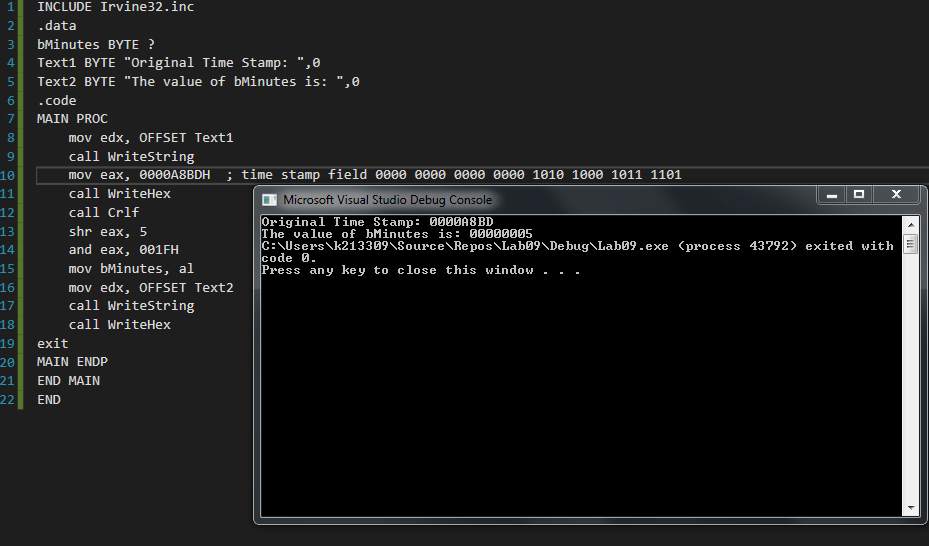
## Task#2

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



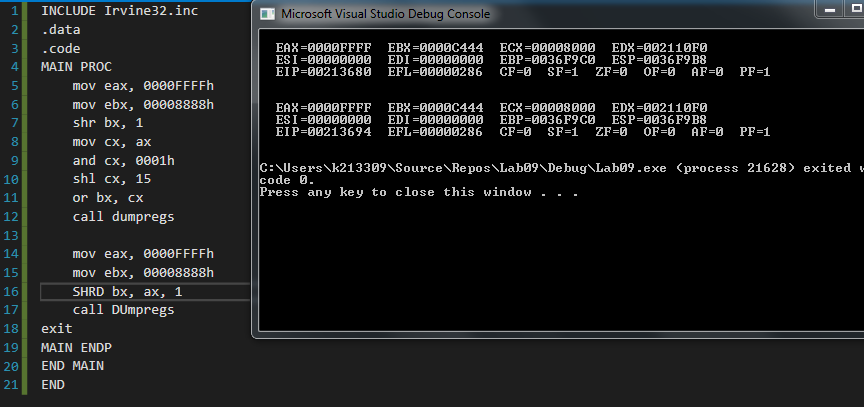
## Task#3

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

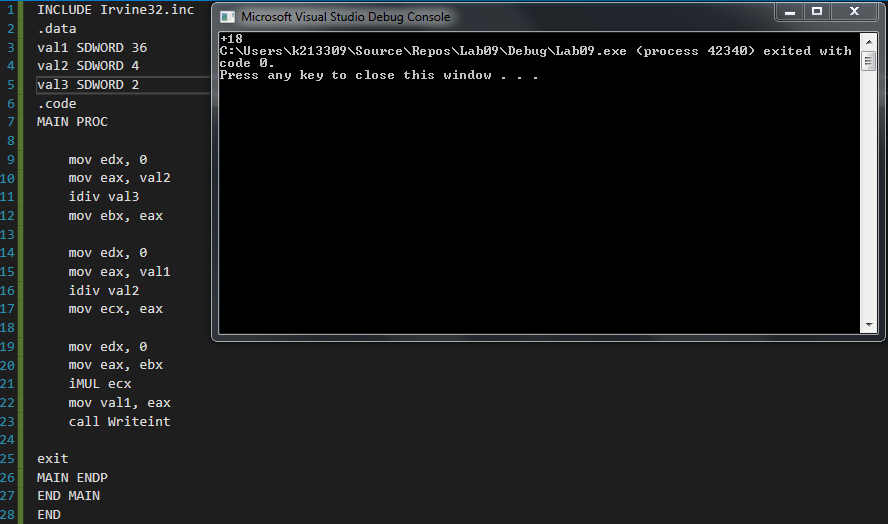
Write a series of instructions that shift the lowest bit of AX into the highest bit of BX without using the SHRD instruction. Next, perform the same operation using SHRD.



## Task#5

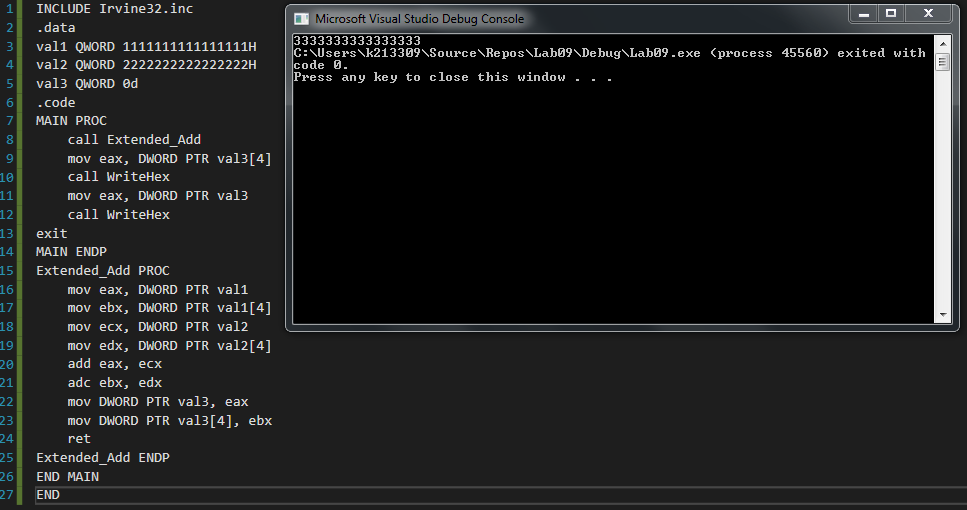
Implement the following C++ expression in assembly language, using 32-bit signed operands:

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



## Task#6

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



Page | 13