Faculty of Engineering
Computer Engineering Department
Islamic University of Gaza

Assembly Language Lab # 9
Shift,Rotate,Multiplication and Division Instruction

Eng. Alaa.I.Haniya

Assembly Language Fundamentals

Objective:

To know more about Assembly language, such Shift, Rotate, Multiplication and Division Instructions.

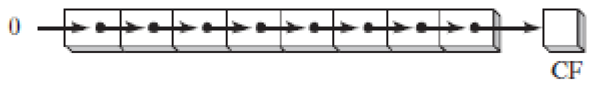
Shift and Rotate Instructions

- Shifting means to move bits right and left inside an operand.
- The following table provides Shift and Rotate Instructions.
- All affecting the Overflow and Carry flags.

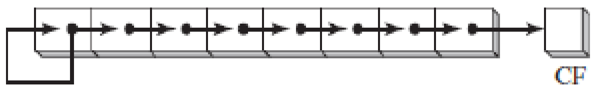
SHL	Shift left
SHR	Shift right
SAL	Shift arithmetic left
SAR	Shift arithmetic right
ROL	Rotate left
ROR	Rotate right
RCL	Rotate carry left
RCR	Rotate carry right
SHLD	Double-precision shift left
SHRD	Double-precision shift right

❖ *Logical Shifts and Arithmetic Shifts*

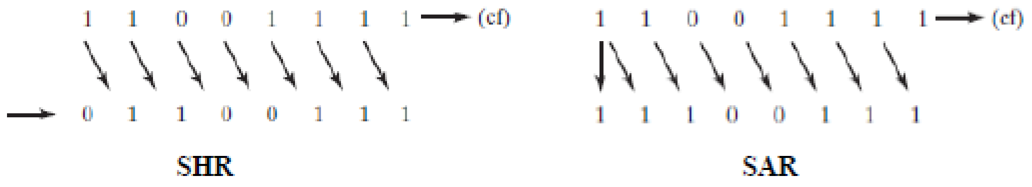
- A logical shift fills the newly created bit position with zero.



An arithmetic shift fills the newly created bit position with a copy of the number’s sign bit.

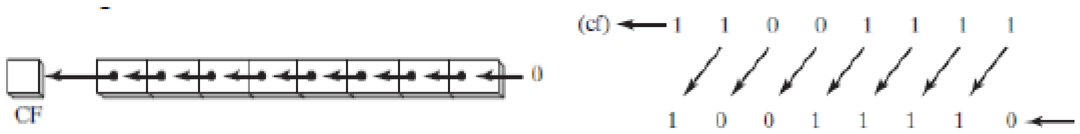


Example of Right Logical Shifts and Right Arithmetic Shifts



SHL Instruction

-The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.



The first operand in SHL is the destination and the second is the shift count:

```
SHL destination,count
```

- Operand types for SHL:

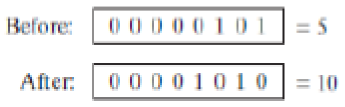
```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

- Formats shown here also apply to the SHR, SAL, SAR, ROR, ROL, RCR, and RCL instructions.

Application: Fast Multiplication

- Shifting left 1 bit multiplies a number by 2
- Shifting the integer 5 left by 1 bit yields the product of 5 * 2^1=10

```
mov dl,5
shl dl,1
```

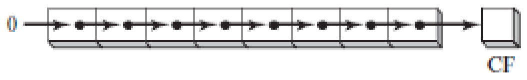


Shifting left *n* bits multiplies the operand by2^n
For example, 5 * 22 = 20

```
mov dl,5
shl dl,2 ;DL = 20, CF = 0
```

SHR Instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.



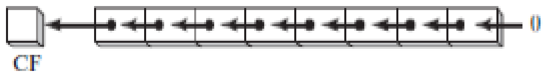
Application: Division

- Shifting right n bits divides the operand by 2^n

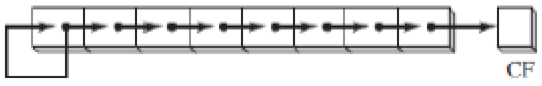
```
mov dl,80      ; DL = 01010000b
shr dl,1       ; DL = 00101000b = 40, CF = 0
shr dl,2       ; DL = 00001010b = 10, CF = 0
```

❖ SAL and SAR Instructions

- SAL (shift arithmetic left) is identical to SHL.



- SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.



❖ *Applications:*

1. Signed Division

An arithmetic shift preserves the number's sign.

```
mov dl,-80      ; DL = 10110000b
sar dl,1        ; DL = 11011000b = -40, CF = 0
sar dl,2        ; DL = 11110110b = -10, CF = 0
```

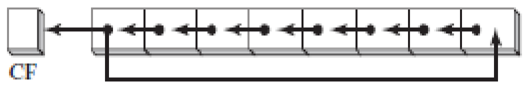
2. Sign-Extend

Suppose AX contains a signed integer and you want to extend its sign into EAX. First shift EAX 16 bits to the left, then shift it arithmetically 16 bits to the right:

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

❖ *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.
- No bits are lost.



Example:

```
mov al,11110000b
rol al,1                    ;AL = 11100001b, CF = 1
```

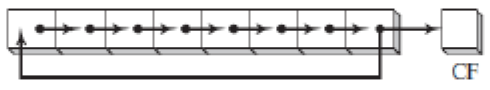
Application: Exchanging Groups of Bits

You can use ROL to exchange the upper (bits 4–7) and lower (bits0–3) halves of a byte.

```
mov dl,3Fh             ; DL = 00111111b
rol dl,4               ; DL = 11110011b = F3h, CF = 1
```

❖ *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.
- No bits are lost.



✓ **Example:**

```
mov al,11110000b
ror al,1                ; AL = 01111000b, CF = 0
```

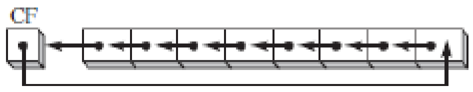
✓ **Application: Exchanging Groups of Bits**

- You can use ROL to exchange the upper (bits 4–7) and lower (bits0–3) halves of a byte.

```
mov dl,3Fh              ; DL = 00111111b
ror dl,4                ; DL = 11110011b = F3h, CF = 1
```

RCL Instruction

- 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.

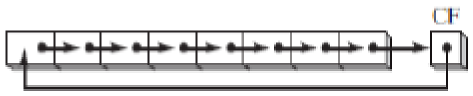


Example:

```
Clc                 ; clear carry, CF = 0
mov bl,88h          ;CF = 0 ,BL = 10001000b
rcl bl,1            ;CF = 1 ,BL = 00010000b
rcl bl,1            ;CF = 0 ,BL = 00100001b
```

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             ; set carry, CF = 1
mov ah,10h      ;CF = 1, AH = 00010000b
rcr ah,1        ; CF = 0, AH = 10001000b 5
```

SHLD Instruction

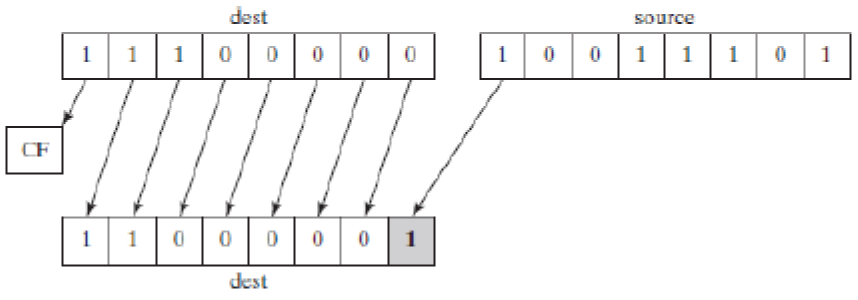
- The SHLD (shift left double) instruction shifts a destination operand a given number of bits to the left.
- The bit positions opened up by the shift are filled by the most significant bits of the source operand.
- Only the destination is modified, not the source.

Syntax:

```
SHLD dest, source, count
```

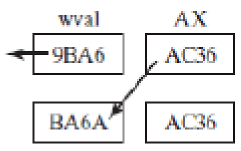
Operand types:

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



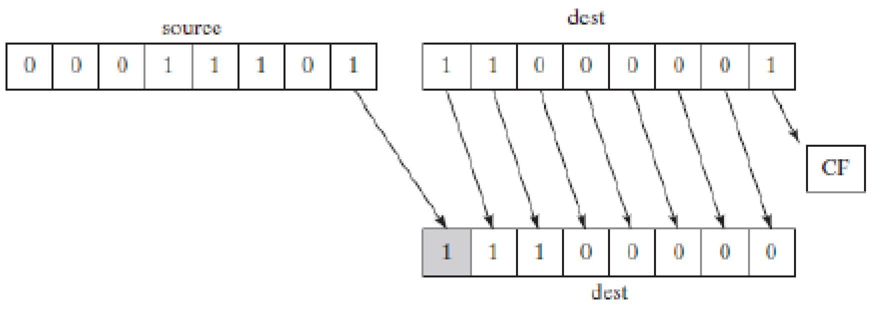
✓ **Example:**

```
.data
wval WORD 9BA6h
.code
mov ax,0AC36h
shld wval,ax,4
```



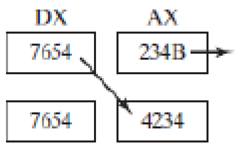
SHRD Instruction

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



Example:

```
mov ax,234Bh
mov dx,7654h
shrd ax,dx,4
```



Multiplication and Division Instructions

MUL Instruction

- The MUL (unsigned multiply) instruction comes in three versions:

  - ✓ The first version multiplies an 8-bit operand by the AL register.
  - ✓ The second version multiplies a 16-bit operand by the AX register.
  - ✓ The third version multiplies a 32-bit operand by the EAX register.

- The multiplier and multiplicand must always be the same size, and the product is twice their size.
- The three formats accept register and memory operands, but not immediate operands:

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

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

- MUL sets the Carry and Overflow flags if the upper half of the product is not equal to zero.

✓ **Example1: Multiply 16-bit var1 (2000h) * var2 (100h)**

```
.data
var1 WORD 2000h
var2 WORD 100h
.code
mov ax,var1
mul var2        ; DX:AX = 00200000h, CF = OF = 1
```

✓ **Example2: Multiply EAX (12345h) * EBX (1000h)**

```

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

❖ 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

```
o DIV reg/mem8

o DIV reg/mem16

o DIV reg/mem32
```

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

✓ **Example1: Divide AX = 8003h by CX = 100h, using 16-bit operands**

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

✓ **Example2: Same division, using 32-bit operands**

```
mov edx,0         ;clear dividend, high
mov eax,8003h     ;dividend, low
mov ecx,100h      ;divisor
div ecx ; EAX = 00000080h, EDX = 00000003h
```

Lab work:

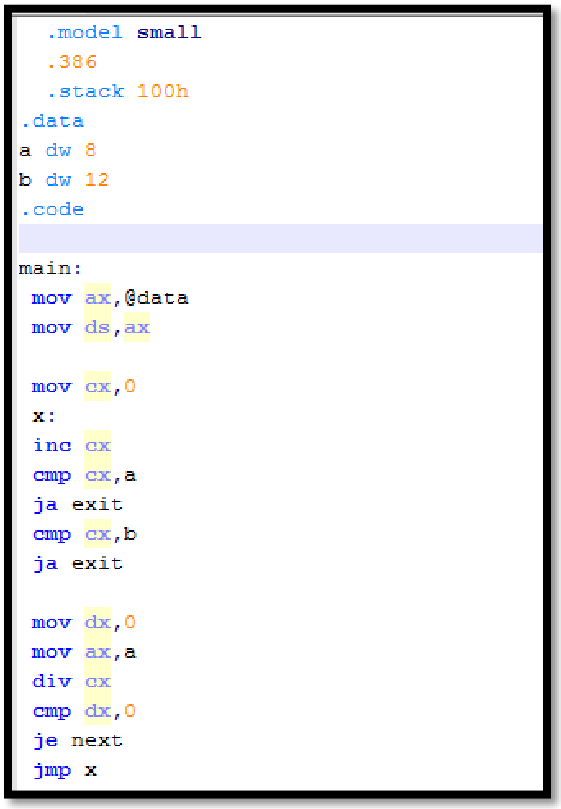
Excercise1:

The greatest common divisor of two integers is the largest integer that will evenly divide (without a remainder) both integers . The GCD algorithm involves integer division in a loop, described by the following c++ code:

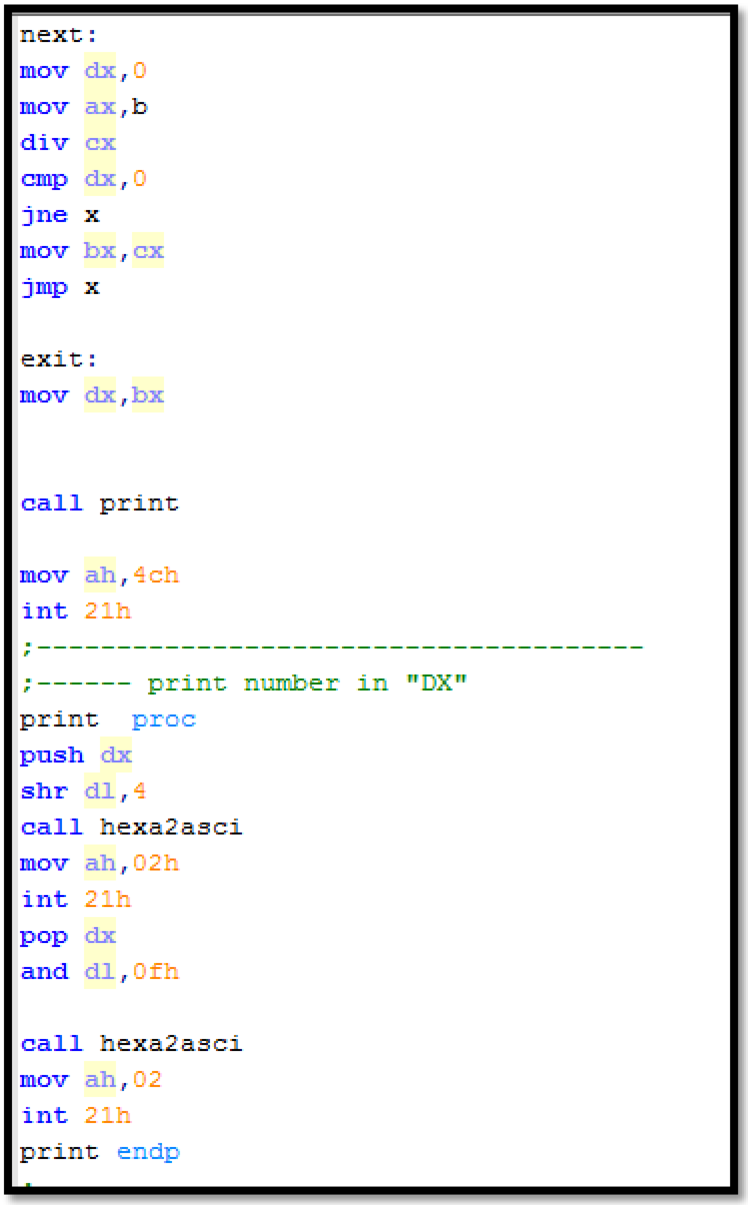
```
int GCD(int x , int y)
{
x = abs (x);
y = abs (y);
do {
int n = x % y;
x = y;
y = n;
} while (y > 0) ;
return x ;
}
```

Write an assembly program that determines the gcd (greatest common divisor) of two positive integer numbers for example (8,12).

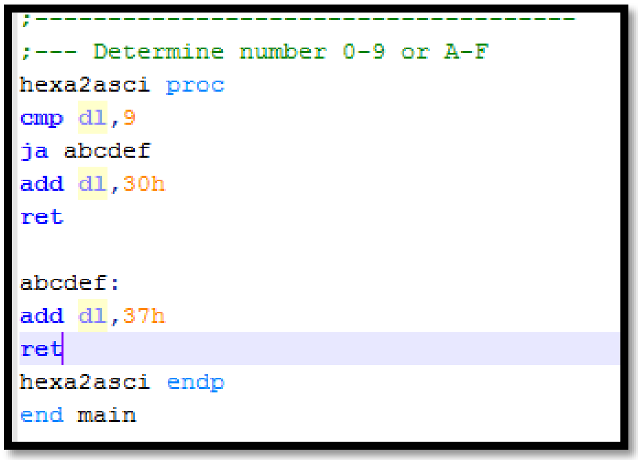
This solution is just for positive integers and don't follow the previous algorithm. In your homework you should implement the algorithm exactly.



```
  .model small
  .386
  .stack 100h
.data
a dw 8
b dw 12
.code

main:
 mov ax,@data
 mov ds,ax

 mov cx,0
 x:
 inc cx
 cmp cx,a
 ja exit
 cmp cx,b
 ja exit

 mov dx,0
 mov ax,a
 div cx
 cmp dx,0
 je next
 jmp x
```

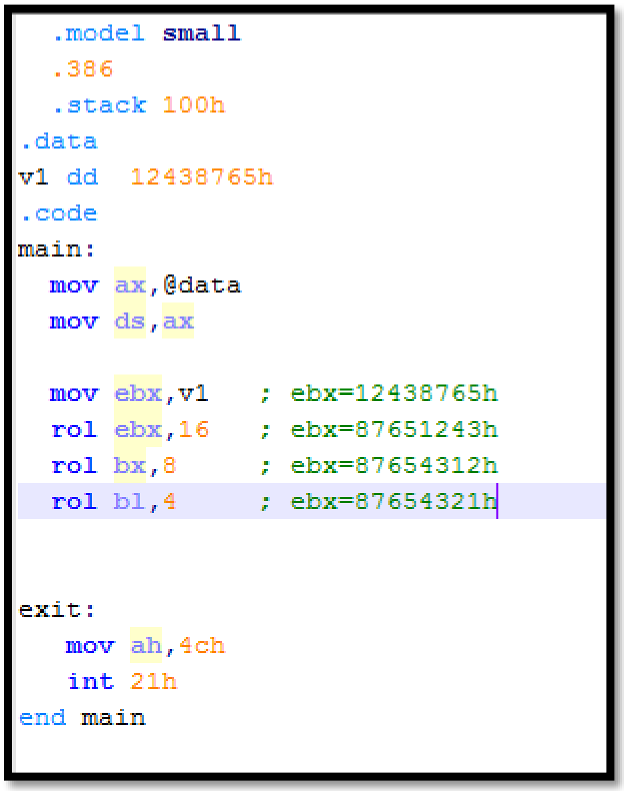


```
next:
mov dx,0
mov ax,b
div cx
cmp dx,0
jne x
mov bx,cx
jmp x

exit:
mov dx,bx


call print

mov ah,4ch
int 21h
;--------------------------------------
;------ print number in "DX"
print  proc
push dx
shr dl,4
call hexa2asci
mov ah,02h
int 21h
pop dx
and dl,0fh

call hexa2asci
mov ah,02
int 21h
print endp
```



```
;--------------------------------------
;--- Determine number 0-9 or A-F
hexa2asci proc
cmp dl,9
ja abcdef
add dl,30h
ret

abcdef:
add dl,37h
ret
hexa2asci endp
end main
```

Excercise2:
Write an assembly code to convert the number 12438765h to 87654321h.



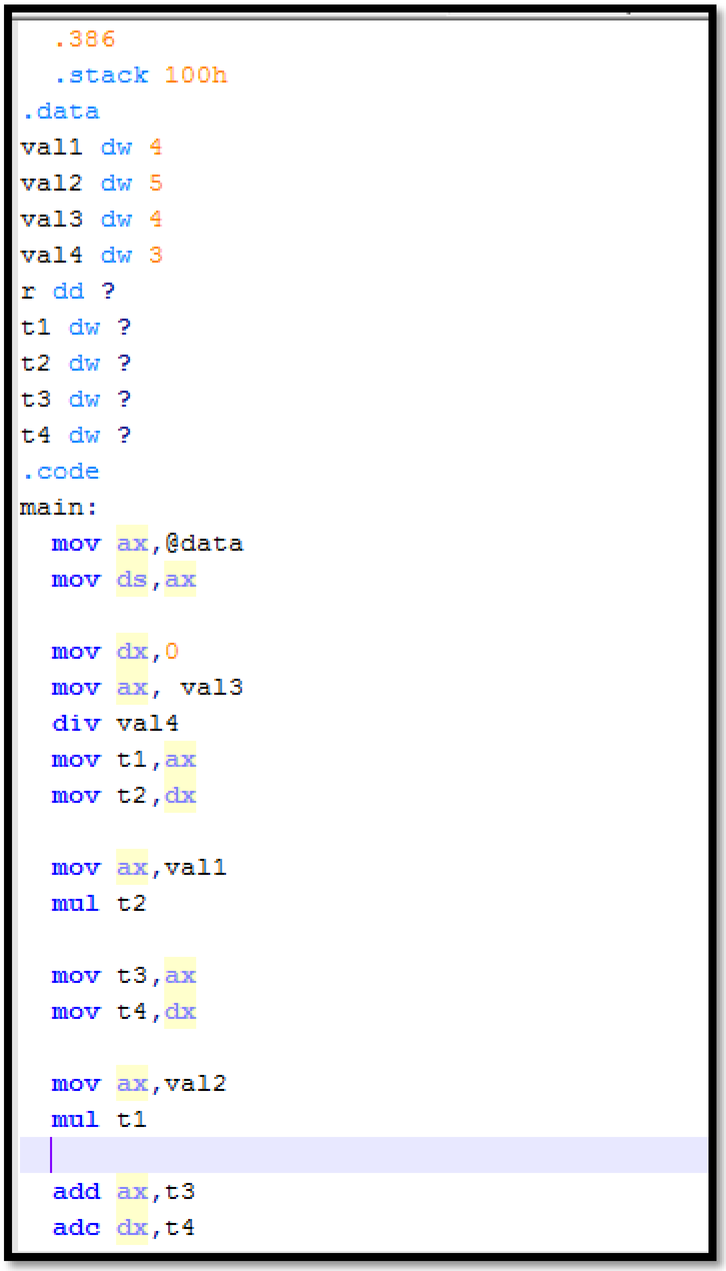
```
  .model small
  .386
  .stack 100h
.data
v1 dd  12438765h
.code
main:
  mov ax,@data
  mov ds,ax

  mov ebx,v1    ; ebx=12438765h
  rol ebx,16    ; ebx=87651243h
  rol bx,8      ; ebx=87654312h
  rol bl,4      ; ebx=87654321h


exit:
   mov ah,4ch
   int 21h
end main
```

Excercise3:
Write an assembly code to evaluate the following expression:

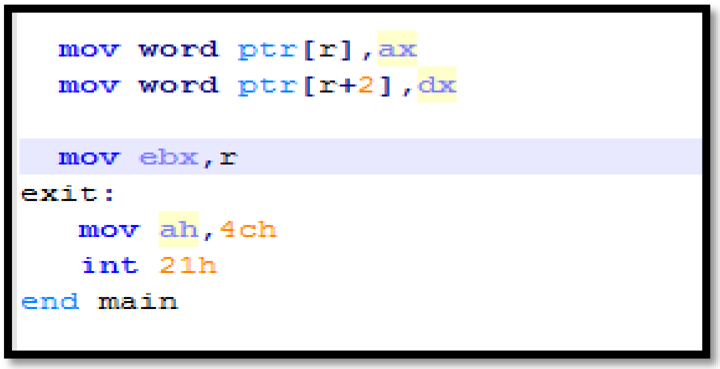
(val1*(val3/val4)) + (val2*(val3%val4))

Use these values:

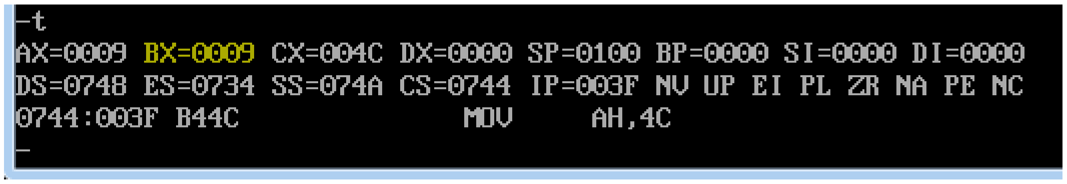
val1 dw 4
val2 dw 5
val3 dw 4
val4 dw 3
result dd ?



```
  .386
  .stack 100h
.data
val1 dw 4
val2 dw 5
val3 dw 4
val4 dw 3
r dd ?
t1 dw ?
t2 dw ?
t3 dw ?
t4 dw ?
.code
main:
  mov ax,@data
  mov ds,ax

  mov dx,0
  mov ax, val3
  div val4
  mov t1,ax
  mov t2,dx

  mov ax,val1
  mul t2

  mov t3,ax
  mov t4,dx

  mov ax,val2
  mul t1

  add ax,t3
  adc dx,t4
```



```
  mov word ptr[r],ax
  mov word ptr[r+2],dx

  mov ebx,r
exit:
   mov ah,4ch
   int 21h
end main
```



```
-t
AX=0009 BX=0009 CX=004C DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=0748 ES=0734 SS=074A CS=0744 IP=003F NV UP EI PL ZR NA PE NC
0744:003F B44C          MOV     AH,4C
-
```

Homework:

1. Write an assembly code to convert a binary string into hexadecimal value. If the binary String is greater than 32 digits length a value zero must be returned. Use this value declaration:

```
B_Val db '10001111' , '$'
```

2. Write an assembly code to find the power of any integer to any integer using **mul** instruction. Use these values for testing:

```
base db 3h
power db 5h
result dd ? ;3^5 =243 = F3H
```

3. Implement the following GCD algorithm:

```
int GCD(int x , int y)
{
x = abs (x);
y = abs (y);
do {
int n = x % y;
x = y;
y = n;
} while (y > 0) ;
return x ;
}
```

Find: GCD(-8,12)