Coal Assignment: 03

# **Question 01:**

**TAKS-01:**

The product is stored in registers that are twice the size of the multiplier and multiplicand. If you multiply 0FFh by 0FFh, for example, the product (FE01h) easily fits within 16 bits.

**TAKS-02:**

When the product fits completely within the lower register of the product, IMUL sign extends the product into the upper product register. MUL, on the other hand, zero-extends the product.

**TAKS-03:**

With IMUL, the Carry and Overflow flags are set when the upper half of the product is not a sign extension of the lower half of the product.

**TAKS-04:**

EAX register holds the quotient.

**TAKS-05:**

AX holds the product.

**TAKS-06:**

## For Example:

mov ax, dividendLow

cwd; sign - extend dividend

mov bx, divisor

idiv bx

**TAKS-07:**

EDX = 0, EAX = 00012340h.

**TAKS-08:**

The DIV will cause a divide overflow, so the values of AX and DX cannot be determined.

**Question 02:**

mov ecx, 8; loop counter

mov esi, 0; use the same index reg

clc; clear Carry flag

top :

mov al, byte ptr val1[esi]; get first number

sbb al, byte ptr val2[esi]; subtract second

mov byte ptr result[esi], al; store the result

inc esi; move to next pair

loop top

**Question 03:**

## Program:

include irvine32.inc

.data

in1 db "Enter 1st number: ", 0

in2 db "Enter 2nd number: ", 0

out1 db "GCD is: ", 0

isPrime db "Numbers are relative Prime ", 0

notPrime db "Numbers are not relative Prime ", 0

gcd dw ?

temp dw ?

.code

main proc

call DEC\_IN

call GCD\_AB

call DEC\_OUT

exit

main endp

DEC\_IN proc

mov edx, offset in1

call writeString

mov eax, 0

call readDec

mov bx, ax

mov edx, offset in2

call writeString

mov eax, 0

call readDec

mov dx, ax

ret

DEC\_IN endp

GCD\_AB proc

mov gcd, bx

cmp bx, dx

ja next

je endd

xchg bx, dx

next :

mov temp, dx

mov dx, 0

mov ax, bx

again :

div temp

cmp dx, 0

je end1

mov ax, temp

mov temp, dx

mov dx, 0

jmp again

end1 :

mov ax, temp

mov gcd, ax

endd :

ret

GCD\_AB endp

DEC\_OUT proc

call crlf

mov edx, offset out1

call writeString

mov eax, 0

mov ax, gcd

call writeDec

call crlf

cmp gcd, 1

jne next

mov edx, offset isPrime

call writeString

jmp next1

next :

mov edx, offset notPrime

call writeString

next1 :

ret

DEC\_OUT endp

end main

**Output:**

