# Question 1.

## Task 1.

Product is stored in registers that may be the twice the size of the multiplier and multiplicand. The answer after multiply will be in 16 bits.

## Task 2.

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.

## Task 3.

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.

## Task 4.

eax register holds the quotient

## Task 5.

ax holds the product.

## Task 6.

mov ax, dividendL

cwd; sign - extend dividend

mov bx, divisor

idiv bx

## Task 7.

edx=0, eax=00012350h

## Task 8.

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

# Question 2.

mov ecx, 8;

mov esi, 0;

clc;

top :

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

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

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

inc esi;

loop top; loop end here

# Question 3.

Include irvine32.inc

.data

n1 db "Enter first number:: ", 0

n2 db "Enter second number:: ", 0

op 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 n1

call writeString

mov eax, 0

call readDec

mov bx, ax

mov edx, offset n2

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 op

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