Question 1

Part-1:

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.

Part-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.

Part-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.

Part-4:

EAX register holds the quotient.

Part-5:

AX holds the product.

Part-6:

Example:

mov ax,dividendLow

cwd ; sign-extend dividend

mov bx,divisor

idiv bx

Part-7:

EDX = 0, EAX = 00012340h.

Part-8:

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

Question 2

In correcting this example, it is easiest to reduce the number of instructions. You can use a single register (ESI) to index into all three variables. ESI should be set to zero before the loop because the integers are stored in little endian order with their low-order bytes occurring first:

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

Of course, you could easily reduce the number of loop iterations by adding doublewords rather than

bytes.

Question 3

INCLUDE Irvine32.inc

includelib irvine32.lib

.data

GCD DWord ?

msg1 DB "Enter 1st Number: ", 0

msg2 DB "Enter 2nd Number: ", 0

gcdmsg DB "GCD is: ", 0

Prime DB "Numbers are relative prime", 0

NotPrime DB "Numbers are not relative prime", 0

msg6 BYTE "try again" , 0

.code

DEC\_IN proc

up1:

call crlf

mov eax , 0

mov edx , offset msg1

call writestring

call readdec

cmp eax , 1

jb down1

cmp eax , 99

ja down1

mov bx , ax

up2:

call crlf

mov eax , 0

mov edx , offset msg2

call writestring

call readdec

cmp eax , 1

jb down2

cmp eax , 99

ja down2

mov dx , ax

jmp done1

down1:

mov edx , offset msg6

call writestring

jmp up1

down2:

mov edx , offset msg6

call writestring

jmp up2

done1:

ret

DEC\_IN endp

Check PROC

cmp bx, dx

JGE Greater

xchg bx, dx

Greater:

ret

Check endp

GCD\_AB PROC

mov ax, bx

mov cx, dx

mov dx, 0

div cx

add dx, 0

JNZ next

movzx edx, cx

mov GCD, edx

ret

next:

mov bx, cx

call Check

call GCD\_AB

ret

GCD\_AB endp

DEC\_OUT PROC

mov ebx, gcd

mov edx, offset gcdmsg

call WriteString

mov eax, gcd

call Writedec

call crlf

cmp ebx, 1

JZ Relative

mov edx, offset NotPrime

call WriteString

JMP ExitProgram

Relative:

mov edx, offset Prime

call WriteString

ExitProgram:

ret

DEC\_OUT endp

main PROC

call DEC\_IN

cmp ebx, edx

JNZ NotEqual

mov GCD, ebx

JMP EndA

NotEqual:

cmp ebx, 0

JNZ ebxNotZero

mov GCD, edx

JMP EndA

ebxNotZero:

cmp edx, 0

JNZ edxNotZero

mov GCD, ebx

JMP EndA

edxNotZero:

call GCD\_AB

EndA:

call DEC\_OUT

call Readchar

invoke ExitProcess, 0

main endp

end main

Question 4

INCLUDE Irvine32.inc

includelib irvine32.lib

.data

str1 db ' a b c ','$'

len equ $-str1

cntr dw ?

.code

mov ax,@data

mov ds,ax

mov ah,09h

LEA DX,str1

int 21h

LEA DI,str1

mov BX,0000

mov SI,0000

mov cntr,len

sub cntr,1

L1:

mov ah,str1[SI]

cmp ah,' '

je remove

INC SI

cmp SI,cntr

jge ext

jmp L1

remove:

mov BX,SI

R1:

mov ah,str1[BX+1]

mov str1[BX],ah

cmp BX,len

jge ChangeLen

add BX,1

jmp R1

ChangeLen:

mov str1[BX],'$'

sub cntr,1

jmp L1

ext:

mov ah,09h

LEA DX,str1

int 21h

mov ax,4C00h

int 21h

end