18F-0358 Mehdi Raza *COAL*

Assignment# 2

Question#1

Product is ordered in registers which are twice the size of multiple and multiplicities. If multiply 0ffh by 0ffh

Question#1ii

The product fits completely within lower register of product.

IMUL

Extent the product into the upper register.

MUL

Zero extend register.

Question#1iii

With IMUL carry and overflow flags are set when upper half of the product is not sign extension of lower half of register in product.

Question#1iv

EAX register hold lower quotient.

Question#1v

AX register hold the product.

Question#1vi

Mov ax, lowdiv

Cwd

Mov bx, diviset

Div bx

Question#1vii

Edx=0

Eax=00012340h

Question#2

Correcting 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

Question#5

INCLUDE Irvine32.inc

.data

Op1 byte 34h,12h,98h,74h,06h

Byte 02h,45h,23h,00h,00h

Byte 23h,55h,92h,41h,82h

Row\_index=2

Sum=6A

strR1 BYTE "Enter Number Of Rows : ",0

strC1 BYTE "Enter Numbeer Of Columns : ",0

matx BYTE "First : ",0

matx1 BYTE "Second : ", 0

matx2 BYTE "Final : ", 0

row1 BYTE ?

col1 BYTE ?

arr1 BYTE 30 dup(? )

arr2 BYTE 30 dup(? )

arr3 BYTE 30 dup(?)

spac BYTE " ", 0

temp BYTE ?

row2 BYTE ?

col2 BYTE ?

x1 BYTE ?

sum BYTE ?

.code

main proc

mov edx, OFFSET strR1

call writestring

call readdec

xor ecx,ecx

mov row1, al

mov edx,OFFSET strC1

call writestring

call readdec

mov col1, al

mov cl, row1

mov esi, OFFSET arr1

l1 :

push ecx

mov cl, col1

l2 :

call readdec

mov[esi], al

inc esi

loop l2

pop ecx

loop l1

mov cl, row1

mov esi, OFFSET arr1

mov edx, OFFSET matx

call writestring

call crlf

l3 :

push ecx

mov cl, col1

l4 :

mov al, [esi]

call writedec

mov edx, OFFSET spac

call writestring

inc esi

loop l4

call crlf

pop ecx

loop l3

mov edx, OFFSET strR1

call writestring

call readdec

xor ecx, ecx

mov row2, al

mov edx, OFFSET strR1

call writestring

call readdec

mov col2, al

mov cl, row2

mov esi, OFFSET arr2

l5 :

push ecx

mov cl, col2

l6 :

call readdec

mov[esi], al

inc esi

loop l6

pop ecx

loop l5

mov cl, row2

mov esi, OFFSET arr2

mov edx, OFFSET matx1

call writestring

call crlf

l7 :

push ecx

mov cl, col2

l8 :

mov al, [esi]

call writedec

mov edx, OFFSET spac

call writestring

inc esi

loop l8

call crlf

pop ecx

loop l7

mov esi, offset arr1

mov edx, offset arr3

mov cl, row1

l9:

push ecx

mov x1,0

movzx ecx,col2

l10:

mov sum,0

mov ebx,OFFSET arr2

mov al,x1

inc x1

add ebx,eax

push ecx

movzx ecx,row1

push esi

l11:

push edx

mov dl, [ebx]

mov al, [esi]

mul dl

pop edx

add sum, al

inc esi

movzx eax, row2

add ebx, eax

loop l11

mov al, sum

mov[edx], al

inc edx

pop esi

pop ecx

loop l10

pop ecx

movzx eax, col1

add esi, eax

loop l9

mov edx, OFFSET matx2

call writestring

xor ecx, ecx

mov cl, row2

mov esi, OFFSET arr3

call crlf

l12 :

push ecx

mov cl, col2

l13 :

mov al, [esi]

call writedec

mov edx, OFFSET spac

call writestring

inc esi

loop l13

call crlf

pop ecx

loop l12

exit

main endp

end main