

TRABAJO AUTÓNOMO 3 - ASSEMBLY

1.- Escribe en el espacio correspondiente el contenido final de AX, BX, CX, DX y DS después de ejecutar todas las instrucciones. Debes justificar el proceso de resolución paso a paso. Puedes ayudarte con una tabla u hoja de cálculo que muestre cómo cambian los registros cada vez que se ejecuta una instrucción.

mov ax, 10001b	mov cx, bx
mov bx, 19h AX	mov dx, ax
mov cx, 16h	mov ds, dx
mov dx, 21	add ax, 55
mov ds, ax BX	sub ax, 10h
sub bx, 12	
add ax, bx	
div bl CX	
add cx, bx	
add ax, dx	
sub ax, bx DX DX	
mov bx, ds	
mov cx, bx	
mov dx, ax DS	
add ax, 55	
sub ax, 10h	
mov ax, 11101b	
mov bx, 16h	
mov cx, 19h	
mov dx, 21	
mov ds, ax	
sub bx, 12	
add ax, bx	
div bl	
add cx, bx	
add ax, dx	
sub ax, bx	
mov bx, ds	

La tabla es:

Instrucción	AX	BX	CX	DX	DS
mov ax, 1001b	00010001	0	0	0	0
mov bx, 19h	00010001	00011001	0	0	0
mov cx, 16h	00010001	00011001	00010110	0	0
mov dx, 21	00010001	00011001	00010110	00010101	0
mov ds, ax	00010001	00011001	00010110	00010101	00010001
sub bx, 12	00010001	00000111	00010110	00010101	00010001
add ax, bx	00010110	00000111	00010110	00010101	00010001
div bl	00000000	00000111	00010110	00010101	00010001
add cx, bx	00010110	00000111	00011101	00010101	00010001
add ax, dx	00011011	00000111	00011101	00010101	00010001
sub ax, bx	00010100	00000111	00011101	00010101	00010001
mov bx, ds	00010100	00010001	00011101	00010101	00010001
mov cx, bx	00010100	00010001	00010001	00010101	00010001
mov dx, ax	00010100	00010001	00010001	00010101	00010001
mov ds, dx	00010100	00010001	00010001	00010101	00010100
add ax, 55	00100011	00010001	00010001	00010101	00010100
sub ax, 10h	00011011	00010001	00010001	00010101	00010100

El contenido final de los registros es:

AX = 00011011 (binario) = 1B (hexadecimal) = 27 (decimal)

BX = 00010001 (binario) = 11 (hexadecimal) = 17 (decimal)

CX = 00010001 (binario) = 11 (hexadecimal) = 17 (decimal)

DX = 00010100 (binario) = 14 (hexadecimal) = 20 (decimal)

DS = 00010100 (binario) = 14 (hexadecimal) = 20 (decimal)

Explicación de lo que hace cada línea:

mov ax, 1001b: Se copia ese valor en el registro AX, sobrescribiendo cualquier valor anterior que tuviera.

mov bx, 19h: Se copia ese valor en el registro BX, sobrescribiendo cualquier valor anterior que tuviera.

mov cx, 16h: Se copia ese valor en el registro CX, sobrescribiendo cualquier valor anterior que tuviera.

mov dx, 21: Se copia ese valor en el registro DX, sobrescribiendo cualquier valor anterior que tuviera.

mov ds, ax: Se copia el valor que tiene el registro AX en el registro DS, sobrescribiendo cualquier valor anterior que tuviera.

sub bx, 12: Se resta 12 al valor que tiene el registro BX y guarda el resultado en el mismo registro BX, sobrescribiendo el valor anterior que tenía.

add ax, bx: Se suma el valor que tiene el registro BX al valor que tiene el registro AX y guarda el resultado en el mismo registro AX, sobrescribiendo el valor anterior que tenía.

div bl: Se divide el valor de AX entre el valor de BL. BL es la parte baja de BX, es decir, los 8 bits menos significativos. El resultado de la división se guarda en dos partes: el cociente se guarda en AL, que es la parte baja de AX, y el resto se guarda en AH, que es la parte alta de AX. AL y AH son los 8 bits menos y más significativos de AX, respectivamente.

add cx, bx: Se suma el valor que tiene el registro BX al valor que tiene el registro CX y guarda el resultado en el mismo registro CX, sobrescribiendo el valor anterior que tenía.

add ax, dx: Se suma el valor que tiene el registro DX al valor que tiene el registro AX y guarda el resultado en el mismo registro AX, sobrescribiendo el valor anterior que tenía.

sub ax, bx: Se resta el valor que tiene el registro BX al valor que tiene el registro AX y guarda el resultado en el mismo registro AX, sobrescribiendo el valor anterior que tenía.

mov bx, ds: Se copia el valor que tiene el registro DS en el registro BX, sobrescribiendo cualquier valor anterior que tuviera.

mov cx, bx: Se copia el valor que tiene el registro BX en el registro CX, sobrescribiendo cualquier valor anterior que tuviera.

mov dx, ax: Se copia el valor que tiene el registro AX en el registro DX, sobrescribiendo cualquier valor anterior que tuviera.

mov ds, dx: Se copia el valor que tiene el registro DX en el registro DS, sobrescribiendo cualquier valor anterior que tuviera.

add ax, 55: Se suma 55 al valor que tiene el registro AX y guarda el resultado en el mismo registro AX, sobrescribiendo el valor anterior que tenía.

sub ax, 10h: Se resta 10 al valor que tiene el registro AX y guarda el resultado en el mismo registro AX, sobrescribiendo el valor anterior que tenía.

2.- Escribe el uso o un concepto sobre los siguientes registros de estado (indicadores):

PF (Flag de Paridad):

La paridad es una propiedad que se utiliza en la detección de errores en transmisiones de datos. Si el resultado de una operación tiene una paridad par, el PF se establece en 1; de lo contrario, se establece en 0.

IF (Flag de Interrupción):

El registro de bandera de interrupción (IF) se utiliza para habilitar o deshabilitar las interrupciones en una CPU. Cuando está habilitado (IF = 1), la CPU atenderá interrupciones; cuando está deshabilitado (IF = 0), las interrupciones no se atenderán.

TF (Flag de Trampa):

El registro de bandera de trampa (TF) es utilizado para habilitar la depuración de programas. Cuando se activa (TF = 1), la CPU ejecuta instrucciones paso a paso, lo que facilita la depuración.

CF (Flag de Acarreo):

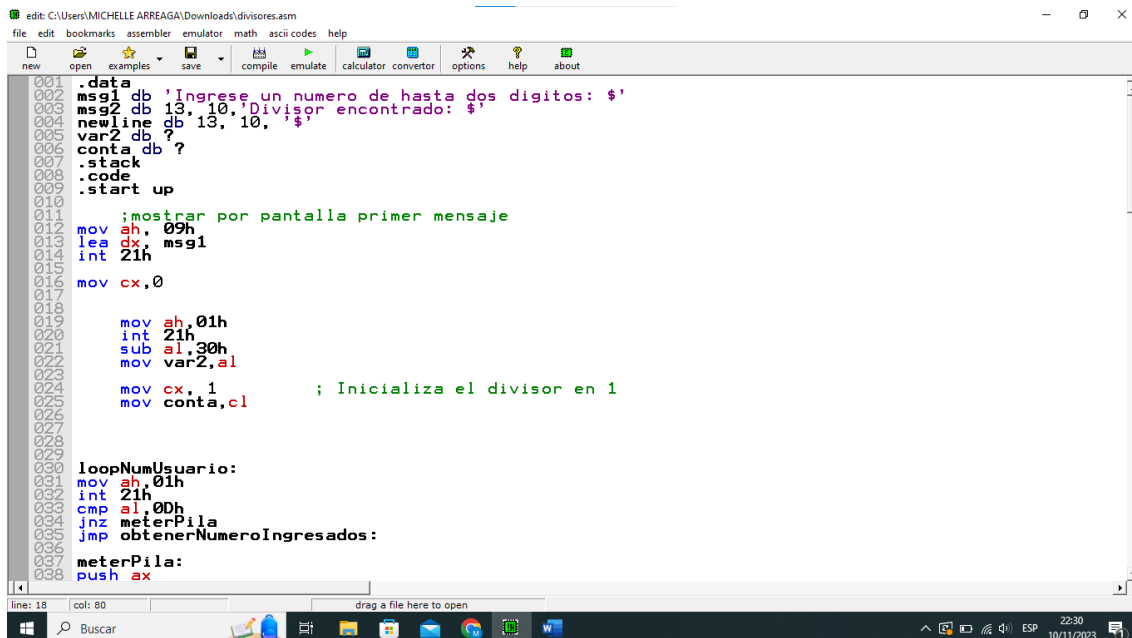
El acarreo es importante en operaciones aritméticas con números que desbordan su capacidad de representación en un número fijo de bits. El CF ayuda a detectar si se ha perdido información durante una operación.

OF (Flag de Sobreflujo):

El registro de bandera de sobreflujo (OF) se establece o borra en función del resultado de una operación aritmética. Indica si se produce un desbordamiento (overflow) durante una operación de suma o resta.

3.- Escribe un programa en Assembly que calcule y muestre todos los divisores de un número de hasta 2 cifras ingresado por el usuario. Además del código, coloca capturas de pantalla que muestren el funcionamiento del programa.

Código:



```
edit: C:\Users\MICHELLE ARREAGA\Downloads\divisores.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
001 data
002 msg1 db 'Ingrese un numero de hasta dos digitos: $'
003 msg2 db 13, 10, 'Divisor encontrado: $'
004 newline db 13, 10, '$'
005 var2 db ?
006 conta db ?
007 .stack
008 .code
009 .start up
010
011 ;mostrar por pantalla primer mensaje
012 mov ah, 09h
013 lea dx, msg1
014 int 21h
015
016 mov cx, 0
017
018
019
020 mov ah, 01h
021 int 21h
022 sub al, 30h
023 mov var2, al
024
025 mov cx, 1 ; Inicializa el divisor en 1
026 mov conta, cx
027
028
029
030 loopNumUsuario:
031 mov ah, 01h
032 int 21h
033 cmp al, 0Dh
034 jnz meterPila
035 jmp obtenerNumeroIngresados:
036
037 meterPila:
038 push ax
line: 18 col: 80 drag a file here to open
Buscar 22:30 10/11/2023
```

```
edit: C:\Users\MICHELLE ARREAGA\Downloads\divisores.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about

037 meterPila:
038 push ax
039 add cx,1
040 jmp loopNumUsuario
041
042 obtenerNumeroIngresados:
043 mov ah,09h
044 int 21h
045
046 ;obtener numero de usuario de la pila
047 primerNumero:
048 pop ax
049 sub ax,30h
050 cmp cx,2
051 jz segundoNumero
052
053
054 segundoNumero:
055 pop ax
056 sub ax,30h
057 mov bx,10
058 mul bx
059
060
061
062
063 calcular_divisores:
064
065 mov bx,cx ; Carga el divisor en el registro bx
066 xor dx,dx ; Inicializa dx en 0 para la division
067
068 mov al,var2
069 div bx ; Divide ax por bx, el cociente se almacena en ax, el residuo en dx
070
071 cmp dx,0 ; Compara el residuo con 0
072 je es_divisor ; Salta a la etiqueta 'es_divisor' si el residuo es 0
073
074 jmp siguiente_divisor ; Salta a la etiqueta 'siguiente_divisor' si el residuo no es 0
```

```
edit: C:\Users\MICHELLE ARREAGA\Downloads\divisores.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about

076 es_divisor:
077 ; Imprime el mensaje de divisor encontrado
078 mov ah,09h
079 lea dx,[msg2]
080 int 21h
081
082 ; Imprime el cociente
083 mov bx,cx
084 xor dx,dx
085 div bx
086 mov al,conta
087 mov dl,al
088 mov ah,02h
089 add dl,30h ;se suma 30h a dl para obtener el caracter ascii correcto
090 int 21h
091 mov al,var2
092 cmp cl,al
093 je salir
094
095
096 ; Imprime nueva linea
097 mov ah,09h
098 lea dx,[newline]
099 int 21h
100
101
102 siguiente_divisor:
103 inc cx
104 mov conta,cl ; Incrementa el divisor
105 mov ax,99 ;se pone el 99 para que se detenga cuando sea menor a 100 osea 99
106 cmp cx,ax ; Compara el divisor con el n'mero menos que 100
107 xor dx,dx
108 jle calcular_divisores ; Salta de nuevo a 'calcular_divisores' si el divisor es menor o igual
109
110 salir:
111
112
```

