

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO  
Organización y Programación de Computadoras

## Ejercicios BF

Mario Bros., G1

Integrantes

Mario Ivan Montes Vidal – 156167

Víctor Daniel Cruz González – 157948

Fecha (s) de elaboración

14 de octubre de 2018

## Ejercicios BF 10oct18

En el reporte, por cada ejercicio, muestre como resultado el despliegue de la ventana Console (cmd.exe) y explique su contenido. Entre su despliegue y lo que usted responda, deberá visualizarse la correcta justificación. Incluya en su respuesta todos los valores que se necesiten.

En cada ejercicio también despliegue la parte importante del programa en lenguaje ensamblador.

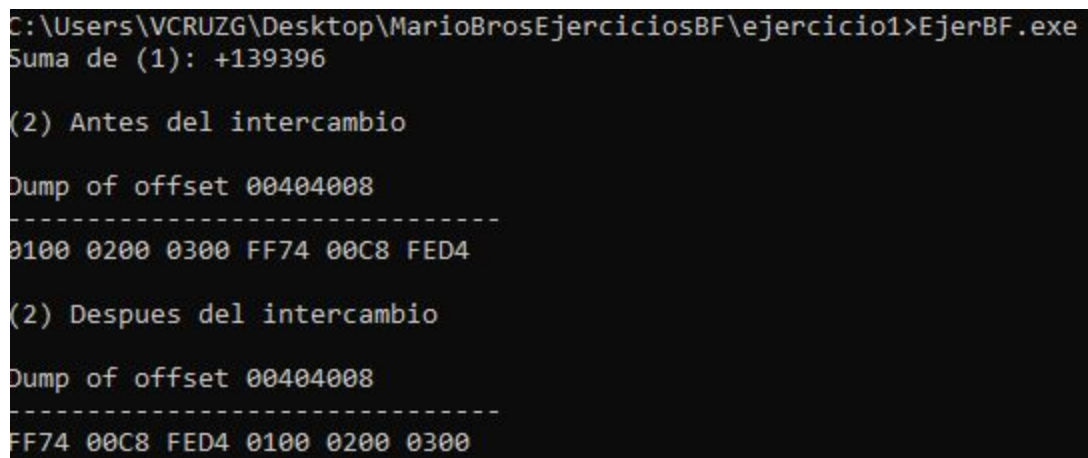
-----  
Los datos y resultados deben ser precedidos por un texto adecuado.

- 1- Modifique el programa ensamblador "EjerBF.asm" que sume los contenidos de *val3*, *val4*, el tercer elemento de *arrSW* y el segundo elemento de *arrSD*, e imprima el resultado en entero decimal con *WriteInt*.

Antes del despliegue imprima un mensaje adecuado, en el mismo renglón donde imprime el entero decimal.

También intercambie los contenidos entre *arrW* y *arrSW*. Imprima con *DumpMem* sus contenidos para comprobar el intercambio. Este intercambio debe ser lo más óptimo posible.

Antes del despliegue imprima un mensaje adecuado.



```

C:\Users\VCRUZG\Desktop\MarioBrosEjerciciosBF\ejercicio1>EjerBF.exe
Suma de (1): +139396

(2) Antes del intercambio

Dump of offset 00404008
-----
0100 0200 0300 FF74 00C8 FED4

(2) Despues del intercambio

Dump of offset 00404008
-----
FF74 00C8 FED4 0100 0200 0300
  
```

**Justificación:** El ejercicio se divide en dos partes: la primera, obtener el resultado (en EAX) de una suma de valores tanto positivos, como negativos, en específico, la suma de los primeros valores (*val3* y *val4*) da un número positivo (0x01B0), ya que ambos números son positivos, cabe destacar que no afecta los diferentes tipos de datos de cada uno de estos, porque ambos son mayores que 0. Para el siguiente número, el último número del arreglo *arrSW*, dado que es un número negativo (0xFED4), el resultado será un número positivo, pero menor que el resultado anterior, 0x0084. Por último, el último valor de *arrSD*, 0x00022000, creará un número positivo más grande, 0x00022084. Por lo tanto, el valor final, que está dentro del registro EAX, es 139396 en decimal.

Para la segunda parte del ejercicio, existe un intercambio de valores entre los datos o variables en memoria de los arreglos *arrW* y *arrSW*, puesto que el lenguaje ensamblador nos ayuda a hacer este tipo de acciones, sólo si se usa un registro, en este caso, decidimos usar EBX. En primer lugar, aprovechamos al máximo las propiedades de *cast* o la instrucción de PTR del lenguaje ensamblador, porque si cambiamos de word a double-word el tipo de valor de un elemento del arreglo, el sistema

usará 4 bytes próximos de donde se le indique que inicie a tomar los datos, por ende, al usar **DWORD PTR**, obtuvimos los dos primeros elementos del arreglo arrW y, al momento de hacer el intercambio con arrSW, los dos valores de los arreglos se cambiaron con una sola instrucción (XCHG). Para su último valor, nos basamos en el tipo WORD, ya que sólo restaba ese elemento, por lo que de arrW se cambió a EBX, de EBX se cambió el valor con arrSW y el nuevo valor de EBX se cambió a arrW para que todos los valores de arrW estuvieran en arrSW y viceversa.

```

; Pte. 1
mov EAX, 0 ; EAX = 0
mov AL, val3 ; EAX = val3
add AX, val4 ; EAX += val4
add AX, [arrSW + 4] ; EAX += [arrSW + 2]
add EAX, [arrSD + 4] ; EAX += [arrSD + 1]

mov ebx, SDWORD PTR [arrW]
xchg ebx, SDWORD PTR [arrSW]
xchg ebx, DWORD PTR [arrW]

mov bx, [arrW + 4]
xchg bx, [arrSW + 4]
xchg bx, [arrW + 4]

```

**Descripción:** Dado que el ejercicio se compone en dos partes diferentes, primero, observamos como la suma de los valores val3, val4, arrSW+4 (su último elemento) y arrSD+4 (su último elemento) dan un número positivo y del tipo DoubleWord, ya que es un resultado en decimal entero que cabe dentro de su intervalo y, segundo, existe un antes y un después de los contenidos en memoria de arrW y arrSW, ya que se hizo un intercambio de datos utilizando el registro EBX.

- 2- Elabore el programa ensamblador “**Segundo.asm**”, que dado un valor **n** entero, calcule e imprima la siguiente serie sumatoria de **n** términos:

$$1 + 4 + 7 + 10 + 13 + 16 + \dots$$

**n** es un dato entero  $\geq 1$ .

Deberá imprimir los **n** términos, así como el total resultante, con textos adecuados.

Para la lectura y escritura use los procedimientos de librería vistos en clase.

La siguiente es un despliegue de la ejecución del programa para una N de valor 4.

```

Teclee el dato N: 4

Siguiendo termino: +1
Siguiendo termino: +4
Siguiendo termino: +7
Siguiendo termino: +10

TOTAL: +22

```

```

Teclee el dato N: 4

Siguiendo termino: +1
Siguiendo termino: +4
Siguiendo termino: +7
Siguiendo termino: +10

TOTAL: +22

```

**Justificación:** El mayor reto de este ejercicio fue calcular / deducir la fórmula matemática que nos da cada componente de la suma dependiendo del valor que se ingrese. Por ejemplo, si  $m = 4$ , entonces tenemos una suma desde  $n = 1$  que va sumando el término dado por la ecuación  $(3n - 2)$  con  $n$  aumentando desde 1 hasta el valor  $m$  que se introduce.

$$\sum_{n=1}^m 3n - 2$$

#### Suma implementada en ensamblador

```

l1:
    MOV EAX, cont
    MUL alfa
    SUB EAX, 2
    mov edx, OFFSET mensaje2
    call WriteString
    MOV EBX, EAX
    call WriteInt
    ADD suma, EBX
    MOV EAX, 0
    MOV EBX, 0
    INC cont
    DEC ECX
    JNZ l1

```

Aquí podemos observar el código que implementa la suma mostrada anteriormente. El valor introducido se guarda en el registro ECX y se tiene un contador que empieza en  $n = 1$  guardado como una variable en memoria. Se toma el contador y se coloca en EAX, se le realiza una operación MUL (multiplicación por 3) y SUB 2 (resta 2 al valor de EAX); con esto ya tenemos el primer término de la suma, el cual se coloca en una variable [suma] en memoria. Se limpian los registros y se incrementa el contador en 1 y se decrementa el valor del registro ECX en 1. Todas estas operaciones están dentro de un ciclo compuesto por las condiciones JNZ (Jump is  $zf = 0$ ) que repite el ciclo hasta que el registro ECX está en cero. Lo que nos indica que la suma ya alcanzó el  $m$  número introducido al inicio. El resultado se imprime y se muestra según los formatos establecidos inicialmente.

```

l1:
    MOV EAX, cont
    MUL alfa
    SUB EAX, 2
    mov edx, OFFSET mensaje2
    call WriteString
    MOV EBX, EAX
    call WriteInt
    ADD suma, EBX
    MOV EAX, 0
    MOV EBX, 0
    INC cont
    DEC ECX
    JNZ l1

```

- 3- Elabore el programa ensamblador “**multiplicar.asm**”, y revise que datos dados a  $m$  y  $n$  sean  $m \geq 1$  y  $n \geq 0$ , ambos datos enteros) para calcular  $m*n$  sin utilizar la operación de multiplicación. En caso de no cumplirse la condición anterior el programa deberá terminar imprimiendo un texto de error e imprimiendo los valores de “m” y “n” tecleados.

HINT, se sugiere repetir  $m$  veces  $n$ .

Deberá imprimir el resultante producto, con el texto correspondiente.

```
C:\Users\VCRUZG\Desktop\MarioBrosEjerciciosBF\ejercicio3>multiplicar.exe
Inserta M: 0

Alguno de los datos NO cumplen con las condiciones (M>=1, N>=0; ENTEROS)
M: +0
N: +0

C:\Users\VCRUZG\Desktop\MarioBrosEjerciciosBF\ejercicio3>multiplicar.exe
Inserta M: 1

Inserta N: -1

Alguno de los datos NO cumplen con las condiciones (M>=1, N>=0; ENTEROS)
M: +1
N: -1

C:\Users\VCRUZG\Desktop\MarioBrosEjerciciosBF\ejercicio3>multiplicar.exe
Inserta M: 5

Inserta N: 12

RESULTADO: +60
```

**Justificación:** El lenguaje ensamblador ha pensado en varias opciones que son básicas para cualquier programador, por ejemplo, las 5 operaciones básicas para calcular números, aunque en este ejercicio, observamos que podemos prescindir de una si sabemos la definición básica de una multiplicación, es

decir, aumentar  $M$  veces un número  $N$  ( $\sum_{i=1}^M N$ ), por ende, si tan solo repetimos la operación de suma,

podemos obtener la manera análoga de la multiplicación. Una de las grandes ventajas de estas operaciones es que el tiempo de ejecución es extremadamente rápido (un ciclo de reloj), por lo que el procesamiento de los datos siempre será mucho más rápido.

En este caso, se hizo la misma operación de la suma, bajo un tipo de implementación que es análoga a un ciclo FOR, ya que el lenguaje ensamblador permite etiquetar líneas de código para *brincar* entre ellas y, así, ejecutar ciertas líneas de código y no todo el bloque de CODE.

El ensamblador reconoce que los datos siempre pueden variar dado que el usuario ingresará ciertos datos, por lo que automáticamente tiende a reconocer si la entrada es un número o un String (la función ReadInt nos ayuda en esto), aunque faltan los valores enteros aceptados dentro de la multiplicación que nosotros implementamos. Para reconocer que un número es mayor, menor, igual o diferente de algo, necesariamente hay que utilizar JMPs condicionales, en este caso, el mejor uso fue de *menor que* para ir a un caso donde aparezca un error o, en este caso, cuando  $M < 1$  y  $N < 0$ . Si no existe un error, la operación se realiza con normalidad, como se explicó previamente.

```
mult:
    add EAX, n
    dec EBX
    cmp EBX, 0
    JG mult
    JMP showRes
```

**Descripción:** Dado dos números,  $M \geq 1$  y  $N \geq 0$ , se realiza una operación de multiplicación, ya que estamos viendo el caso de números positivos enteros, en caso contrario, el programa arroja un error que indica cuál era el tipo de número que se introdujo en ambos casos. Una vez que se obtienen la pareja de números, el ejecutable imprime un mensaje con el resultado de la operación de multiplicar.

#### **OBSERVACIONES:**

- La respuesta a esta tarea deberá subirla a Comunidad, a la sección de TRABAJOS Y EXAMENES, a más tardar este domingo 14 de octubre, antes de las 23:30 hs.
- La primera página de este reporte deberá contener la portada sugerida en el archivo "caratulaTareas.docx" con la información correspondiente.
- El nombre del archivo zip donde usted pondrá el archivo respuesta, deberá llamarse con el nombre del grupo de trabajo seguido en este caso del texto EjerciciosBF, p.e. "**Spark EjerciciosBF.zip**", donde "Spark" sería el nombre de su grupo de trabajo.
- También incluir los tres programas fuente (archivos .asm).