Índice

1. Introducción

El problema planteado es el siguiente: *Utilizando la instrucción ADC, entre otras, realizar un programa en lenguaje ensamblador para arquitectura x86 que calcule la suma de los siguientes 2 números hexadecimales:*

1. **ADBC 45BD CF97 DCB3 26C4 89D7 0452 FE13**
2. **BDF3 96AC 435F E163 E307 417B FDE1 87CD**

   Almacenar el resultado en memoria de datos.

2. Planteamiento del problema

El primer aspecto importante se encuentra en la declaración que se hará inicialmente de los valores hexadecimales que se operarán, ya que, para poder manipularlos, se dividirán en secciones o *segmentos* de 16 [bits] separados en comas e invertido para que Turbo Debugger pueda operar con ellos correctamente.

Una vez hecho lo anterior, hay que declarar una variable donde pueda almacenarse el resultado de la operación entre ambos números inicializada en 0 en cada *segmento*; añadiendo un *segmento* ó 0 adicional para considerar una eventual operación con acarreo que pudiera presentarse.

```
;Datos de una word/palabra (16 bits)-----------------------------------------------
hexNumber1      dw      0FE13h, 0452h, 89D7h, 26C4h, 0DCB3h, 0CF97h, 45BDh, 0ADBCh
hexNumber2      dw      87CDh, 0FDE1h, 417Bh, 0E307h, 0E163h, 435Fh, 96ACh, 0BDF3h
hexTotal        dw      0, 0, 0, 0, 0, 0, 0, 0, 0; Variable declarada donde será alm
```

Figura 1: *Declaración de los números a utilizar en el programa.*

Ahora, para realizar el procedimiento inicial, el primer número será enviado al acumulador y, una vez hecho lo anterior, se sumará con la instrucción **add** para obtener el primer valor y almacenarlo en la variable *hexTotal*. Nótese que la suma fue realizada únicamente en el primer bloque o segmento de ambos números, y el resultado será almacenado en el primer bloque de 16 [bits] declarado para *hexTotal*. Así mismo, esta operación contiene un acarreo, así que la bandera $C = 1$, pero su valor no se considera en esta operación.

```
mov ax, [hexNumber1]; El contenido de la primer
add ax, [hexNumber2]; Se realiza la suma del pr
mov [hexTotal], ax  ; El contenido de AX se tra

;Vista actual: 85E0h, 0, 0, 0, 0, 0, 0, 0, 0
```

Figura 2: *Bloque de instrucciones iniciales.*

A continuación, se sigue el siguiente procedimiento para el resto de bloques:

- Se utiliza la operación **mov** para mover cada segmento de *hexNumber* añadiendo un desplazamiento de 2 en 2 conforme se recorre cada bloque de 16 [bits] de cada número.
- A diferencia del bloque de operaciones inicial, ahora se utiliza la operación **adc** para considerar el valor de C. De igual forma, el valor que será sumado corresponderá al siguiente bloque de 16 [bits] del segundo número hexadecimal, considerando el mismo deplazamiento de +2 unidades
- Finalmente, el resultado será trasladado por medio de la instrucción **mov** del acumulador a [*hexTotal + 2(n)*], considerando que en el bloque inicial $n = 0$ y que $n = 1, 2, 3, ..., 8$.

```
mov ax, [hexNumber1 + 2];
adc ax, [hexNumber2 + 2];
mov [hexTotal + 2], ax  ;

;Vista actual: 85E0h, 0234

mov ax, [hexNumber1 + 4];
adc ax, [hexNumber2 + 4];
mov [hexTotal + 4], ax  ;

;Vista actual: 85E0h, 0234

mov ax, [hexNumber1 + 6];
adc ax, [hexNumber2 + 6];
mov [hexTotal + 6], ax  ;
```

Figura 3: *Algoritmo principal del programa para realizar la suma de bloques.*

Finalmente, para $n = 8$, se envía un 0 a [*hexTotal + 16*] por medio de **adc** para que pueda considerarse un acarreo presente en el bloque $n = 7$.

```
mov ax, [hexNumber1 + 14]; Ahora, realizamos un desplazamiento de 14 ubicaciones para llegar a 0ADBCh. Entonces: AX = [hex
adc ax, [hexNumber2 + 14]; Se realiza la suma con el contenido de AX y el octavo "segmento" de hexNumber2, de allí: AX = [
mov [hexTotal + 14], ax  ; El contenido de AX se traslada a [hexTotal + 14] en la octava ubicación declarada, entonces: [h

;Vista actual: 85E0h, 0234h, 0CB53h, 09CBh, 0BE17h, 12F7h, DC6Ah, 6BAFh, 0

adc [hexTotal + 16], 0   ; Se suma el acarreo obtenido de la suma anterior, entonces: [hexTotal + 16] + 0000h + 1 = 0001h

;Vista actual: 85E0h, 0234h, 0CB53h, 09CBh, 0BE17h, 12F7h, DC6Ah, 6BAFh, 0001h
```

Figura 4: *Bloque final de instrucciones del programa.*

3. Diagrama de Flujo

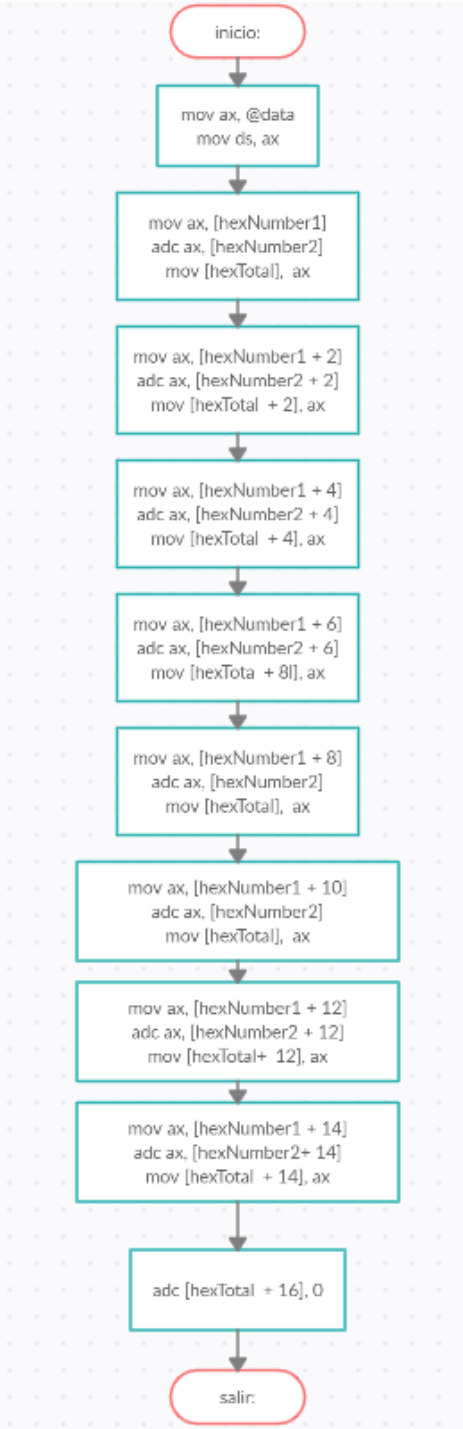

Figura 5: *Diagrama de flujo representativo del programa.*

4. Justificación de la solución

Una vez que el programa ha sido ensamblado y se ha generado el ejecutable **.exe**, se procede a abrir el programa en *Turbo Debugger*.

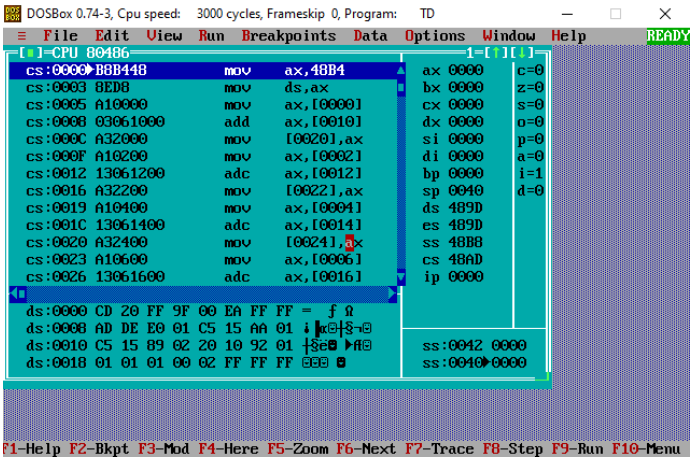

Figura 6: *Apertura del programa en Turbo Debugger*

Iniciando la ejecución del programa y verificando que cada paso se ha realizado correctamente, el resultado final deberá estar guardado en la memoria de datos. Para esto, se ejecuta todo el programa y por medio de **goto** se accede a la dirección **ds:0000** para verificar que efectivamente el resultado de la operación de suma se encuentra en el segmento de datos.

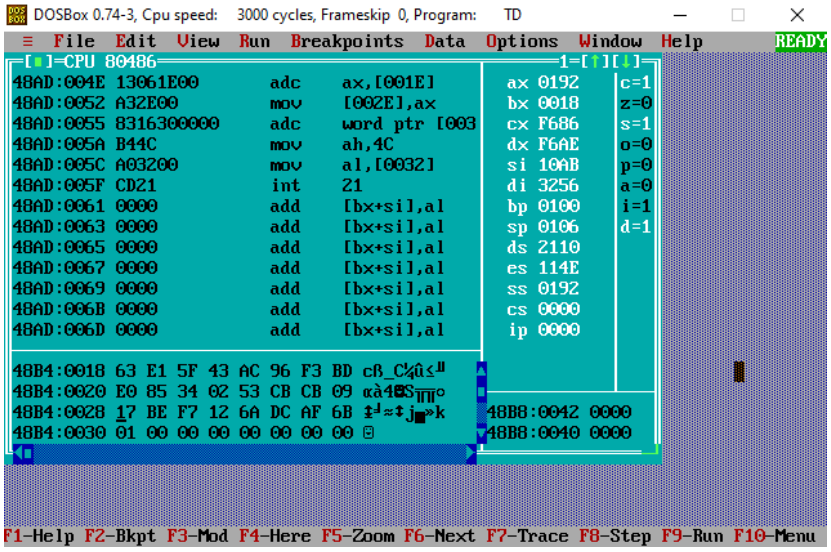

Figura 7: *Resultado esperado de la operación en DS:0030 a DS:0020.*

LaTeX