Práctica 4

Assembly, Instrucciones y Simulador MSX88

____ con algunos ejercicios resueltos

1) Dada la siguiente definición de datos y el código: **`F = [(A+B)/C]-D`** **RESUELTO**

nombre	tamaño	valor
A:	1 byte	6
B:	1 byte	4
C:	1 byte	2
D:	1 byte	1
F:	1 byte	?

Suponiendo que se poseen las instrucciones necesarias en cada caso, escribir el programa que implemente el código anterior utilizando máquinas de 1, 2 ó 3 direcciones.

Todas las operaciones se identifican en assembly, por lo general, con algunas letras de la palabra en inglés que identifica la operación a realizar. Así la suma es add (addition), la multiplicación es mul, la división es div y la resta es sub (substraction)

En máquinas de 1 dirección

a) Un operando se supone implícito, es el registro acumulador. Así load A (cargar A) supone cargar el acumulador con el valor de la variable A.
b) add B supone sumar al registro acumulador el contenido de la variable B, guardando el resultao en el acumulador.

En máquinas de 2 direcciones

a) El primer operando es el destino, donde se guarda el resultado, y los operandos fuentes son los dos operandos, en caso de las operaciones con 2 operandos fuente como add, sub, mul, div y sub.
b) O solo un segundo operando fuente en el caso de mov.

Así add F, B implica que se almacena en F (destino) el resultado de sumar F y A (fuentes)
En el caso de mov F, A implica que se copia el contenido de A (fuente) en F (destino)

En máquinas de 3 direcciones

a) El primer operando indicará el destino donde se almacena el resultado. Por lo general se elije uno, en este caso F y se mantiene, actuando como acumulador, aunque pudiera ser cualquiera.

Así add F, A, B indica que se suman los valores de A y B y el resultado se almacena en F.

b) En la siguiente instrucción del programa que sigue, lo que está en F se divide por el valor de C y el resultado se almacena en F.

Maq. de 1 dirección	Maq. de 2 direcciones	Maq. de 3 direcciones

`load A`
`add B`
`div C`
`sub D`
`store F`
	`Mov F, A`
	`add F, B`
	`div F, C`
	`sub F, D`
		`add F, A, B`
		`div F, F, C`
		`sub F, F, D`

2) Suponga que cada código de operación ocupa 6 bits y las direcciones son de 10 bits. Analice las soluciones implementadas en el ejercicio anterior y complete la siguiente tabla: **RESUELTO**

Advertencia! Hay un ejercicio similar en la teoría, que supone que las referencias a operandos no ocupan lugar en la memoria de instrucciones MI, por lo que cada instrucción usa un acceso a MI. No es lo que pasa en la realidad. Este caso, como los operandos son referenciados mediante direcciones de 10 bits, y se supone que el bus de tatos que comunica con la memoria es de 8 bits, cada operando necesitaría 2 accesos a MI (8 bits + 2 bits, pero debe obligadamente acceder a 8 bits). Además, debe acceder al código de operación, que especifica no solo de que operación se trata, sino que operandos necesita. Vamos a suponer en este ejercicio que una instrucción puede acceder a dos operandos en memoria, acceso conocido como memoria-memoria, pero no todos los procesadores, en particular el msx88, tienen dicha capacidad.

Accesos a memoria:

Se supone en este ejercicio que los códigos de operación ocupan un Byte de memoria. Las direcciones de las variables 10 bits, o sea ocupan 2 lugares de 1 Byte, y referencian un valor de 1 Byte = 8 bits. Por lo que todas las instrucciones acceden una vez a memoria de instrucciones (MI) para buscar el código de operación, 2 por cada dirección de operando (a MI) y una a la memoria de datos (MD) para buscar cada operando de memoria.

Código de operación (1 Byte)	Dirección/es operando	Total acceso a MI
Load	A (10bits, acceso a 2 Bytes)	3 accesos, ocupa 3 Bytes en MI
add (1 Byte)	F, B (2 x 10 bits, acceso a 2+2 = 4 Bytes)	5 accesos, ocupa 5 Bytes en MI

En máquinas de 1 dirección

a) load A accede una vez a memoria de instrucciones (MI) para buscar código de operación, dos más para la dirección de la variable, y una a la de datos (MD) para buscar el operando A.

b) En las instrucciones que haya resultado, el mismo se guarda en el registro acumulador, o sea no accede a memoria, por lo que las demás operaciones add, div, sub, etc. Acceden una sola vez a MD.

Código	Tamaño del programa en memoria (cod. op + operandos) en Bytes	Cantidad de accesos a memoria (instrucciones (MI) + operandos (MD))
`load A`	1 + 2 = 3	3 + 1 = 4
`add B`	1 + 2 = 3	3 + 1 = 4
`div C`	1 + 2 = 3	3 + 1 = 4
`sub D`	1 + 2 = 3	3 + 1 = 4
`store F`	1 + 2 = 3	3 + 1 = 4
Total: 5 instrucciones	**15 Bytes**	**20 Bytes**

En máquinas de 2 direcciones

a) add F, A y demás instrucciones, deben acceder una vez a MI para buscar la instrucción, dos más para cada referencia de las dos variables (4 accesos más a MI) y dos veces a MD, una para leer cada variable (como son 2, dos veces) y otra para escribir el resultado en la variable destino.

b) add A, B ---> 5 MI (1 instrucción, 2 x 2 operandos) --> 3 MD (leer A, leer B y escribir resultado en A)

c) mov F, A ---> 1 MI ---> 2 MD (leer A, escribir F)

Código	Tamaño del programa en memoria (cod. op + operandos) en Bytes	Cantidad de accesos a memoria (instrucciones (MI) + operandos (MD))
`mov F, A`	1 + 4 = 5	5 + 2 = 7
`add F, B`	1 + 4 = 5	5 + 3 = 8
`div F, C`	1 + 4 = 5	5 + 3 = 8
`sub F, D`	1 + 4 = 5	5 + 3 = 8
Total: 4 instrucciones	**20 Bytes**	**31 Bytes**

En máquinas de 3 direcciones

a) debe accederse una vez a MI para buscar la instrucción, dos veces para cada uno de las tres referencias a operandos fuentes y 3 veces a MD, dos para leer cada valor de cada operando y una más para almacenar el resultado en la variable destino.

b) add F, A, B ---> 7 MI ---> 3 MD (Leer A, Leer B y escribir F)

Código	Tamaño del programa en memoria (cod.op + operandos) en Bytes	Cantidad de accesos a memoria (instrucciones (MI) + operandos (MD))
`add F, A, B`	1 + 6 = 7	7 + 3 = 10
`div F, F, C`	1 + 6 = 7	7 + 3 = 10
`sub F, F, D`	1 + 6 = 7	7 + 3 = 10
Total: 3 instrucciones	**21 Bytes**	**30 Bytes**

	Maq. 1 dir.	Maq. 2 dir	Maq. 3 dir
Tamaño del programa en memoria (cod.operación + operandos)	15	20	21
Cantidad de accesos a memoria (instrucciones + operandos)	20	31	30

3) Dado el siguiente código: **F = ((A - B)*C) + (D/E)**; **A RESOLVER POR ALUMNO**

a) Implemente el código utilizando máquinas de 1, 2 y 3 direcciones.

b) Realice una tabla de comparación similar a la del ejercicio 2.

c) ¿Cuál máquina elegiría haciendo un balance de la cantidad de instrucciones, el espacio en memoria ocupado y el tiempo de ejecución (1 acceso a memoria = 1 ms)? ¿Es ésta una conclusión general?

Organización de Computadoras 2023

Ejercicios de Programación

Para cada programa propuesto en los siguientes ejercicios, deberá editar el archivo fuente con extensión asm (ej: ejer1.asm), con el notepad.exe, luego ensamblarlo usando asm88.exe (comando: asm88 ejer1.asm) y enlazarlo con link88.exe (comando: link88 ejer1.o). Cada archivo obtenido con extensión eje (ej: ejer1.eje) deberá ser cargado y ejecutado en el simulador MSX88. Se recomienda:

1) En la ejecución de asm88.exe le pregunta si el nombre del archivo será nulo.lst, por defecto, escriba el nombre que le dio a su programa, de tal manera que al ejecutarse el comando, si su archivo editado en Notepad se llamaba ejer1.asm, se generará un archivo ejer1.o y otro ejer1.lst. Este archivo **.lst** contendrá además del programa en texto, una tabla con los valores hexadecimal que representará la máquina en la memoria. Esto le permitirá seguir la ejecución del programa mirando el simulador MSX88 y el archivo ejer1.lst.
2) Ejecutar los programas cargados en el simulador MSX88 en el simulador paso a paso, oprimendo la tecla F6 o F7 en vez de utilizar el comando G, así tendrá tiempo de seguir el programa observando el simulador y el archivo .lst.

4) El siguiente programa utiliza una **instrucción de transferencia de datos** (instrucción MOV) con diferentes modos de direccionamiento para referenciar sus operandos. Ejecutar y analizar el funcionamiento de cada instrucción en el Simulador MSX88 observando el flujo de información a través del BUS DE DATOS, el BUS DE DIRECCIONES, el BUS DE CONTROL, el contenido de REGISTROS, de posiciones de MEMORIA, operaciones en la ALU, etc. **RESUELTO**

```
      ORG  1000h
NUM0  DB   0CAh
NUM1  DB   0
NUM2  DW   ?
NUM3  DW   0ABCDh
NUM4  DW   ?

      ORG  2000h
      MOV  BL, NUM0
      MOV  BH, 0FFh              ; inmediato
      MOV  CH, BL                ; registro
      MOV  AX, BX                ; registro
      MOV  NUM1, AL              ; directo
      MOV  NUM2, 1234h           ; inmediato
      MOV  BX, OFFSET NUM3       ; inmediato
      MOV  DL, [BX]              ; ind.via registro
      MOV  AX, [BX]              ; ind.via registro
      MOV  BX, 1006h             ; inmediato
      MOV  WORD PTR [BX],0CDEFh  ; inmediato
      HLT
      END
```

Cuestionario:

a) Explicar detalladamente qué hace cada instrucción MOV del programa anterior, en función de sus operandos y su modo de direccionamiento.
b) Confeccionar una tabla que contenga todas las instrucciones MOV anteriores, el modo de direccionamiento y el contenido final del operando destino de cada una de ellas.
c) Notar que durante la ejecución de algunas instrucciones MOV aparece en la pantalla del simulador un registro temporal denominado "**ri**", en ocasiones acompañado por otro registro temporal denominado "**id**". Explicar con detalle que función cumplen estos registros. Para ello observe que valor se copia en cada registro temporario ri e id.

ORG 1000h y ORG 2000h son directivas al ensamblador ASM88. ORG es la abreviatura de ORIGIN e indica a partir de que dirección (en hexadecimal, por eso la h final) de celda de memoria se ubicará el bloque. En el caso de ORG 1000h, obsérvese que se declaran datos, o sea corresponderá a memoria de datos (MD). Cada dato tiene 3 campos: una etiqueta, por ejemplo NUM0, que se corresponderá con la dirección (1000h en este caso). La dirección del dato siguiente será 1001h, ya que cada dato ocupa un Byte (por eso dice DB, si fuera de 2 Bytes sería DW). Es el segundo campo. El tercer campo es el valor en hexadecimal (h al final), OCAh para el primer caso. Se debe notar que el contenido es CA. El cero 0 al principio es debido a que se trata de un número, no una etiqueta, no agrega valor y el ensamblador entiende que es un número. En el caso del signo ? indica que no se inicializa con ningún valor, o sea tiene basura.

Respecto al sector de código en ORG 2000h, las instrucciones MOV suponen que el valor del segundo operando se copian en el primer operando. Las referencias de los operandos son diferentes para cada instrucción. Implican los usos de diferentes modos de direccionamiento.

Inmediato: Ejemplo `MOV BH, 0FFh.` El primer operando es un identificador de un registro de 8 bits. El segundo operando es un número, la instrucción se ejecuta de forma inmediata, sin acceder a memoria de datos a buscar datos. El número FF se copia al registro BH.

En el caso de `MOV BX, OFFSET NUM3` la palabra reservada para uso del ensamblador OFFSET indica que a BX se copiará no el valor contenido en la dirección simbolizada por la etiqueta NUM3, sino la dirección misma. Si hacemos la cuenta, a partir de 1000h tenenos NUM0 DB (un Byte) dirección 1000h, NUM1 también DB dirección 1001h, NUM2 DW (2 Bytes), dirección 1001h para Byte de parte baja, y 1002h para parte alta, Por lo que la etiqueta NUM3 se corresponde con la dirección 1003h, dirección que será copiada en el registro BX.

Directo: Ejemplo `MOV BL, NUM0.` El primer operando es un registro de 8 bits, el segundo es una etiqueta, o sea una representación simbólica de la dirección de esa etiqueta. El número 0, contenido en NUM0, se copia en BH. La dirección del operando está incluida en la instrucción, característica del modo directo, no hace falta buscarla.

Registro: ambos operandos son registros, lo que está en el segundo se copia en el primero. Observese que ambos registros deben tener el mismo tamaño, por ejemplo CH y BL de 8 bits o AX y BX de 16 bits.

Indirecto vía registro: si bien hay varios modos de direccionamiento indirecto, el MSX88 soporta solo el modo indirecto vía registro, caracterizado por los corchetes alrededor de [BX]. BX es el único registro con posibilidad de usarse como este tipo de direccionamiento. En este modo, BX tiene como contenido una dirección, o sea una vez cargada la instrucción desde l MI, la dirección debe ser leída accediendo al registro BX. Veamos dos ejemplos:

`MOV DL, [BX]` ; desde la dirección indicada en BX se copia un Byte al registro DL

`MOV AX, [BX]` ; desde la dirección indicada en BX se copia un Byte al registro AL, y desde la dirección BX + 1 otro Byte al registro AH, completándose de esta manera los 2 Bytes de tamaño del registro AX.

`MOV WORD PTR [BX],0CDEFh` ; al no haber un registro ni fuente ni destino, se debe especificar si se copia un Byte o 2 Bytes. Esto es por la directive WORD PTR (2 Bytes) o BYTE PTR (1 Byte). Siempre serán las celdas especificadas por el valor contenido en BX, y siguiente, BX +1 en el caso de 2 Bytes.

La instrucción HLT indica al procesador que debe detenerse y no buscar mas instrucciones ejecutables.

END es una directiva al ensamblador para indicarle que no hay maslinaes de programa para ensamblar. Detalle: al editar el programa, luego del ésta directiva se debe introducir un ENTER, sino dará error el ensamblado con el commando ASM88.

5) El siguiente programa utiliza diferentes **instrucciones de procesamiento de datos** (instrucciones aritméticas y lógicas). Analice el comportamiento de ellas y ejecute el programa en el MSX88. **A COMPLETAR**

ADD suma los dos operando, guarda resultado en el primero
INC suma 1 al operando especificado
DEC resta 1 al operando especificado
AND hace la operación lógica and entre ambos operandos, guardando el resultado en el primer operando
NOT hace el complemento a uno, cambia 1 x 0 y 0 x 1.
OR hace la operación OR entre ambos operandos, guardando el resultado en el primer operando
XOR hace el OR exclusivo entre ambos operando, guardando el resultado en el primer operando

```
        ORG 1000H
NUM0    DB  80h
NUM1    DB  200
NUM2    DB  -1
BYTE0   DB  01111111B
BYTE1   DB  10101010B

        ORG  2000H
        MOV  AL, NUM0
        ADD  AL, AL
        INC  NUM1
        MOV  BH, NUM1
        MOV  BL, BH
        DEC  BL
```

```
        SUB  BL, BH
        MOV  CH, BYTE1
        AND  CH, BYTE0
        NOT  BYTE0
        OR   CH, BYTE0
        XOR  CH, 11111111B
        HLT
        END
```

Cuestionario:

a) ¿Cuál es el estado de los FLAGS después de la ejecución de las instrucciones ADD y SUB del programa anterior? Justificar el estado (1 ó 0) de cada uno de ellos. ¿Dan alguna indicación acerca de la correctitud de los resultados?

b) ¿Qué cadenas binarias representan a NUM1 y NUM2 en la memoria del simulador? ¿En qué sistemas binarios están expresados estos valores?

c) Confeccionar una tabla que indique para cada operación aritmética ó lógica del programa, el valor de sus operandos, en qué registro o dirección de memoria se almacenan y el resultado de cada operación.

6) El siguiente programa implementa un contador utilizando una **instrucción de transferencia de control**. Analice el funcionamiento de cada instrucción y en particular las del lazo repetitivo que provoca la cuenta.

COMENTADO

```
        ORG  1000H
  INI   DB   0
  FIN   DB   15

        ORG  2000H
        MOV  AL, INI
        MOV  AH, FIN
 SUMA:  INC  AL
        CMP  AL, AH
        JNZ  SUM
        HLT
        END
```

Observe que el salto JNZ SUM (ir a ejecutar la instrucción identificada por la etiqueta SUM) se ejecuta solo si es FALSO el valor del flag Z. Este valor se determina en la operación anterior, CMP AL, AH. Esta instrucción hace la resta AL – AH pero no escribe el resultado en AL. Sin embargo, los flags son afectados. Sólo en el caso de que AL = AH el resultado será todos bits cero, en cuyo caso el flaz Z será VERDADERO y el salto no se efectúa y la siguiente instrucción, HLT, será ejecutada.

Cuestionario:

a) ¿Cuántas veces se ejecuta el lazo? ¿De qué variables depende esto en el caso general?

b) Analice y ejecute el programa reemplazando la instrucción de salto condicional JNZ por las siguientes, indicando en cada caso el contenido final del registro AL:

1º) JS
2º) JZ
3º) JMP

7) Escribir un programa en lenguaje assembly del MSX88 que implemente la sentencia condicional de un lenguaje de alto nivel IF **A** <**B** THEN **C** = **A** ELSE **C** = **B**. Considerar que las variables de la sentencia están almacenadas en los registros internos de la CPU del siguiente modo **A** en AL, **B** en BL y **C** en CL.

Determine las modificaciones que debería hacer al programa si la condición de la sentencia IF fuera:

a) A ≤ B

A RESOLVER POR ALUMNO

b) A = B

8) El siguiente programa suma todos los elementos de una tabla almacenada a partir de la dirección 1000H de la memoria del simulador. Analice el funcionamiento y determine el resultado de la suma. Comprobar resultado en el MSX88.

RESUELTO

```
        ORG 1000H
 TABLA  DB  2,4,6,8,10,12,14,16,18,20
```

```
FIN     DB  ?
TOTAL   DB  ?
MAX     DB  13
        ORG 2000H
        MOV AL, 0
        MOV CL, OFFSET FIN-OFFSET TABLA
        MOV BX, OFFSET TABLA
SUMA:   ADD AL, [BX]
        INC BX
        DEC CL
        JNZ SUMA
        HLT
        END
```

Observe que la instrucción ADD AL, [BX] va sumando a AL, inicialmente en 0, los valores direccionados por el registro BX (direccionamiento indirecto vía registro). Pero la instrucción siguiente INC BX provoca que dicha dirección apunte al siguiente elemento del arreglo TABLA, todos DB, ubicados en direccione de memoria contiguas a partir de la dirección 1000H.entre hasta la dirección 1009H, con lo que la etiqueta FIN se corresponde con la dirección 100AH (nótese que al ser hexadecimal el número siguiente a 1009H es 100Ah).

La instrucción `MOV CL, OFFSET FIN - OFFSET TABLA` implica restar a la dirección de la etiqueta FIN la dirección inicial TABLA, su resultado es 10, que son la cantidad de elementos de TABLA. La instrucción DEC CL irá restando 1 por cada valor sumado de la tabla, y cuando se sumen los 10 elementos, tendrá valor 0 y JNZ SUMA no efectuará el saldo a la etiqueta SUMA, y el programa finalizara con el HLT.

¿Qué modificaciones deberá hacer en el programa para que el mismo almacene el resultado de la suma en la celda etiquetada TOTAL?

9) Escribir un programa que, utilizando las mismas variables y datos que el programa del punto anterior (TABLA, FIN, TOTAL, MAX), determine cuántos de los elementos de TABLA son menores o iguales que MAX. Dicha cantidad debe almacenarse en la celda TOTAL. **A RESOLVER POR ALUMNO**

10) Analizar el funcionamiento del siguiente programa. **A RESOLVER POR ALUMNO**

```
        ORG 2000H
        MOV AX, 1
        MOV BX, 1000h
CARGA:  MOV [BX], AX
        ADD BX, 2
        ADD AX, AX
        CMP AX, 200
        JS  CARGA
        HLT
        END
```

Cuestionario:

a) El programa genera una tabla. ¿Cómo están relacionados sus elementos entre sí?

b) ¿A partir de qué dirección de memoria se crea la tabla? ¿Cuál es la longitud de cada uno de sus elementos (medida en bits)?

c) ¿Cuántos elementos tiene la tabla una vez finalizada la ejecución del programa? ¿De qué depende esta cantidad?

11) Escribir un programa que genere una tabla a partir de la dirección de memoria almacenada en la celda DIR con los múltiplos de 5 desde cero hasta MAX. **A RESOLVER POR ALUMNO**

12) Escribir un programa que, dado un número X, genere un arreglo con todos los resultados que se obtienen hasta llegar a 0, aplicando la siguiente fórmula: si X es par, se le resta 7; si es impar, se le suma 5, y al resultado se le aplica nuevamente la misma fórmula. Ej: si X = 3 entonces el arreglo tendrá: 8, 1, 6, -1, 4, -3, 2, -5, 0.
A RESOLVER POR ALUMNO

13) Dada la frase "Organización y la Computación", almacenada en la memoria, escriba un programa que determine cuantas letras ‘a’ seguidas de ‘c’ hay en ella. **A RESOLVER POR ALUMNO**

Obsverve que la frase "Organización y la Computación" puede ser contenida en una declaración de dato como la que sigue:

```
        ORG 1000H
```

```
FRASE  DB  "Organización y la Computación"
```

Donde a partir de la dirección 1000H se almacenarán en las direcciones sucesivas los caracteres ASCII de 8 bits correspondientes a las letras de la frase.

14) Escribir un programa que sume dos números representados en Ca2 de 32 bits almacenados en memoria de datos y etiquetados NUM1 y NUM2 y guarde el resultado en RESUL (en este caso cada dato y el resultado ocuparán 4 celdas consecutivas de memoria). Verifique el resultado final y almacene 0FFH en la celda BIEN en caso de ser correcto o en otra MAL en caso de no serlo. Recordar que el MSX88 trabaja con números en Ca2 pero tener en cuenta que las operaciones con los 16 bits menos significativos de cada número deben realizarse en BSS. **A RESOLVER POR ALUMNO**

15) Escribir un programa que efectúe la suma de dos vectores de 6 elementos cada uno (donde cada elemento es un número de 32 bits) almacenados en memoria de datos y etiquetados TAB1 y TAB2 y guarde el resultado en TAB3. Suponer en primera instancia que no existirán errores de tipo aritmético (ni carry ni overflow), luego analizar y definir los cambios y agregados necesarios que deberían realizarse al programa para tenerlos en cuenta. **RESUELTO**

```
      ; Memoria de Datos
      ORG 1000H
 TAB1 DW 1, 12, 5, 10     ; Los num de tabla 1 parte baja y seguido parte alta
      DW 25, 48, 12, 25
      DW 51, 20, 1, 4
 TAB2 DW 4, 26, 25, 42    ; Los num de tabla 2 parte baja y parte alta
      DW 23, 7, 2, 56
      DW 14, 3, 14, 74
 TAB3 DW 12 DUP (?)       ; en TAB3 se guarda la suma de TAB1 y TAB2, con DUP
                          ; dejamos 12 lugares
 CANT DB 12
 DIR3 DW ?

      ; Memoria de Instrucciones
      ORG 2000H
      MOV AX, OFFSET TAB1  ; guarda en AX la dirección del comienzo de TAB1
      MOV CX, OFFSET TAB2  ; guarda en CX la dir de comienzo de TAB2
      MOV DIR3, OFFSET TAB3 ; guardo en DIR la dir de comienzo de TAB3
LAZO: MOV BX, AX          ; copia en BX el valor que contiene el registro AX
      MOV DX, [BX]        ; copia en DX el valor del num que está en la dir BX
      MOV BX, CX          ; copia en BX el valor que contiene el registro CX
      ADD DX, [BX]        ; Suma el contenido de DX, con el valor del num que
                          ; está en la dir BX
      MOV BX, DIR3        ; copia en BX enl valor de DIR3
      MOV [BX], DX        ; copia en la dirección contenida en el registro BX,
                          ; el valor de DX
      ADD AX, 2           ; suma 2 al registro AX
      ADD CX, 2           ; suma 2 al registro CX
      ADD DIR3, 2         ; suma 2 al valor contenido en DIR3
      DEC CANT            ; CANT controla la cantidad de veces que se repite el lazo
      JNZ LAZO            ; Si flag Z no es cero, se repite el lazo
      HLT
      END
```

Se deben analizar los cambios para tener en cuenta errores aritméticos (carry y overflow).

16) Los siguientes programas realizan la misma tarea, en uno de ellos se utiliza una **instrucción de transferencia de control con retorno**. Analícelos y compruebe la equivalencia funcional.

A RESOLVER POR ALUMNO

```
; Memoria de Datos
      ORG 1000H
NUM1  DB    5H
NUM2  DB    3H

; Memoria de Instrucciones
      ORG 2000H
      MOV   AL, NUM1
      CMP   AL, 0
      JZ    FIN
      MOV   AH, 0
      MOV   DX, 0
      MOV CL, NUM2
LOOP: CMP   CL, 0
      JZ    FIN
      ADD   DX, AX
      DEC   CL
      JMP   LOOP
FIN:  HLT
      END
```

```
; Memoria de Datos
      ORG 1000H
NUM1  DB    5H
NUM2  DB    3H

; Memoria de Instrucciones
      ORG 3000H    ; Subrutina SUB1
SUB1: CMP   AL, 0
      JZ    FIN
      CMP   CL, 0
      JZ    FIN
      MOV   AH, 0
      MOV   DX, 0
LAZO: ADD   DX, AX
      DEC   CX
JNZ   LAZO
FIN:  RET

      ORG 2000H    ; Programa principal
      MOV   AL, NUM1
      MOV   CL, NUM2
      CALL  SUB1
      HLT
      END
```

Responder:

1) ¿Cuál es la tarea realizada por ambos programas?
2) ¿Dónde queda almacenado el resultado?
3) ¿Cuál programa realiza la tarea más rápido? ¿El tiempo de ejecución de la tarea depende de los valores almacenados en NUM1, en NUM2, en ambos lugares o en ninguno?

Explicar detalladamente:

a) Todas las acciones que tienen lugar al ejecutarse la instrucción CALL SUB1.
b) ¿Qué operación se realiza con la instrucción RET?, ¿cómo sabe la CPU a qué dirección de memoria debe retornar desde la subrutina al programa principal?

El siguiente programa es otra forma de implementación de la tarea del punto anterior (ejercicio 16). Analizar y establecer las diferencias con las anteriores, en particular las relacionadas a la forma de 'proveer' los operandos a las subrutinas.

RESUELTO

```
      ; Memoria de datos
      ORG 1000H
NUM1  DW    5H    ; NUM1 y NUM2 deben ser mayores que cero
NUM2  DW    3H
      ; Memoria de Instrucciones
      ORG 3000H    ; Subrutina SUB2
SUB2: MOV   DX, 0        ; Mueve a DX un 0
LAZO: MOV   BX, AX; Mueve a BX AX. En AX está la dir de NUM1
      ADD   DX, [BX]     ; Suma a DX el valor que apunta BX, que es 5H
      PUSH  DX           ; Apila el valor de DX. A SP se le resta dos. Pasa de 7FFEH a
                         ; 7FFCH. Y en esa dirección se guarda el valor de DX.
      MOV   BX, CX; Mueve a BX el valor en CX que es la dir de NUM2
      MOV   DX, [BX]     ; Mueve a DX el valor que apunta BX. En la primera iteración
                  ; será 3H. DX se convierte en un registro de ayuda para procesar los
                  ; datos que hay en la pila y donde se calculan las iteraciones
                  ; restantes y el valor acumulado de la multiplicación.
      DEC   DX           ; Decrementa el valor de DX
      MOV   [BX], DX     ; Guarda en NUM2 el valor de DX.
      POP   DX     ; Desapila en DX la suma de DX y [BX] anterior. A SP se le suma dos.
                   ; Pasa de 7FFCH a 7FFEH. En DX queda el valor apuntado por esa direc-;
                  ción. Notar que SP se decrementa e incrementa en cada iteración.
      JNZ   LAZO   ; Si DEC DX dio como resultado valor distinto a 0 en DX salta a LAZO.
      RET          ; sino retorna al programa principal.
                   ; Dejando en DX, el valor de NUM1 multiplicado por NUM2
```

```
      ORG 2000H    ; Programa principal
      MOV   AX, OFFSET NUM1    ; Se carga en AX la dir de NUM1. Con valor 1000H
      MOV   CX, OFFSET NUM2    ; Se carga en CX la dir de NUM2. Con valor 1002H
      CALL  SUB2         ; Se invoca la subrutina SUB2. Si SP, que es el puntero a
                         ; la pila "STACK POINTER", está en 8000H se le resta 2.
            ; Esto es debido a que la pila sólo almacena datos de 16 bits.
            ; Observe que la pila crece a direcciones de memoria mas chicas.
            ; Entonces en la dir apuntada por SP (7FFEH) se guarda la dirección de
            ; retorna a la siguiente instrucción (HLT)
      HLT
      END
```

Explicar detalladamente:

a) Todas las acciones que tienen lugar al ejecutarse las instrucciones PUSH DX y POP DX.
b) Cuáles son los dos usos que tiene el registro DX en la subrutina SUB2.

18) Escribir un programa que sume 2 vectores de 6 elementos (similar al realizado en el ejercicio 15), de modo tal que utilice una subrutina que sume números de 32 bits (similar al programa escrito en ejercicio 14). **A RESOLVER**

19) Escriba una subrutina que reciba la mantisa entera en BSS y el exponente en BSS de un número en los registros AH y AL respectivamente y devuelva, en ellos, una representación equivalente del mismo pero con el exponente disminuido en 1 y la mantisa ajustada. De no ser posible el ajuste, BL debe contener 0FFH en vez de 00H en el retorno. **RESUELTO**

```
Listado Fuente: P4Ej19.lst
Programa Fuente en: P4Ej19.asm
Fecha: Sun Jun 21 10:02:19 2020

Dir. Cod. Maq.     Linea Codigo en lenguaje ensamble
                   1       ORG 1000h
1000 C2            2 MANTISA   DB 11000010B
1001 03            3 EXPONENTE DB 3
                   4
                   5       ORG 3000H
                   6     ; Disminuye el exponente (en AL) en 1 y ajusta la
                   7     ; mantisa (en AH) moviendola un bit a la izquierda.
                   8     ; BL = 00H si se pudo ajustar, BL = 0FFH caso contrario
3000 FE C8         9 AJUSTAR: DEC AL
3002 02 E4         10      ADD AH, AH    ; Mover a la izq. es multiplicar x2
3004 72 05         11      JC  NO-AJUSTA ; Si da carry, se fué de rango (no ajustado)
3006 B3 00         12      MOV BL, 00H   ; Indicador de que pudo ajustar la mantisa
3008 E9 0D 30      13      JMP FIN       ; Vuelve de la subrutina
300B B3 FF         14 NO-AJUSTA: MOV BL, 0FFH ; Indica mantisa NO ajustada
300D C3            15 FIN: RET
                   16
                   17      ORG 2000H
2000 8A 26 00 10   18      MOV AH, MANTISA
2004 8A 06 01 10   19      MOV AL, EXPONENTE
2008 E8 00 30      20      CALL AJUSTAR
200B F4            21      HLT
            22       END
S I M B O L O S:

Nombre:      Tipo:       Valor:

MANTISA      Byte        1000h
EXPONENTE    Byte        1001h
AJUSTAR      Label       3000h
NO-AJUSTA    Label       300Bh
FIN          Label       300Dh
```

20) Escriba una subrutina que reciba como parámetro un número en el formato IEEE 754 de simple precisión y analice/verifique las características del mismo devolviendo en el registro CL un valor igual a 0 si el número está sin normalizar, 1 en caso de ser +/- infinito, 2 si es un NAN, 3 si es un +/- cero y 4 si es un número normalizado. La subrutina recibe en AX la parte alta del número y en BX la parte baja. **RESUELTO**

```
ORG 1000h
UNOl   DW    0      ; El numero 1.0
UNOh   DW    03F80h  ; 0 01111111 0000000 00000000 00000000
SN1l   DW    0FFFFh  ; El mas grande sin normalizar
SN1h   DW    7Fh    ; 0 00000000 1111111 11111111 11111111
SN2l   DW    1      ; El mas chico sin normalizar
SN2h   DW    0      ; 0 00000000 0000000 00000000 00000001
NAN1l  DW    1      ; Un NaN
NAN1h  DW    7F80h  ; 0 11111111 0000000 00000000 00000001
NAN2l  DW    0      ; Otro NaN
NAN2h  DW    7F81h  ; 0 11111111 0000001 00000000 00000000
ZEROl  DW    0
ZEROh  DW    0
INFl   DW    0      ; + infinito
INFh   DW    7f80h  ; 0 11111111 0000000 00000000 00000000
; esto está a partir de la dirección de memoria 101Ch
RESuno   DB  ?      ; finaliza en 4
RESsn1   DB  ?      ; finaliza en 0
RESsn2   DB  ?      ; finaliza en 0
RESnan1  DB  ?      ; finaliza en 2
RESnan2  DB  ?      ; finaliza en 2
RESzero  DB  ?      ; finaliza en 3
RESinf   DB  ?      ; finaliza en 1

      ORG 3000h
TIPO-IEEE: MOV DX, AX
      AND DX, 7F80h      ; 01111111 10000000
      JZ E-CERO    ; si Z = 1, el exponente es cero
      XOR DX, 7F80h
      JZ E-UNOS    ; si Z = 1, el exponente es todos unos
      MOV CL, 4    ; si el exponente no es ni 0 y 255 es un numero normalizado
      RET
E-CERO: MOV DX, AX
      AND AX, 7Fh  ; 00000000 01111111
      JNZ SIN_NOR  ; si la mantisa no es cero, es un numero denormalizado
      MOV DX, BX
      AND DX, 0FFFFh
      JNZ SIN-NOR  ; si la mantisa no es cero, es un numero denormalizado
      MOV CL, 3    ; la mantisa es cero, y el exponente tambien, es +/- 0.
      RET
E-UNOS: MOV DX, AX
      AND DX, 7Fh
      JNZ N-a-N    ; si la mantisa no es cero, es un NaN
      MOV DX, BX
      AND DX, 0FFFFh
      JNZ N-a-N    ; si la mantisa no es cero, es un NaN
      MOV CL, 1    ; si el exponente es 255 y la mantisa es cero, es +/- Inf
      RET
SIN-NOR: MOV CL, 0
```

```
        ORG 2000h
        MOV BX, UNOl
        MOV AX, UNOh
        CALL TIPO-IEEE
        MOV RESuno, CL
        MOV BX, SN1l
        MOV AX, SN1h
        CALL TIPO-IEEE
        MOV RESsn1, CL
        MOV BX, SN2l
        MOV AX, SN2h
        CALL TIPO-IEEE
        MOV RESsn2, CL
        MOV BX, NAN1l
        MOV AX, NAN1h
        CALL TIPO-IEEE
        MOV RESnan1, CL
        MOV BX, NAN2l
        MOV AX, NAN2h
        CALL TIPO-IEEE
        MOV RESnan2, CL
        MOV BX, ZEROl
        MOV AX, ZEROh
        CALL TIPO-IEEE
        MOV RESzero, CL
        MOV BX, INFl
        MOV AX, INFh
        CALL TIPO-IEEE
        MOV RESinf, CL
        HLT
        END
```

21) Modifique la subrutina del ejercicio 19 para el caso en que la mantisa y el exponente estén representados en BCS. **A RESOLVER**