**Trabajo Práctico N° 4:**

**Assembly, Instrucciones y Simulador MSX88.**

**Ejercicio 1.**

*Dada la siguiente definición de datos y el código F= - D.*

|  |  |  |
| --- | --- | --- |
| **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 o 3 direcciones.*

Máquina de 1 dirección:

load A

add B

div C

sub D

store F

Instrucciones: 5.

Tamaño del programa en memoria: 10 bytes.

Accesos: 10 MI y 5 MD.

Máquina de 2 dirección:

mov F, A

add F, B

div F, C

sub F, D

Instrucciones: 4.

Tamaño del programa en memoria: 12 bytes.

Accesos: 12 MI y 11 MD.

Máquina de 3 dirección:

add F, A, B

div F, F, C

sub F, F, D

Instrucciones: 3.

Tamaño del programa en memoria: 12 bytes.

Accesos: 12 MI y 9 MD.

**Ejercicio 2.**

*Suponer que cada código de operación ocupa 6 bits y las direcciones son de 10 bits. Analizar las soluciones implementadas en el ejercicio anterior y completar la siguiente tabla:*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Máquina de 1 dirección** | **Máquina de 2 direcciones** | **Máquina de 3 direcciones** |
| Tamaño del programa en memoria (código de operación más operandos) | |  |  | | --- | --- | | load A | 1 + 2= 3 | | add B | 1 + 2= 3 | | div C | 1 + 2= 3 | | sub D | 1 + 2= 3 | | store F | 1 + 2= 3 | | **total** | **15 bytes** | | |  |  | | --- | --- | | mov F, A | 1 + 4= 5 | | add F, B | 1 + 4= 5 | | div F, C | 1 + 4= 5 | | sub F, D | 1 + 4= 5 | | **total** | **20 bytes** | | |  |  | | --- | --- | | add F, A, B | 1 + 6= 7 | | div F, F, C | 1 + 6= 7 | | sub F, F, D | 1 + 6= 7 | | **total** | **21 bytes** | |
| Cantidad de accesos a memoria (instrucciones + operandos) | |  |  | | --- | --- | | load A | 3 + 1= 4 | | add B | 3 + 1= 4 | | div C | 3 + 1= 4 | | sub D | 3 + 1= 4 | | store F | 3 + 1= 4 | | **total** | **20 bytes** | | |  |  | | --- | --- | | mov F, A | 5 + 2= 7 | | add F, B | 5 + 3= 8 | | div F, C | 5 + 3= 8 | | sub F, D | 5 + 3= 8 | | **total** | **31 bytes** | | |  |  | | --- | --- | | add F, A, B | 7 + 3= 10 | | div F, F, C | 7 + 3= 10 | | sub F, F, D | 7 + 3= 10 | | **total** | **30 bytes** | |

**Ejercicio 3.**

*Dado el siguiente código: F= (A - B) C + .*

**(a)** *Implementar el código utilizando máquinas de 1, 2 y 3 direcciones.*

Máquina de 1 dirección:

load A

sub B

mul C

store A

load D

div E

add A

store F

Instrucciones: 8.

Tamaño del programa en memoria: 16 bytes.

Accesos: 16 MI y 8 MD.

Máquina de 2 dirección:

mov F, A

sub F, B

mul F, C

div D, E

add F, D

Instrucciones: 5.

Tamaño del programa en memoria: 15 bytes.

Accesos: 15 MI y 14 MD.

Máquina de 3 dirección:

sub F, A, B

mul F, F, C

div D, D, E

add F, F, D

Instrucciones: 4.

Tamaño del programa en memoria: 16 bytes.

Accesos: 16 MI y 12 MD.

**(b)** *Realizar una tabla de comparación similar a la del Ejercicio 2.*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Máquina de 1 dirección** | **Máquina de 2 direcciones** | **Máquina de 3 direcciones** |
| Tamaño del programa en memoria (código de operación más operandos) | |  |  | | --- | --- | | load A | 1 + 1= 2 | | sub B | 1 + 1= 2 | | mul C | 1 + 1= 2 | | store A | 1 + 1= 2 | | load D | 1 + 1= 2 | | div E | 1 + 1= 2 | | add A | 1 + 1= 2 | | store F | 1 + 1= 2 | | **total** | **16 bytes** | | |  |  | | --- | --- | | mov F, A | 1 + 2= 3 | | sub F, B | 1 + 2= 3 | | mul F, C | 1 + 2= 3 | | div D, E | 1 + 2= 3 | | add F, D | 1 + 2= 3 | | **total** | **15 bytes** | | |  |  | | --- | --- | | sub F, A, B | 1 + 3= 4 | | mul F, F, C | 1 + 3= 4 | | div D, D, E | 1 + 3= 4 | | add F, F, D | 1 + 3= 4 | | **total** | **16 bytes** | |
| Cantidad de accesos a memoria (instrucciones más operandos) | |  |  | | --- | --- | | load A | 2 + 1= 3 | | sub B | 2 + 1= 3 | | mul C | 2 + 1= 3 | | store A | 2 + 1= 3 | | load D | 2 + 1= 3 | | div E | 2 + 1= 3 | | add A | 2 + 1= 3 | | store F | 2 + 1= 3 | | **total** | **24 bytes** | | |  |  | | --- | --- | | mov F, A | 3 + 2= 5 | | sub F, B | 3 + 3= 6 | | mul F, C | 3 + 3= 6 | | div D, E | 3 + 3= 6 | | add F, D | 3 + 3= 6 | | **total** | **29 bytes** | | |  |  | | --- | --- | | sub F, A, B | 4 + 3= 7 | | mul F, F, C | 4 + 3= 7 | | div D, D, E | 4 + 3= 7 | | add F, F, D | 4 + 3= 7 | | **total** | **28 bytes** | |

**(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= 1ms)? ¿Es ésta una conclusión general?*

La máquina que elegiría haciendo un balance de la cantidad de instrucciones, el espacio en memoria ocupado y el tiempo de ejecución (1 acceso a memoria= 1ms) es la de 1 dirección. Sin embargo, ésta no es una conclusión general, ya que depende del tamaño del bus de datos respecto al bus de direcciones. Cuanto más grande es el primero respecto al segundo, más eficiente es migrar hacia máquinas de más direcciones.

**Ejercicio 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.*

*ORG 1000h*

*NUM0 DB 0CAh*

*NUM1 DB 0*

*NUM2 DW ?*

*NUM3 DW 0ABCDh*

*NUM4 DW ?*

*END*

*ORG 2000H*

*MOV BL, NUM0*

*MOV BH, 0FFh*

*MOV CH, BL*

*MOV AX, BX*

*MOV NUM1, AL*

*MOV NUM2, 1234h*

*MOV BX, OFFSET NUM3*

*MOV DL, [BX]*

*MOV AX, [BX]*

*MOV BX, 1006h*

*MOV WORD PTR [BX], 0CDEFh*

*HLT*

*END*

**(a)** *Explicar, detalladamente, qué hace cada instrucción MOV del programa anterior, en función de sus operandos y su modo de direccionamiento.*

MOV BL, NUM0: Con un modo de direccionamiento directo, copia el contenido de la variable NUM0 (CAh, 202 en decimal) a la parte baja del registro B.

MOV BH, 0FFh: Con un modo de direccionamiento inmediato, copia el valor FFh (255 en decimal) a la parte alta del registro B.

MOV CH, BL: Con un modo de direccionamiento por registro, copia el contenido de la parte baja del registro B (CAh, 202 en decimal) a la parte alta del registro C.

MOV AX, BX: Con un modo de direccionamiento por registro, copia el contenido del registro B (FFCAh, 65482 en decimal) al registro A.

MOV NUM1, AL: Con un modo de direccionamiento directo, copia el contenido de la parte baja del registro A (CAh, 202 en decimal) a la variable NUM1.

MOV NUM2, 1234h: Con un modo de direccionamiento inmediato, copia el valor 1234h (4660 en decimal) a la variable NUM2.

MOV BX, OFFSET NUM3: Con un modo de direccionamiento inmediato, copia la posición de memoria de NUM3 (1004h) al registro BX.

MOV DL, [BX]: Con un modo de direccionamiento indirecto por registro, copia el contenido que haya en la posición de memoria contenida en el registro B (CDh) a la parte baja del registro D.

MOV AX, [BX]: Con un modo de direccionamiento indirecto por registro, copia el contenido que haya en la posición de memoria contenida en el registro B (ABCDh) al registro A.

MOV BX, 1006h: Con un modo de direccionamiento inmediato, copia el valor 1006h (4102 en decimal) al registro B.

MOV WORD PTR [BX], 0CDEFh: Con un modo de direccionamiento indirecto por registro, copia el valor CDEFh (52719 en decimal) a la posición de memoria que apunta el registro B (1006h).

**(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.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrucción** | **Modo de direccionamiento** | **AL** | **AH** | **BL** | **BH** | **CL** | **CH** | **DL** | **DH** |
| MOV BL, NUM0 | Directo |  |  | CA |  |  |  |  |  |
| MOV BH, 0FFh | Inmediato |  |  | CA | FF |  |  |  |  |
| MOV CH, BL | Por registro |  |  | CA | FF |  | CA |  |  |
| MOV AX, BX | Por registro | CA | FF | CA | FF |  | CA |  |  |
| MOV NUM1, AL | Directo | CA | FF | CA | FF |  | CA |  |  |
| MOV NUM2, 1234h | Inmediato | CA | FF | CA | FF |  | CA |  |  |
| MOV BX, OFFSET NUM3 | Inmediato | CA | FF | 04 | 10 |  | CA |  |  |
| MOV DL, [BX] | Indirecto por registro | CA | FF | 04 | 10 |  | CA | CD |  |
| MOV AX, [BX] | Indirecto por registro | CD | AB | 04 | 10 |  | CA | CD |  |
| MOV BX, 1006h | Inmediato | CD | AB | 06 | 10 |  | CA | CD |  |
| MOV WORD PTR [BX], 0CDEFh | Indirecto por registro | CD | AB | 06 | 10 |  | CA | CD |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrucción** | **Modo de direccionamiento** | **NUM0 (1000)** | **NUM1 (1001)** | **NUM2 L (1002)** | **NUM2 H (1003)** | **NUM3 L (1004)** | **NUM3 H (1005)** | **NUM4 L (1006)** | **NUM4 H (1007)** |
| MOV BL, NUM0 | Directo | CA | 00 |  |  | CD | AB |  |  |
| MOV BH, 0FFh | Inmediato | CA | 00 |  |  | CD | AB |  |  |
| MOV CH, BL | Por registro | CA | 00 |  |  | CD | AB |  |  |
| MOV AX, BX | Por registro | CA | 00 |  |  | CD | AB |  |  |
| MOV NUM1, AL | Directo | CA | CA |  |  | CD | AB |  |  |
| MOV NUM2, 1234h | Inmediato | CA | CA | 34 | 12 | CD | AB |  |  |
| MOV BX, OFFSET NUM3 | Inmediato | CA | CA | 34 | 12 | CD | AB |  |  |
| MOV DL, [BX] | Indirecto por registro | CA | CA | 34 | 12 | CD | AB |  |  |
| MOV AX, [BX] | Indirecto por registro | CA | CA | 34 | 12 | CD | AB |  |  |
| MOV BX, 1006h | Inmediato | CA | CA | 34 | 12 | CD | AB |  |  |
| MOV WORD PTR [BX], 0CDEFh | Indirecto por registro | CA | CA | 34 | 12 | CD | AB | EF | CD |

**(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, qué función cumplen estos registros.*

El registro temporal denominado “ri” cumple la función de guardar, temporalmente, la dirección de la variable (fuente o destino, dependiendo del caso). El registro temporal denominado “id” cumple la función de guardar temporalmente el valor que se quiere copiar en la variable.

**Ejercicio 5.**

*El siguiente programa utiliza diferentes instrucciones de procesamiento de datos (instrucciones aritméticas y lógicas). Analizar el comportamiento de ellas y ejecutar el programa en el MSX88.*

*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*

**(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 o 0) de cada uno de ellos. ¿Dan alguna indicación acerca de la correctitud de los resultados?*

En la operación ADD, se suma 80h (128 en decimal) más 80h (128 en decimal), lo que debería dar 100h (256 en decimal). El estado de los flags después de la ejecución de la instrucción ADD es Z= 1 (resultado cero), N= 0 (resultado no negativo), C= 1 (resultado incorrecto en BSS), V= 1 (resultado incorrecto en Ca2).

En la operación SUB, se resta 200 menos 201, lo que debería dar -1. El estado de los flags después de la ejecución de la instrucción SUB es Z= 0 (resultado no cero), N= 1 (resultado negativo), C= 1 (resultado incorrecto en BSS), V= 0 (resultado correcto en Ca2).

La correctitud de las operaciones ADD y SUB depende:

* en BSS, de la bandera C (carry); si se tiene C= 1, el resultado va a ser incorrecto; y
* en Ca2, de la bandera V (overflow); si se tiene V= 1, el resultado va a ser incorrecto.

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

Las cadenas binarias que representan NUM1 y NUM2 en la memoria del simulador (al finalizar la ejecución del programa) son 11001001b - C9h (201 en decimal) y 11111111b - FFh (-1 en decimal), respectivamente. Estos valores están expresados en BSS y Ca2, respectivamente.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operación** | **Operando 1** | **Operando 2** | **Dirección 1** | **Dirección 2** | **Resultado** |
| ADD AL, AL | 80h | 80h | AL | AL | 00h (Z= 1, C= 1, V= 1) |
| INC NUM1 | 200 (C8h) |  | 1001h |  | 201 (C9h) |
| DEC BL | 201 (C9h) |  | BL |  | 200 (C8h) |
| SUB BL, BH | 200 (C8h) | 201 (C9h) | BL | BH | -1 (FF) (N= 1, C= 1) |
| AND CH, BYTE0 | 10101010b (AAh) | 01111111b (7Fh) | CH | 1003h | 00101010b (2Ah) |
| NOT BYTE0 | 01111111b (7Fh) |  | 1003h |  | 10000000b (80h) |
| OR CH, BYTE0 | 00101010b (2Ah) | 10000000b (80h) | CH | 1003h | 10101010b (AAh) |
| XOR CH, 11111111B | 10101010b (AAh) | 11111111b (FFh) | CH |  | 01010101b (55h) |

**Ejercicio 6.**

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

*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*

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

El lazo se ejecuta 15 veces. En el caso general, esto depende de las variables INI y FIN.

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

* JS: 15.
* JZ: 1.
* JMP: no tiene contenido final (lazo infinito).

**Ejercicio 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. Determinar las modificaciones que debería hacer al programa si la condición de la sentencia IF fuera:*

ORG 2000H

CMP AL, BL

JS IF

MOV CL, BL

JMP FIN

IF: MOV CL, AL

FIN: HLT

END

**(a)** *A B.*

Opción 1:

ORG 2000H

CMP AL, BL

JS IF

JZ IF

MOV CL, BL

JMP FIN

IF: MOV CL, AL

FIN: HLT

END

Opción 2:

ORG 2000H

CMP AL, BL

JNS AUX

AUX: JNZ ELSE

MOV CL, AL

JMP FIN

ELSE: MOV CL, BL

FIN: HLT

END

**(b)** *A= B.*

Opción 1:

ORG 2000H

CMP AL, BL

JZ IF

MOV CL, BL

JMP FIN

IF: MOV CL, AL

FIN: HLT

END

Opción 2:

ORG 2000H

CMP AL, BL

JNZ ELSE

MOV CL, AL

JMP FIN

ELSE: MOV CL, BL

FIN: HLT

END

**Ejercicio 8.**

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

*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*

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

El resultado de la suma es 110.

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

MOV TOTAL, AL

HLT

END

**Ejercicio 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.*

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 AH, MAX

MOV CL, OFFSET FIN - OFFSET TABLA

MOV BX, OFFSET TABLA

LAZO: CMP AH, [BX]

INC BX

JS MAYOR

INC AL

MAYOR: DEC CL

JNZ LAZO

MOV TOTAL, AL

HLT

END

**Ejercicio 10.**

*Analizar el funcionamiento del siguiente programa:*

*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*

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

Los elementos entre sí están relacionados de manera tal que cada valor siguiente será el doble del valor anterior, comenzando la tabla en el valor 1.

**(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)?*

La tabla se crea a partir de la dirección de memoria 1000h (4096 en decimal) y la longitud de cada uno de sus elementos es de 16 bits (DW).

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

La tabla, una vez finalizada la ejecución del programa, tiene 8 elementos (1, 2, 4, 8, 16, 32, 64, 128).

**Ejercicio 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.*

ORG 1000H

MAX DB 50

DIR DB ?

ORG 2000H

MOV AL, 0

MOV AH, MAX

MOV BX, OFFSET DIR

MULT: MOV [BX], AL

INC BX

ADD AL, 5

CMP AL, AH

JS MULT

HLT

END

**Ejercicio 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. Por ejemplo: si X= 3, entonces, el arreglo tendrá 8, 1, 6, -1, 4, -3, 2, -5, 0.*

Opción 1:

ORG 1000H

NUM DB 20

TABLA DB ?

ORG 2000H

MOV AL, NUM

MOV BX, OFFSET TABLA - 2

INC BX

LAZO: JZ FIN

INC BX

MOV AH, AL

AND AH, 1

JZ PAR

ADD AL, 5

MOV [BX], AL

JMP LAZO

PAR: SUB AL, 7

MOV [BX], AL

JMP LAZO

FIN: HLT

END

ORG 1000H

NUM DB 20

TABLA DB ?

END

Opción 2:

ORG 1000H

NUM DB 20

TABLA DB ?

ORG 2000H

MOV AL, NUM

MOV BX, OFFSET TABLA - 2

INC BX

LAZO: JZ FIN

INC BX

MOV AH, AL

AND AH, 1

JNZ IMPAR

SUB AL, 7

MOV [BX], AL

JMP LAZO

IMPAR: ADD AL, 5

MOV [BX], AL

JMP LAZO

FIN: HLT

END

**Ejercicio 13.**

*Dada la frase “Organización y la Computación”, almacenada en la memoria, escribir un programa que determine cuántas letras “a” seguidas de “c” hay en ella.*

ORG 1000H

AC DB ?

INICIO DB “Organización y la Computación”

FINAL DB ?

ORG 2000H

MOV AL, 0

MOV BX, OFFSET INICIO - 1

MOV CX, OFFSET FINAL

LAZO: INC BX

CMP CX, BX

JZ FIN

MOV DX, [BX]

CMP DX, 9799h

JZ SUMA

JMP LAZO

SUMA: INC AL

JMP LAZO

FIN: MOV AC, AL

HLT

END

**Ejercicio 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). Verificar el resultado final y almacenar 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.*

ORG 1000H

NUM1 DW 0AAFFH, 0BBFFH

NUM2 DW 0H, 1

BIEN DB ?

MAL DB ?

RESUL DW ?

ORG 2000H

MOV BX, OFFSET NUM1 + 2

MOV AX, [BX]

MOV BX, OFFSET NUM2 + 2

ADD AX, [BX]

MOV BX, OFFSET RESULT + 2

MOV [BX], AX

MOV AX, NUM1

ADC AX, NUM2

MOV RESUL, AX

JO M

JC M

MOV BIEN, 0FFH

JMP FIN

M: MOV MAL, 0FFH

FIN: HLT

END

**Ejercicio 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.*

Sin errores de tipo aritmético:

ORG 1000H

TAB1 DW 1,2,3,4,5,6,7,8,9,10,11,12

TAB2 DW 13,14,15,16,17,18,19,20,21,22,23,24

TAB3 DW 12 DUP (?)

CANT DB 12

DIR3 DW ?

ORG 2000H

MOV AX, OFFSET TAB1

MOV CX, OFFSET TAB2

MOV DIR3, OFFSET TAB3

LAZO: MOV BX, AX

MOV DX, [BX]

MOV BX, CX

ADD DX, [BX]

MOV BX, DIR3

MOV [BX], DX

ADD AX, 2

ADD CX, 2

ADD DIR3, 2

DEC CANT

JNZ LAZO

HLT

END

Con errores de tipo aritmético:

ORG 1000H

TAB1 DW 1,2,3,4,5,6,7,8,9,10,11,12

TAB2 DW 13,14,15,16,17,18,19,20,21,22,23,24

TAB3 DW 12 DUP (?)

CANT DB 6

DIR3 DW ?

ORG 2000H

MOV AX, OFFSET TAB1

MOV CX, OFFSET TAB2

MOV DIR3, OFFSET TAB3

LAZO: MOV BX, AX

MOV DX, [BX]

MOV BX, CX

ADD DX, [BX]

PUSHF

MOV BX, DIR3

MOV [BX], DX

ADD AX, 2

ADD CX, 2

ADD DIR3, 2

MOV BX, AX

MOV DX, [BX]

MOV BX, CX

POPF

ADC DX, [BX]

MOV BX, DIR3

MOV [BX], DX

ADD AX, 2

ADD CX, 2

ADD DIR3, 2

DEC CANT

JNZ LAZO

HLT

END

**Ejercicio 16.**

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

*Programa 1:*

*ORG 1000H*

*NUM1 DB 5H*

*NUM2 DB 3H*

*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*

*Programa 2:*

*ORG 1000H*

*NUM1 DB 5H*

*NUM2 DB 3H*

*ORG 3000H*

*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*

*ORG200H*

*MOV AL, NUM1*

*MOV CL, NUM2*

*CALL SUB1*

*HLT*

*END*

**(a)** *¿Cuál es la tarea realizada por ambos programas?*

La tarea realizada por ambos programas es multiplicar NUM1 por NUM2.

**(b)** *¿Dónde queda almacenado el resultado?*

El resultado queda almacenado en el registro DX.

**(c)** *¿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?*

El programa que realiza la tarea más rápido es el primero, ya que hace menos llamados a memoria que el segundo. Por otra parte, el tiempo de ejecución de la tarea depende de los valores almacenados en NUM2.

**(d)** *Explicar, detalladamente, todas las acciones que tienen lugar al ejecutarse la instrucción CALL SUB1.*

Al ejecutarse la instrucción CALL SUB1, se guarda el valor de la posición de memoria que está en el puntero de instrucción (IP) en la pila (PUSH del IP), se asigna el valor de la posición de memoria correspondiente a la etiqueta SUB1 al IP y la CPU comienza a ejecutar las instrucciones de la subrutina SUB1.

**(e)** *¿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?*

La operación que se realiza con la instrucción RET es retornar al programa principal a partir de la instrucción siguiente a la instrucción CALL SUB1. La CPU sabe a qué dirección de memoria debe retornar desde la subrutina al programa principal porque el puntero de instrucción (IP) se carga con el valor de la posición de memoria guardada en la pila (POP del IP) y, por lo tanto, la ejecución del programa sigue a partir de la instrucción siguiente a la instrucción CALL SUB1.

**Ejercicio 17.**

*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. Explicar detalladamente:*

*ORG 1000H*

*NUM1 DW 5H*

*NUM2 DW 3H*

*ORG 3000H*

*SUB2: MOV DX, 0*

*LAZO: MOV BX, AX*

*ADD DX, [BX]*

*PUSH DX*

*MOV BX, CX*

*MOV DX, [BX]*

*DEC DX*

*MOV [BX], DX*

*POP DX*

*JNZ LAZO*

*RET*

*ORG 2000H*

*MOV AX, OFFSET NUM1*

*MOV CX, OFFSET NUM2*

*CALL SUB2*

*HLT*

*END*

**(a)** *Todas las acciones que tienen lugar al ejecutarse las instrucciones PUSH DX y POP DX.*

Al ejecutarse la instrucción PUSH DX, se resta dos al valor apuntado en el registro SP y se guarda el valor de DX en la pila. Al ejecutarse la instrucción POP DX, se suma dos al valor apuntado en el registro SP y se carga el valor almacenado en la pila en DX.

**(b)** *Cuáles son los dos usos que tiene el registro DX en la subrutina SUB2.*

Los dos usos que tiene el registro DX en la subrutina SUB2 son de acumulador y de decrementador.

**Ejercicio 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 el Ejercicio 14).*

ORG 1000H

TAB1 DW 1,2,3,4,5,6,7,8,9,10,11,12

TAB2 DW 13,14,15,16,17,18,19,20,21,22,23,24

TAB3 DW 12 DUP (?)

CANT DB 6

DIR3 DW ?

ORG 3000H

SUMA: MOV BX, AX

MOV DX, [BX]

MOV BX, CX

ADD DX, [BX]

PUSHF

MOV BX, DIR3

MOV [BX], DX

ADD AX, 2

ADD CX, 2

ADD DIR3, 2

MOV [BX], AX

MOV DX, [BX]

MOV BX, CX

POPF

ADC DX, [BX]

MOV BX, DIR3

MOV [BX], DX

ADD AX, 2

ADD CX, 2

ADD DIR3, 2

DEC CANT

JNZ SUMA

RET

ORG 2000H

MOV AX, OFFSET TAB1

MOV CX, OFFSET TAB2

MOV DIR3, OFFSET TAB3

CALL SUMA

HLT

END

**Ejercicio 19.**

*Escribir 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.*

ORG 1000H

MANTISA DB 0AAh

EXPONENTE DB 5

ORG 3000H

AJUSTAR: CMP AL, 0

JZ NOPOSIBLE

DEC AL

ADD AH, AH

JC NOAJUSTA

MOV BL, 00H

JMP FIN

NOPOSIBLE: MOV BL, 0FFH

FIN: RET

ORG 2000H

MOV AH, MANTISA

MOV AL, EXPONENTE

CALL AJUSTAR

HLT

END

**Ejercicio 20.**

*Escribir 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.*

ORG 1000H

MANTISA DB 0AAH, 0BBH, 0CCH

EXPONENTE DB 00AH

ORG 3000H

ANALIZAMANTISA: MOV DH, [BX]

CMP DH, 128

JNZ MANTISA\_NOCERO

LAZO: MOV DH, [BX]

CMP DH, 0

JNZ MANTISA\_NOCERO

INC BX

CMP BX, OFFSET EXPONENTE

JZ MANTISA\_CERO

JMP LAZO

MANTISA\_CERO: MOV DL, 0

JMP FIN\_MANTISA

MANTISA\_NOCERO: MOV DL, 1

FIN\_MANTISA: RET

NORMALIZADO: CALL ANALIZAMANTISA

CMP AL, 0

JZ SINNORMALIZAR

CMP AL, 255

JZ NAN

MOV CL, 4

JMP FIN\_EXPONENTE

SINNORMALIZAR: CMP DL, 0

JZ CERO

MOV CL, 0

JMP FIN\_EXPONENTE

INFINITO: MOV CL, 1

JMP FIN\_EXPONENTE

NAN: CMP DL, 0

JZ INFINITO

MOV CL, 2

JMP FIN\_EXPONENTE

CERO: MOV CL, 3

FIN\_EXPONENTE: RET

ORG 2000H

MOV BX, OFFSET MANTISA

MOV AL, EXPONENTE

CALL NORMALIZADO

HLT

END

**Ejercicio 21.**

*Modificar la subrutina del Ejercicio 19 para el caso en que la mantisa y el exponente estén representados en BCS.*

ORG 1000H

MANTISA DB 0AAh

EXPONENTE DB 5

ORG 3000H

AJUSTAR: DEC AL

ADD AH, AH

JC NOPOSIBLE1

JO NOPOSIBLE2

POSIBLE: MOV BL, 00H

JMP FIN

NOPOSIBLE1: JO POSIBLE

MOV BL, 0FFH

JMP FIN

NOPOSIBLE2: MOV BL, 0FFH

FIN: RET

ORG 2000H

MOV AH, MANTISA

MOV AL, EXPONENTE

CALL AJUSTAR

HLT

END