Práctica 6

Programando

Objetivos de la práctica: que el alumno

- ***Realice el diseño de programas utilizando instrucciones del MSX88.***
- ***Comprenda la utilidad y funcionamiento de las subrutinas.***

Bibliografía:

- ***Apunte 4 de la cátedra, "Lenguaje Assembler".***
- ***Manual del simulador MSX88.***
- ***Set de Instrucciones de MSX88.***

--------- *con algunos ejercicios resueltos*

Para cada programa propuesto, deberá editar el archivo fuente con extensión **asm** (ej: ejer1.asm), 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.

1. 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

2. 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).

3. 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?

4. El siguiente programa es otra forma de implementación de la tarea del punto anterior (ejercicio 3). 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.

5. Escribir un programa que sume 2 vectores de 6 elementos (similar al realizado en el ejercicio 2), de modo tal que utilice una subrutina que sume números de 32 bits (similar al programa escrito en ejercicio 1). **A RESOLVER**

6. 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: P6Ej6.lst
Programa Fuente en: P6Ej6.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
```

7. 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**

Enlace a video con explicación P6ej7: https://youtube.com/watch?v=IQ-MJETXwTI

```
      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 esta a partir de la direccion 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
      RET
N_a_N: MOV CL, 2
      RET
```

```
        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
```

8. Modifique la subrutina del ejercicio 6 para el caso en que la mantisa y el exponente estén representados en BCS.

A RESOLVER

Datos útiles:

- Las subrutinas siempre se escriben antes que el programa principal, aunque su dirección de comienzo sea más alta.
- Las etiquetas de subrutinas y bucles van seguidas de dos puntos (:).
- Los operandos en hexadecimal terminan en H y los que comienzan con una letra van precedidos por un cero (0) para no ser confundidos con etiquetas (por ejemplo, 0A4H en lugar de A4H).
- Se pueden incluir comentarios en los programas, anteponiendo siempre un punto y coma (;).
- El direccionamiento indirecto solo está implementado con el registro BX.
- Cada celda de memoria almacena un byte. Los datos de dos bytes (words) se almacenan de la siguiente manera: primero la parte baja (byte menos significativo) y luego la parte alta. Esto se corresponde con la idea de que la parte baja del dato se almacena en la dirección más baja y la parte alta, en la dirección más alta.