Práctica **5**

Assembly y Programando con el Simulador MSX88

Organización de Computadoras 2024
Turno RECURSANTES

– Objetivos –

Que el alumno:

- Use las instrucciones del lenguaje assembly del simulador MSX88.
- Domine los diferentes modos de direccionamiento.
- Realice el diseño de programas utilizando instrucciones del MSX88.
- Comprenda la utilidad y funcionamiento de las subrutinas.

– Bibliografía –

- Apunte 4 de la cátedra, “Lenguaje Assembly”.
- Manual del simulador MSX88.
- Set de Instrucciones de MSX88.

Para cada programa propuesto, deberá editar el archivo fuente y guardarlo con extensión `ASM` (ej: `ejer1.asm`), luego ensamblarlo usando el comando `asm88.exe` (haciendo: `asm88 ejer1.asm`), que producirá en su salida un archivo con extensión O (ej: `ejer1.o`), el cual deberá ser enlazarlo con el comando `link88.exe` (haciendo: `link88 ejer1.o`) para obtener finalmente un archivo con extensión `EJE` (ej: `ejer1.eje`), que podrá ser cargado y ejecutado dentro del simulador MSX88.

1) El Programa 1 utiliza una **instrucción de transferencia de datos** (instrucción `MOV`) con diferentes modos de direccionamiento para hacer referencia a sus operandos. Ejecutar y analizar 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 los REGISTROS, de las posiciones de MEMORIA accedidas, de las operaciones en la ALU, etc...
   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.
2) El Programa 2 utiliza diferentes **instrucciones de procesamiento de datos** (instrucciones aritméticas y lógicas). Analice y ejecute el comportamiento de ellas en el MSX88.
   a) ¿Cuál es el estado de los FLAGS después de la ejecución de las instrucciones `ADD` y `SUB` del programa anterior? Justificar el estado de cada uno de ellos (¿por qué quedaron en 0 ó en 1?). ¿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 o lógica del programa, el valor de sus operandos, en qué registro o dirección de memoria se almacenan y el resultado obtenido luego de realizar cada operación.
3) El Programa 3 implementa un contador utilizando una **instrucción de transferencia de control**. Analice el funcionamiento de cada instrucción, poniendo atención en las del lazo repetitivo que provoca la cuenta.
   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`:
      1. `JS`
      2. `JZ`
      3. `JMP`

4) 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

b) A = B

5) Analizar el funcionamiento del Programa 4 .
   a) ¿El programa genera una tabla con las potencias de 2? Es decir, 2^0, 2^1, 2^2,...
   b) ¿A partir de qué dirección de memoria comienza la tabla? ¿Para qué sirve DUP en esa definición?
   c) ¿Cuál es la longitud de cada uno de sus elementos (medida en bits)?
   d) ¿Cuántos elementos tiene la tabla una vez finalizada la ejecución del programa?
   e) ¿De qué depende esta cantidad?

6) El Programa 5 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 ese resultado ejecutando el programa en el MSX88. ¿Qué modificaciones deberá hacer en el programa para que el mismo almacene el resultado de la suma en la celda etiquetada TOTAL?

7) Escribir un programa que, utilizando las mismas variables y datos que el Programa 5 (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.

8) 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.

9) Escribir un programa que sume en BSS dos números de 32 bits almacenados en memoria de datos (etiquetados como NUM1 y NUM2) y guarde el resultado en RESUL. Tanto NUM1 como NUM2 y RESUL necesitarán cada uno 4 bytes consecutivos en memoria, definidos como dos WORD, de esta manera:

```
NUM1    DW  05678H, 01234H ; De esta forma se representa en NUM1 el valor 012345678H.
                           ; NUM1 apunta a la parte baja y NUM1+2, a la parte alta.
```

Verificar el resultado final y en caso de ser correcto, almacene 0FFH en una celda etiquetada BIEN o, en caso de no serlo, almacene 0FFH en otra celda etiquetada MAL.

10) Escribir un programa que efectúe la suma de dos vectores de 6 elementos cada uno (donde cada elemento es un número de 16 bits) almacenados en memoria de datos y etiquetados TAB1 y TAB2, guardando el resultado de la suma 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.

11) El Programa 6 y el Programa 7 realizan la misma tarea, pero en uno de ellos se utiliza una **instrucción de transferencia de control con retorno**. Analizar y comprobar que son equivalentes.
   a) ¿Cuál es la tarea realizada por ambos programas? ¿Dónde queda almacenado el resultado?
   b) ¿Cuál programa realiza la tarea más rápido? ¿El tiempo de ejecución de la tarea depende de los valores asignados a NUM1, a NUM2, a ambos o a ninguno?
   c) Explicar detalladamente todas las acciones que tienen lugar al ejecutarse la instrucción CALL SUB1.
   d) Describir en detalle los pasos que realiza la instrucción RET.
   e) ¿Cómo sabe la CPU a qué dirección de memoria debe retornar al ejecutar la instrucción RET?

12) El Programa 8 implementa la misma tarea que los programas del punto 11). Indicar las diferencias con las rutinas anteriores, en particular lo relacionado con la forma de 'proveer' los operandos a la subrutina.
   a) Explicar detalladamente 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?
   c) ¿Qué efecto consiguen el par de instrucciones "ADD DL,[BX]" y "ADC DH,0" sobre el registro DX?

```
        ORG 1000H
NUM0    DB  0CAH
NUM1    DB  0
NUM2    DW  ?
NUM3    DW  0ABCDH
NUM4    DW  ?
        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], 1006H
        HLT
END
```

Programa 1

```
        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
```

Programa 2

```
        ORG 1000H
INI     DB  0
FIN     DB  15
        ORG 2000H
        MOV AL, INI
        MOV AH, FIN
SUMA:   INC AL
        CMP AL, AH
        JNZ SUMA
        HLT
END
```

Programa 3

```
        ORG 1000H
IABLA   DW  500 DUP (?)
        ORG 2000H
        MOV AX, 1
        MOV BX, OFFSET IABLA
CARGA:  MOV [BX], AX
        ADD BX, 2
        ADD AX, AX
        CMP AX, 200
        JS  CARGA
        HLT
END
```

Programa 4

```
        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
```

Programa 5

```
        ORG 1000H
NUM1    DB  150
NUM2    DB  10
        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 6

```
        ORG 1000H
NUM1    DB  150
NUM2    DB  10
RESUL   DW  ?
; Subrutina SUB1
        ORG 3000H
SUB1:   MOV DX, 0
        CMP AL, 0
        JZ  FIN
        CMP CL, 0
        JZ  FIN
        MOV AH, 0
LAZO:   ADD DX, AX
        DEC CX
        JNZ LAZO
FIN:    RET
; Programa principal
        ORG 2000H
        MOV  AL, NUM1
        MOV  CL, NUM2
        CALL SUB1
        MOV  RESUL, DX
        HLT
END
```

Programa 7

```
        ORG 1000H
NUM1    DB  150
NUM2    DB  10
RESUL   DW  ?
; Subrutina SUB2
        ORG 3000H
SUB2:   MOV  DX, 0
        MOV  BX, AX
        CMP  BYTE PTR [BX], 0
        JZ   FIN
        MOV  BX, CX
        CMP  BYTE PTR [BX], 0
        JZ   FIN
LAZO:   MOV  BX, AX
        ADD  DL, [BX]
        ADC  DH, 0
        PUSH DX
        MOV  BX, CX
        MOV  DL, [BX]
        DEC  DL
        MOV  [BX], DL
        POP  DX
        JNZ  LAZO
FIN:    RET
; Programa principal
        ORG 2000H
        MOV  AX, OFFSET NUM1
        MOV  CX, OFFSET NUM2
        CALL SUB2
        MOV  RESUL, DX
        HLT
END
```

Programa 8

Datos útiles:

- Las subrutinas siempre se deben escribir 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 está bien pero A4H representa una etiqueta llamada A4H, no el valor hexadecimal).
- Se pueden incluir comentarios en los programas, anteponiendo siempre un punto y coma (;) y se toma como comentario todo el texto que haya después del punto y coma hasta llegar al final de esa línea.
- El direccionamiento indirecto solo está implementado con el registro BX.
- Cada celda de memoria almacena un byte. Los datos de tipo WORD (16 bits o 2 bytes) se almacenan así: Se usa un byte para la parte baja del valor de 16 bits (los 8 bits menos significativos del valor) y otro byte para la parte alta del valor de 16 bits (los 8 bits más significativos del valor). El byte menos significativo está en la dirección asociada al valor de 16 bits y el más significativo se encuentra en la dirección siguiente al primero. 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 (conocido como *little-endian*).