Organización de Computadoras 2020

Práctica 5

Assembly, Instrucciones y Simulador MSX88

Objetivos de la práctica: que el alumno domine
- ***Las instrucciones básicas del lenguaje assembly del simulador MSX88.***
- ***Los diferentes modos de direccionamiento.***
- ***El diseño de programas utilizando instrucciones de salto condicional.***

Bibliografía:
- ***Apunte 4 de la cátedra, "Lenguaje Assembly".***
- ***Manual del simulador MSX88.***
- ***Set de Instrucciones de 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`	`Mov F, A`	`add F, A, B`
`add B`	`add F, B`	`div F, F, C`
`div C`	`div F, C`	`sub F, F, D`
`sub D`	`sub F, D`	
`store F`		

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?

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.