Directivas, etiquetas y funciones en ensamblador SISA

En estas hojas se presentan algunas ayudas que nos ofrece el lenguaje ensamblador del SISA para escribir programas.

- Secciones

En un programa escrito en lenguaje ensamblador se diferencia entre instrucciones y datos. Las instrucciones y los datos se agrupan en diferentes secciones. En un programa puede haber varias secciones de datos y varias de instrucciones (también llamadas secciones de texto).

- Sección de datos. Es la zona donde se declaran las variables y se inicializan o se reserva espacio de memoria para su contenido tal como se detalla más adelante. Esta parte del programa ha de comenzar con la directiva **`.data`**.
- Sección de instrucciones**.** Aquí es donde se escriben las instrucciones del programa en lenguaje ensamblador SISA. Una sección de instrucciones comienza con la directiva **`.text`**.

Cada sección termina donde comienza otra sección. La última sección termina con la directiva **`.end`**, que le indica al programa ensamblador (traductor de lenguaje ensamblador a lenguaje máquina) que termina la sección y el fichero del programa. El programa ensamblador no reconoce ninguna directiva ni instrucción que se encuentre después de la directiva **`.end`**.

Las instrucciones del lenguaje ensamblador pueden estar escritas en mayúsculas o en minúsculas indistintamente.

En una sentencia ensamblador se pueden introducir comentarios. Esto se hace poniendo un "`;`" y seguidamente el comentario que se desee. Todo lo que haya en la línea después del "`;`" será interpretado como un comentario y no será traducido.

Además del repertorio de instrucciones, el lenguaje ensamblador SISA consta de los siguientes elementos:

- Etiquetas:

Son cadenas alfanuméricas seguidas del carácter '`:`' que se escriben al comienzo de una línea de la parte de programa (también puede ser de datos, como se explica después). La etiqueta permite identificar la dirección de memoria donde se ubicará lo que se traduzca a continuación. Por eso a una etiqueta también la denominamos dirección simbólica de memoria. Por ejemplo, supongamos que el siguiente fragmento de código se debe ubicar, una vez traducido a LM, a partir de la dirección de memoria 0x5D3A:

```
            LD R1, 0(R3)
            BZ R1, etiq
            ADD R2, R0, R1
etiq:       AND R1, R2, R3
```

El valor de `etiq` es 0x5D40 = 0x5D3A + 3*2, ya que cada instrucción es un Word, 2 bytes. La sentencia `BZ R1, etiq` se traduce a lenguaje máquina por el vector de bits: `1000 001 0 00000001`, ya que el valor que hay que codificar en los 8 bits de menor peso de la instrucción es el 1. Esto es así para que en tiempo de ejecución se sume 1*2 al PC actualizado de la instrucción de salto. El PC de la instrucción de salto es 0x5D3C =0x5D3A+2 (la dirección de la primera instrucción más 2 bytes que ocupa esta instrucción ya que la de salto es la segunda del fragmento de programa). El PC actualizado es el PC de la instrucción +2: apunta a la siguiente instrucción después del salto, a la dirección 0x5D3C+2= 0x5D3E. De esta forma al hacerse, en tiempo de ejecución de la instrucción de salto, PC=PC+2+SE(N8)*2, se obtendrá el valor de `etiq` (0x5D3C + 2 +1*2 = 0x5D40). En hexadecimal la instrucción se escribe como 0x8201.

Otro ejemplo, supongamos que el siguiente fragmento de código se debe ubicar, una vez traducido a lenguaje máquina, a partir de la dirección de memoria 0x79A4:

```
            MOVI R3, 0x34
            MOVHI  R3, 0xAF
label:      LD R1, 0(R3)
            ADDI R3, R3, 1
            BZ R1, label
            AND R1, R2, R3
```

El valor de `label` es 0x79A8 = 0x79A4 + 2*2. La sentencia `BZ R1, label` se traduce a lenguaje máquina por el vector de bits: `1000 001 0 11111101`, que en hexadecimal se escribe como 0x82FD. El desplazamiento codificado en la instrucción es -3 ya que la instrucción de salto se encuentra almacenada en la dirección 0x79AC y 0x79AC +2 – 3*2 es igual a 0x79A8, que es el valor de `Label`.

Las etiquetas permiten que mientras se escriben los programas en lenguaje ensamblador no se tenga que conocer en qué direcciones de memoria se ubicarán una vez traducidos a lenguaje máquina.

- Directivas

Son sentencias del lenguaje ensamblador que no se traducen a instrucciones de lenguaje máquina. Estas sentencias dan información al programa ensamblador para que realice la traducción del programa. Permiten definir dónde empieza la sección de datos y la de texto (instrucciones), como ya hemos visto, y permiten además definir el valor numérico de constantes (nombres o cadenas de caracteres alfanuméricos) así como inicializar el contenido de memoria de la sección de datos. En las líneas de la parte de datos también se pueden definir etiquetas (direcciones simbólicas en las que se almacenan los datos). El lenguaje ensamblador SISA, además de las directivas .data, .text y .end, ya comentadas al hablar de secciones, tiene otras directivas:

=

Permite dar nombre (cadena de caracteres alfanuméricos) a un valor constante. Por ejemplo,

```
Num_elem = 100
```

Permite que se utilice `Num_elem` durante todo el programa para referirse al valor 100. La definición de una constante no provoca uso alguno de memoria. Las constantes (simbólicas) se pueden usar en expresiones cuyo valor se puede calcular en tiempo de ensamblado.

Otra forma alternativa de hacer lo mismo es mediante la directiva **`.set`**

```
.set  nombre, valor
```

haciendo:

```
.set  Num_elem, 100
```

`.space` <***size***>, <***fill***>

Reserva el número de bytes consecutivos de memoria indicado en el parámetro *size* (tamaño) y las inicializa con el valor indicado en el parámetro *fill* (relleno). Cada byte son 8 bits y los almacena en direcciones consecutivas de memoria. El parámetro *fill* es optativo. Si no aparece el parámetro *fill*, el ensamblador rellena el espacio de memoria con ceros. Por ejemplo, si un programa comienza con las directivas:

```
.data
operandos:      .space 1,10
                .space 1,-1
resultados:     .space 2
                .space 1,100
```

Al empezar la ejecución del programa, el contenido de la memoria será el siguiente (si se han ensamblado los datos a partir de la dirección 0 de memoria:

Dirección	Contenido
0x0000	0x0A
0x0001	0xFF
0x0002	0x00
0x0003	0x00
0x0004	0x64
…	…

`.byte` <***fill-1***>, <***fill-2***>, <***fill-3***>, …, <***fill-n***>

Inicializa `n` bytes consecutivos de memoria con los valores *fill-1*, *fill-2*, etc. Puede usarse para `n=1`.

El ejemplo anterior puede escribirse también como:

```
.data
                .byte 10, -1
resultados:     .byte 0, 0, 100
```

`.word` <***fill-1***>, <***fill-2***>, <***fill-3***>, …, <***fill-n***>
Inicializa `n` palabras consecutivas de memoria con los valores *fill-1*, *fill-2*, etc. Cada palabra son 16 bits, dos bytes consecutivos, en formato Little endian (el byte de menor peso en la dirección par y el de mayor peso en la siguiente dirección (par+1=impar). Puede usarse para `n=1`.

El ejemplo anterior puede escribirse también como:

```
.data
                .word 0xFF0A
resultados:     .word 0
                .byte 100
```

`.even`

Esta directiva hace que la dirección de memoria de la siguiente línea del código sea par. Si la dirección donde se encuentra `.even` es impar, poner `.even` es equivalente a poner `.byte` (se reserva un byte para que la siguiente dirección sea par). Si la dirección donde se encuentra `.even` es par su efecto es como si no hubiera ninguna directiva: no hace nada.

Se utiliza para ponerla delante de la definición de una palabra, cuando no se sabe con certeza o se quiere asegurar que la dirección del Word que se define a continuación sea par (ya que en SISA los words, palabras de 2 bytes, están alineados a dirección par).

- Funciones para referirnos a los 8 bits de menor/mayor peso de una etiqueta o constante.

Una etiqueta hace referencia a una dirección de memoria, y por lo tanto su valor requiere 16 bits. Para inicializar un registro con una dirección de memoria, para que el registro haga de puntero a esa posición de memoria, se deben ejecutar dos instrucciones SISA, la primera `MOVI` carga en el registro los 8 bits de menor peso y la segunda `MOVHI` carga los 8 bits de mayor peso. Para indicar, en lenguaje ensamblador, que nos referimos a los 8 bits de menor peso de una dirección (etiqueta), usaremos la función de ensamblador `lo(etiqueta)` y para referirnos a los 8 bits de mayor peso lo haremos con la función `hi(etiqueta)`.

Por ejemplo, si queremos cargar en R3 la dirección de memoria 0xA7F1 tenemos que hacer:

```
     MOVI       R3, 0xF1
     MOVHI      R3, 0xA7
```

No obstante, para hacer algo equivalente a esto en ensamblador, sabiendo que queremos cargar el valor de la etiqueta `tabla` en R3, pero sin saber cuanto vale la etiqueta `tabla`, podemos hacer:

```
MOVI      R3, lo(tabla)
MOVHI     R3, hi(tabla)
```

También pueden usarse las funciones lo y hi para referirnos a los 8 bits de menor y de mayor peso de una constante. Por ejemplo:

```
N = 1000
…
MOVI      R2, lo(N)
MOVHI     R2, hi(N)
```

o también:

```
MOVI      R2, lo(1000)
MOVHI     R2, hi(1000)
```