IngenieríaTécnica en Informática de Gestión. Curso 2001-2002

LABORATORIO DE ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

Sesión 9
Reserva de espacio e instrucciones de carga y almacenamiento

LOS DATOS EN MEMORIA

El primer paso para el desarrollo de un programa en ensamblador es definir los datos en memoria. La programación en ensamblador permite utilizar directivas que facilitan reservar espacio de memoria para datos e inicializarlos a un valor.

En primer lugar recordar que, aunque la unidad base de direccionamiento es el *byte*, las memorias de estos computadores tienen un ancho de 4 bytes o 32 bits, que se llamará palabra o *word*, el mismo ancho que el del bus de datos. Así pues, cualquier acceso a una palabra de memoria supondrá leer cuatro bytes (el byte con la dirección especificada y los tres almacenados en las siguientes posiciones). Las direcciones de palabra deben estar alineadas en posiciones múltiplos de cuatro. Otra posible unidad de acceso a memoria es transferir media palabra (*half-word*).

Declaración de palabras en memoria

Vamos a ver el uso de las directivas `.data` y `.word`. Para ello, en el directorio de trabajo se crea un fichero con la extensión '.s' y con el siguiente contenido:

```
            .data           # comienza zona de datos
palabra1: .word 15          # decimal
palabra2: .word 0x15        # hexadecimal
```

Descripción:

Este programa en ensamblador incluye diversos elementos que se describen a continuación: la directiva `.data [dir]` indica que los elementos que se definen a continuación se almacenarán en la zona de datos y, al no aparecer ninguna dirección como argumento de dicha directiva, la dirección de almacenamiento será la que hay por defecto (`0x10010000`). Las dos sentencias que aparecen a continuación reservan dos números enteros, de tamaño *word*, en dos direcciones de memoria, una a continuación de la otra, con los contenidos especificados.

Ejecuta el programa XSPIM desde el directorio de trabajo y cárgalo mediante el botón *load (*Linux*) o* bien con las opciones *File. Open* (PCSpim). A continuación contesta a las cuestiones siguientes:

Cuestión 1: Encuentra los datos almacenados en memoria en el programa anterior. Localiza dichos datos en el panel de datos e indica su valor en hexadecimal.

Cuestión 2: ¿En qué direcciones se han almacenado dichos datos? ¿Por qué?

Cuestión 3: ¿Qué valores toman las etiquetas `palabra1` y `palabra2`?

Crea otro fichero o modifica el anterior con el siguiente código:

```
                .data 0x10010000   # comienza zona de datos
    palabras:   .word 15,0x15  # decimal/hexadecimal
```

Borra los valores de la memoria mediante el botón *clear* y carga el fichero.

Cuestión 4: Comprueba si hay diferencias respecto al programa anterior.

Cuestión 5: Crea un fichero con un código que defina un vector de cinco palabras (word), que esté asociado a la etiqueta `vector`, que comience en la dirección 0x10000000 y con los valores 0x10, 30, 0x34, 0x20 y 60. Comprueba que se almacena de forma correcta en memoria.

Cuestión 6: ¿Qué ocurre si se quiere que el vector comience en la dirección 0x10000002? ¿En qué dirección comienza realmente? ¿Por qué?

Declaración de bytes en memoria

La directiva `.byte valor` inicializa una posición de memoria, de tamaño byte, con el contenido `valor`.

Crea un fichero con el siguiente código:

```
                .data           # comienza zona de datos
    octeto:     .byte     0x10 # hexadecimal
```

Borra los valores de la memoria mediante el botón *clear* y carga el fichero.

Cuestión 7: ¿Qué dirección de memoria se ha inicializado con el contenido especificado?

Cuestión 8: ¿Qué valor se almacena en la palabra que contiene el byte?

Crea otro fichero o modifica el anterior con el siguiente código:

```
.data

    palabra1: .byte     0x10,0x20,0x30,0x40 # hexadecimal
    palabra2: .word     0x10203040          # hexadecimal
```

Borra los valores de la memoria y carga el fichero.

Cuestión 9: ¿Cuáles son los valores almacenados en memoria?

Cuestión 10: ¿Qué tipo de alineamiento y organización de los datos (Big-endian o Little-endian) utiliza el simulador? ¿Por qué?

Cuestión 11: ¿Qué valores toman las etiquetas `palabra1` y `palabra2`?

Declaración de cadenas de caracteres

La directiva `.ascii "tira"` permite cargar en posiciones de memoria consecutivas, cada una de tamaño byte, el código ASCII de cada uno de los caracteres que componen `"tira"`.

Crea un fichero con el siguiente código:

```
.data
    cadena:   .ascii    "abcde"   # defino string
    octeto:   .byte     0xff
```

Borra los valores de la memoria y carga el fichero.

Cuestión 12: Localiza la cadena anterior en memoria.

Cuestión 13: ¿Qué ocurre si en vez de `.ascii` se emplea la directiva `.asciiz`? Describe lo que hace esta última directiva.

Cuestión 14: Crea otro fichero cargando la misma tira de caracteres a la que apunta la etiqueta `cadena` en memoria, pero ahora utilizando la directiva `.byte`.

Reserva de espacio en memoria

La directiva `.space n` sirve para reservar espacio para una variable en memoria, inicializándola a valor 0.

Crea un fichero con el siguiente código:

```
.data
    palabra1: .word     0x20
    espacio:  .space    8    # reservo espacio
    palabra2: .word     0x30
```

Borra los valores de la memoria y carga el fichero.

Cuestión 15: ¿Qué rango de posiciones se han reservado en memoria para la variable `espacio`?

Cuestión 16: ¿Cuántos bytes se han reservado en total? ¿Y cuántas palabras?

Alineación de datos en memoria

La directiva `.align n` alinea el siguiente dato a una dirección múltiplo de 2^n.

Crea un fichero con el siguiente código:

```
.data
    byte1:    .byte 0x10
    espacio:  .space 4
    byte2:    .byte 0x20
    palabra:  .word 10
```

Cuestión 2.17: ¿Qué rango de posiciones se han reservado en memoria para la variable `espacio`?

Cuestión 2.18: ¿Estos cuatro bytes podrían constituir los bytes de una palabra? ¿Por qué?

Cuestión 2.19: ¿A partir de que dirección se ha inicializado `byte1`? ¿y `byte2`?

Cuestión 2.20: ¿A partir de que dirección se ha inicializado `palabra`? ¿Por qué?

Crea un fichero con el siguiente código:

```
.data
```

```
byte1:    .byte 0x10
          .align 2
espacio:  .space 4
byte2:    .byte 0x20
palabra:  .word 10
```

Cuestión 2.21: ¿Qué rango de posiciones se ha reservado en memoria para la variable `espacio`?

Cuestión 2.22:¿Estos cuatro bytes podrían constituir los bytes de una palabra? ¿Por qué? ¿Qué ha hecho la directiva `.align`?

Problemas propuestos:

1. Diseña un programa ensamblador que reserva espacio para dos vectores A y B de 20 palabras cada uno a partir de la dirección `0x10000000`.
2. Diseña un programa ensamblador que realiza la siguiente reserva de espacio en memoria a partir de la dirección 0x10001000: una palabra, un byte y otra palabra alineada en una dirección múltiplo de 4.
3. Diseña un programa ensamblador que realice la siguiente reserva de espacio e inicialización en memoria a partir de la dirección por defecto: 3 (palabra), 0x10 (byte), reserve 4 bytes a partir de una dirección múltiplo de 4, y 20 (byte).
4. Diseña un programa ensamblador que defina, en el espacio de datos, la siguiente cadena de caracteres: “Esto es un problema” utilizando

a) .ascii
b) .byte
c) .word

5. Sabiendo que un entero se almacena en un word, diseña un programa ensamblador que defina en la memoria de datos la matriz A de enteros definida como

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

a partir de la dirección `0x10010000` suponiendo que:

a) La matriz A se almacena por filas.
b) La matriz A se almacena por columnas.

CARGA Y ALMACENADO DE LOS DATOS

En este tercer capítulo se estudia la carga y almacenamiento de datos entre memoria y los registros del procesador. Dado que la arquitectura del R2000 es RISC, utiliza un subconjunto concreto de instrucciones que permiten las acciones de carga y almacenamiento de los datos entre los registros del procesador y la memoria. Generalmente, las instrucciones de carga de un dato de memoria a registro comienzan con la letra "l" (de *load* en inglés) y las de almacenamiento de registro en memoria con "s" (de *store* en inglés), seguidos por la letra inicial correspondiente al tamaño del dato que se va a mover, *b* para byte, *h* para media palabra (*half word*) y *w* para palabra (*word*).

3.1. Carga de datos inmediatos (constantes)

Crea un fichero con el siguiente código:

```
            .text #zona de instrucciones
main: lui $s0, 0x8690
```

Descripción:

La directiva `.text` sirve para indicar el comienzo de la zona de memoria dedicada a las instrucciones. Por defecto esta zona comienza en la dirección `0x00400000` y en ella se pueden ver las instrucciones que ha introducido el simulador para ejecutar, de forma adecuada, nuestro programa. La primera instrucción se referencia con la etiqueta `main` y le indica al simulador dónde está el principio del programa que debe ejecutar. Por defecto hace referencia a la dirección `0x00400020`. A partir de esta dirección, el simulador cargará el código de nuestro programa en el segmento de memoria de instrucciones.

La instrucción `lui` es la única instrucción de carga inmediata real, y almacena la media palabra que indica el dato inmediato de 16 bits en la parte alta del registro especificado, en este caso `$s0`. La parte baja del registro, correspondiente a los bits de menor peso de éste, se pone a 0.

Borra los valores de la memoria del simulador y carga el fichero anterior.

Cuestión 1: Localiza la instrucción en memoria de instrucciones e indica:

- La dirección donde se encuentra.
- El tamaño que ocupa.
- La instrucción máquina, analizando cada campo de ésta e indicando qué tipo de formato tiene.

Ejecuta el programa mediante el botón *run* del *xspim*.

Cuestión 2: Comprueba el efecto de la ejecución del programa en el registro.

El ensamblador del MIPS ofrece la posibilidad de cargar una constante de 32 bits en un registro utilizando una pseudoinstrucción. Ésta es la pseudoinstrucción `li`.

Crea un fichero con el siguiente código:

```
            .text #zona de instrucciones
main:       li $s0,0x12345678
```

Borra los valores de la memoria del *xspim* y carga este fichero.
Ejecuta el programa mediante el botón *run* del *xspim*.

Cuestión 3: Comprueba el efecto de la ejecución del programa en el registro.
Cuestión 4: Comprueba qué conjunto de instrucciones reales implementan esta pseudoinstrucción.

Carga de palabras (palabras de memoria a registro)

Crea un fichero con el siguiente código:

```
.data

   palabra:  .word 0x10203040
             .text                #zona de instrucciones
   main:     lw $s0,palabra($0)
```

Descripción:

La instrucción `lw` carga la palabra contenida en una posición de memoria, cuya dirección se especifica en la instrucción, en un registro. Dicha posición de memoria se obtiene sumando el contenido del registro (en este caso `$0`, que siempre vale cero) y el identificador `palabra`.

Borra los valores de la memoria del SPIM y carga el fichero anterior.

Cuestión 5: Localiza la instrucción en memoria de instrucciones e indica cómo ha transformado dicha instrucción el simulador.

Cuestión 6: Explica cómo se obtiene a partir de esas instrucciones la dirección de `palabra`. ¿Por qué crees que el simulador traduce de esta forma la instrucción original?

Cuestión 7: Analiza cada uno de los campos que componen estas instrucciones e indica el tipo de formato que tienen.

Ejecuta el programa mediante el botón *run*.

Cuestión 8: Comprueba el efecto de la ejecución del programa.

Cuestión 9: ¿Qué pseudoinstrucción permite cargar la dirección de un dato en un registro? Modifica el programa original para que utilice esta pseudoinstrucción, de forma que el programa haga la misma tarea. Comprueba qué conjunto de instrucciones sustituyen a la pseudoinstrucción utilizada una vez el programa se carga en la memoria del simulador.

Cuestión 10: Modifica el código para que en lugar de transferir la palabra contenida en la dirección de memoria referenciada por la etiqueta palabra, se transfiera la palabra que está contenida en la dirección referenciada por `palabra+1`. Explica qué ocurre y por qué.

Cuestión 11: Modifica el programa anterior para que guarde en el registro `$s0` los dos bytes de mayor peso de `palabra`. Nota: Utiliza la instrucción `lh` que permite cargar medias palabras (16 bits) desde memoria a un registro (en los 16 bits de menor peso del mismo).

Carga de bytes (bytes de memoria a registro)

Crea un fichero con el siguiente código:

```
.data
octeto:     .byte 0xf3
siguiente: .byte 0x20

            .text    #zona de instrucciones
   main:    lb $s0, octeto($0)
```

Descripción:

La instrucción `lb` carga el byte de una dirección de memoria en un registro. Al igual que antes, la dirección del dato se obtiene sumando el contenido del registro `$0` (siempre vale cero) y el identificador `octeto`.

Borra los valores de la memoria y carga el fichero.

Cuestión 12: Localiza la instrucción en memoria de instrucciones e indica cómo ha transformado dicha instrucción el simulador.

Ejecuta el programa.

Cuestión 13: Comprueba el efecto de la ejecución del programa.

Cuestión 14: Cambia en el programa la instrucción `lb` por `lbu`. ¿Qué sucede al ejecutar el programa? ¿Qué significa esto?

Cuestión 15: Si `octeto` se define como:

"`octeto:  .byte 0x30`", ¿existe diferencia entre el uso de la instrucción `lb` y `lbu`?¿ Por qué?

Cuestión 16: ¿Cuál es el valor del registro `$s0` si `octeto` se define como:

"octeto: .word 0x10203040"? ¿Por qué?

Cuestión 17: ¿Cuál es el valor del registro `$s0` si se cambia en `main` la instrucción existente por la siguiente:

main: lb $s0,octeto+1($0)?

¿Por qué? ¿Por qué en este caso no se produce un error de ejecución (excepción de error de direccionamiento)?

Almacenado de palabras (palabras de registro a memoria)

Crea un fichero con el siguiente código:

.data

```
palabra1: .word 0x10203040
palabra2: .space 4
palabra3: .word 0xffffffff

          .text     #zona de instrucciones
main:     lw $s0, palabra1($0)
          sw $s0, palabra2($0)
          sw $s0, palabra3($0)
```

Descripción:

La instrucción `sw` almacena la palabra contenida en un registro en una dirección de memoria. Esta dirección se obtiene sumando el contenido de un registro más un desplazamiento especificado en la instrucción (identificador).

Borra los valores de la memoria del SPIM y carga el fichero.

Cuestión 18: Localiza la primera instrucción de este tipo en la memoria de instrucciones e indica cómo ha transformado dicha instrucción el simulador.

Ejecuta el programa.

Cuestión 19: Comprueba el efecto de la ejecución del programa.

Almacenado de bytes (bytes de registro a memoria)

Crea un fichero con el siguiente código:

```
.data

   palabra:  .word 0x10203040
   octeto:        .space 2


                  .text     #zona de instrucciones
   main:          lw $s0, palabra($0)
                  sb $s0, octeto($0)
```

Descripción:

La instrucción "`sb`" almacena el byte de menor peso de un registro en una dirección de memoria. La dirección se obtiene sumando el desplazamiento indicado por el identificador y el contenido de un registro.

Borra los valores de la memoria y carga el fichero.

Cuestión 20: Localiza la instrucción en memoria de instrucciones e indica cómo ha transformado dicha instrucción el simulador.

Ejecuta el programa.

Cuestión 21: Comprueba el efecto de la ejecución del programa.

Cuestión 22: Modifica el programa para que el byte se almacene en la dirección "`octeto+1`". Comprueba y describe el resultado de este cambio.

Cuestión 23: Modifica el programa anterior para transferir a la dirección de `octeto` el contenido de la posición `palabra+3`.

Problemas propuestos

1. Diseña un programa ensamblador que defina el vector de enteros V=(10, 20, 25, 500, 3) en la memoria de datos a partir de la dirección 0x10000000 y cargue todos sus componentes en los registros $s0 - $s4.
2. Diseña un programa ensamblador que copie el vector definido en el problema anterior a partir de la dirección 0x10010000.
3. Diseña un programa ensamblador que, dada la palabra 0x10203040 almacenada en una posición de memoria, la reorganice en otra posición, invirtiendo el orden de sus bytes.
4. Diseña un programa ensamblador que, dada la palabra 0x10203040 definida en memoria la reorganice en la misma posición, intercambiando el orden se sus medias palabras. Nota: utiliza la instrucción lh y sh.
5. Diseña un programa en ensamblador que inicialice cuatro bytes a partir de la posición 0x10010002 a los siguientes valores 0x10, 0x20, 0x30, 0x40, y reserve espacio para una palabra a partir de la dirección 0x1001010. El programa transferirá los cuatro bytes contenidos a partir de la posición 0x10010002 a la dirección 0x1001010.