Fundamentos y Arquitectura de Computadores
Práctica 1

Diego R. Llanos Ferraris
Departamento de Informática
Universidad de Valladolid

13 de septiembre de 2010

Índice

1. Objetivos

El objetivo de esta práctica es que el alumno se familiarice con el entorno de desarrollo de programas en ensamblador para la arquitectura MIPS. Para ello se le pedirá que introduzca un programa en ensamblador y lo ejecute con el simulador para comprobar su funcionamiento, así como que realice una de modificación sencilla sobre el mismo.

2. El simulador `spim`

El simulador `spim` es un programa que permite ejecutar directamente código ensamblador MIPS. Por lo tanto, no es necesario ensamblar el código para su ejecución: basta con invocar el simulador, cargar el código fuente y ordenar su ejecución.

El simulador `spim` está disponible en versiones para Windows, Unix y Mac OS X. Puede descargarse de la siguiente dirección:

```
http://www.cs.wisc.edu/~larus/spim.html
```

En el tomo 3 del libro de Patterson y Hennessy [1], secciones A.9 y A.10, se describe en detalle el funcionamiento de `spim` y del juego de instrucciones MIPS.

3. Instalación de `spim` en Linux

Los pasos que hay que seguir para instalar `spim` en Linux son los siguientes:

1. Bajarse el código fuente de `spim` de la dirección:
   `http://www.cs.wisc.edu/~larus/SPIM/spim.tar.gz`
2. Descomprimir el archivo: `tar xvfz spim.tar.gz`
3. Ir al directorio siguiente: `cd spim-7.5/spim/`
4. Asegurarse de que tenemos instalado los programas `bison` y `flex`, con los comandos +which bison+ y +which flex+. El sistema debería responder con la ubicación de los ficheros ejecutables correspondientes: algo así como `/usr/bin/bison`. Lo mismo con `flex`. Si el sistema no responde, entonces habrá que instalarlos. En Ubuntu, esto puede hacerse con el comando `sudo apt-get install flex bison`.
5. Compilar el `spim`, ejecutando `make`.
6. Instalar el `spim`, ejecutando `sudo make install`.

4. Primeros pasos con `spim`

En las prácticas en el laboratorio utilizaremos la versión para Unix del `spim`. Sin embargo, las prácticas pueden realizarse también con las versiones para DOS y Windows, ya que el código fuente funcionará en todas las versiones.

> **Nota:** Lo primero es obtener una cuenta en el ordenador de prácticas, solicitándosela al profesor de la asignatura el primer día de prácticas. El ordenador de prácticas utilizada será `saturno.dcs.eup.uva.es`.

El simulador se ejecuta invocando simplemente:

```
[saturno]$ spim
```

En ese momento se ejecutará el simulador. Se trata de un simulador en línea de comandos, por lo que hay que indicarle a través del teclado los comandos que tiene que ejecutar.

Algunos de los comandos que se pueden utilizar son los siguientes. Para una lista más detallada, consultar el libro [1], sección A.9, página 44.

- **exit**: Sale del simulador.
- **load "fichero.s"**: Carga el programa en ensamblador `fichero.s`.
- **run**: Ejecuta el programa cargado.
- **step N**: Ejecuta N instrucciones del programa. Por defecto N es igual a 1.
- **continue**: Continúa la ejecución del programa.
- **print $N**: Muestra el contenido del registro $N.
- **print dir**: Muestra el contenido de la posición de memoria `dir`.
- **breakpoint dir**: Establece un punto de parada en la dirección `dir`. `dir` puede ser una dirección de memoria o una etiqueta del programa.
- **delete dir**: Borra un punto de parada.

5. Nuestro primer programa en MIPS

La primera parte de la práctica consiste en introducir y ejecutar el siguiente programa en ensamblador MIPS:

```
        .data
cad0:   .asciiz "Introduzca un número entero: "
cad1:   .asciiz "Introduzca otro número entero: "
cad2:   .asciiz "La suma de ambos es: "
cr:     .byte 13,10,0

        .text
main:
        # Se escribe la cadena pidiendo el primer número
        addi $v0, $zero, 4
        la $a0, cad0
        syscall

        # Se lee el primer número
        addi $v0, $zero, 5
        syscall

        add $s0, $v0, $zero   # Se guarda el número en $s0

        # Se escribe la cadena pidiendo el segundo número
        addi $v0, $zero, 4
        la $a0, cad1
        syscall

        # Se lee el segundo número
        addi $v0, $zero, 5
```

```
syscall

add $s1, $v0, $zero  # Se guarda el número en $s1

add $s2, $s0, $s1 # Se suman ambos números

# Se escribe la cadena previa al resultado
addi $v0, $zero, 4
la $a0, cad2
syscall

# Se escribe el resultado
addi $v0, $zero, 1
add $a0, $zero, $s2
syscall

# Se escribe un retorno de carro al final
addi $v0, $zero, 4
la $a0, cr
syscall

# Fin del programa.
addi $v0, $zero, 10
syscall
```

El programa deberá introducirse utilizando el editor `vi`, y almacenado como `suma.s`. Una vez introducido, se lo ejecutará con `spim`, a través de los comandos `load "suma.s"` y `run`, como puede verse en el siguiente ejemplo:

```
[diego@saturno diego]$ spim
SPIM Version 6.3a of January 14, 2001
Copyright 1990-2000 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /bin/trap.handler
(spim) load "suma.s"
(spim) run
Introduzca un número entero: 23
Introduzca otro número entero: 32
La suma de ambos es: 55
(spim)
```

6. Acerca del programa

El programa propuesto lee dos números enteros por teclado y los suma, escribiendo el resultado por pantalla. Como puede verse en el programa, hay dos zonas diferenciadas:

- La zona de datos, que comienza con la directiva `.data`. En esa zona se reserva espacio para los datos que vayan a usarse en el programa.
- La zona de código, que comienza con la directiva `.text`. En esa zona va el código del programa.

Todos los programas deberán contener una etiqueta llamada `main`, que será la etiqueta de la primera instrucción del programa.

6.1. Directivas

En nuestro programa vemos que hay instrucciones conocidas y otras nuevas. En el segmento de datos aparecen algunas instrucciones que comienzan por un punto, como `.asciiz` o `.byte`. Todas las instrucciones que comienzan por un punto se llaman **directivas**. Las directivas no forman parte del juego de instrucciones MIPS, sino que sirven para declarar tipos de datos o (como hemos visto arriba) zonas del programa.

Algunas directivas de declaración de datos son las siguientes. La lista completa puede consultarse en [1], tomo 3, sección A.10, página 50.

`.asciiz cadena` : Sirve para declarar una cadena de caracteres. El final de cadena se marca con un cero (0x00).

`.ascii cadena` : Sirve para declarar una cadena de caracteres. No se marca el final de cadena.

`.byte b1, b2,... ,bn` : Sirve para declarar un conjunto de bytes consecutivos en memoria.

`.double d1, d2,... ,dn` : Sirve para declarar un conjunto de números en coma flotante de doble precisión (8 bytes) consecutivos en memoria.

`.float f1, f2,... ,fn` : Sirve para declarar un conjunto de números en coma flotante de simple precisión (4 bytes) consecutivos en memoria.

`.space tamaño` : Se utiliza para reservar una zona de memoria con el tamaño indicado. Esta directiva es necesaria tanto para crear un *buffer* en donde almacenar cadenas de caracteres leidas por teclado como también para reservar espacio para la pila.

`.word w1, w2,... ,wn` : Sirve para declarar un conjunto de números enteros (4 bytes) consecutivos en memoria.

6.2. Llamadas al sistema

Las llamadas al sistema son solicitudes que nuestro programa realiza al sistema operativo para realizar diferentes operaciones. Normalmente se trata de operaciones de entrada-salida. Las llamadas al sistema se estudiarán en detalle en los temas 7 y 8 de la asignatura: de momento, basta con decir que una llamada al sistema es como una llamada a función, con la diferencia de que el cuerpo de la función no forma parte de nuestro programa sino que forma parte del sistema operativo.

Por ejemplo, para sacar el número 34 por pantalla las instrucciones correspondientes son:

```
addi $v0, $zero, 1
addi $a0, $zero, 34
syscall
```

La tercera instrucción es `syscall`, que actúa como la llamada a la función que hará la tarea. No se trata propiamente de una instrucción MIPS, sino que es una *pseudoinstrucción*: una especie de instrucción que le damos al simulador para que realice la llamada al sistema.

Las dos primeras instrucciones le indican al sistema qué es lo que tiene que hacer. Como puede verse, se almacena información en los registros `$a0` y `$v0`. En las llamadas al sistema, `$v0` recibe el **código** de la operación a realizar. En nuestro caso, la operación es la número 1: mostrar por pantalla un número entero. El número que deseamos mostrar hay que guardarlo en `$a0`. Tras ello, ejecutamos `syscall` y la llamada al sistema realizará la tarea deseada.

La lista de llamadas al sistema que utilizaremos es la siguiente:

Servicio	Código en $v0	Argumentos	Resultado
print_int	1	$a0=entero	Muestra un número entero
print_float	2	$f12=coma flotante	Muestra un número real
print_double	3	$f12=doble precisión	Muestra un número de doble precisión
print_string	4	$a0=dir. de cadena	Muestra una cadena de caracteres
read_int	5		$v0=entero
read_float	6		$f0=coma flotante
read_double	7		$f0=doble precisión
read_string	8	$a0=buffer, $a1=longitud	
end_program	10		Finaliza el programa

6.3. Pseudoinstrucciones

El simulador `spim` añade al juego de instrucciones de MIPS algunas instrucciones que facilitan la tarea del programador. Estas instrucciones añadidas se denominan *pseudoinstrucciones*. Por ejemplo, la pseudoinstrucción `la` permite cargar una dirección de memoria en un registro:

```
la $a0, cad1
```

El simulador `spim` traduce automáticamente esta pseudoinstrucción a las instrucciones MIPS necesarias para llevar a cabo esta tarea.

7. Ejecución paso a paso

Lo normal es que el programa no funcione a la primera. La mejor manera de detectar los errores es la ejecución paso a paso. Para ello cargamos nuestro programa en el simulador y ejecutamos `step`. Esto ejecutará la primera instrucción del programa. Repitiendo este comando continuaremos ejecutando el programa paso a paso. En nuestro ejemplo, la salida será la siguiente:

```
[diego@saturno diego]$ spim
SPIM Version 6.3a of January 14, 2001
Copyright 1990-2000 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /bin/trap.handler
(spim) load "suma.s"
(spim) step
[0x00400000]    0x8fa40000  lw $4, 0($29)           ; 102: lw $a0, 0($sp) # argc
(spim) step
[0x00400004]    0x27a50004  addiu $5, $29, 4        ; 103: addiu $a1, $sp, 4 # argv
(spim) step
[0x00400008]    0x24a60004  addiu $6, $5, 4         ; 104: addiu $a2, $a1, 4 # envp
(spim) step
[0x0040000c]    0x00041080  sll $2, $4, 2           ; 105: sll $v0, $a0, 2      addu $a2, $a2, $v0
(spim) step
[0x00400010]    0x00c23021  addu $6, $6, $2         ; 106: addu $a2, $a2, $v0   jal main
(spim) step
[0x00400014]    0x0c100008  jal 0x00400020 [main]   ; 107: jal main     li $v0 10
(spim) step
[0x00400020]    0x20020004  addi $2, $0, 4          ; 11: addi $v0, $zero, 4
(spim)
```

Lo que aparece en la primera columna es la dirección de la instrucción en memoria. En la segunda columna aparece la codificación de la instrucción en código máquina (en hexadecimal). A continuación la instrucción que se está ejecutando y, finalmente, la línea de código original del programa.

> **Nota:** Como puede verse, las instrucciones que están en las direcciones 0x00400000 a 0x00400014, ambas inclusive, no forman parte de nuestro programa. La primera instrucción de nuestro programa (`addi $v0, $zero, 4`) se encuentra en la dirección

0x00400020. Estas primeras instrucciones las añade el simulador para permitir a nuestro programa recibir argumentos a través de la línea de comandos.

La ventaja de ejecutar paso a paso el programa es que podemos comprobar el valor de cualquier registro tras la ejecución de una instrucción concreta. En nuestro ejemplo, tras haber ejecutado la instrucción `addi $v0, $zero, 4` podemos comprobar el valor del registro `$v0` a través del comando `print $v0`:

```
(spim) step
[0x00400020]    0x20020004  addi $2, $0, 4          ; 11: addi $v0, $zero, 4
(spim) print $v0
Reg 2 = 0x00000004 (4)
(spim)
```

8. Una modificación al programa

Como puede verse en el código fuente del programa, tras escribir el número por pantalla se invoca una vez más a la llamada al sistema número 4. El comentario correspondiente indica que "se escribe un retorno de carro al final". Pon un símbolo de comentario a esas tres líneas, de modo que queden así:

```
# Se escribe un retorno de carro al final
# addi $v0, $zero, 4
# la $a0, cr
# syscall
```

Vuelve a ejecutar el programa. La diferencia es que ahora no se imprime un retorno de carro tras haber escrito el resultado por pantalla. Esas líneas son el equivalente a un `printf("\n");` en un programa en C.

9. Entrega de la práctica

El desarrollo de la práctica es **individual**. La entrega de cada práctica se realizará por correo electrónico. Cada alumno deberá enviar el fichero fuente `suma.s`, con la modificación descrita en la sección anterior. En las tres primeras líneas del fichero fuente enviado deberá figurar el número de la práctica y el nombre completo del alumno que la ha realizado, con el siguiente formato:

```
# Práctica 1
# García García, Juan
```

El fichero deberá enviarse a la dirección de correo electrónico `diego@infor.uva.es`. Desde el ordenador de prácticas `saturno`, esto puede hacerse ejecutando los comandos siguientes:

```
[saturno]$ cp suma.s copia.s
[saturno]$ mail -s "Practica 1" diego@infor.uva.es < copia.s
```

Estos comandos hacen una copia del fichero con el código fuente de la práctica y envían dicha copia. Es mejor hacerlo así que enviar directamente el fichero original, ya que si nos equivocamos en el comando `mail` y ponemos un > en lugar de un <, habremos sobreescrito el fichero con la práctica original. Si primero hacemos una copia, lo peor que puede pasar es que la copia se pierda.

Fecha de entrega

Sólo se admitirán las prácticas que *lleguen* a la dirección de correo electrónico indicada hasta las 23:59 hs. del día fijado como fecha límite de entrega de la misma. Para esta práctica la fecha límite de envío es el

domingo 23 de octubre de 2010

La entrega de las prácticas es obligatoria para poder presentarse al examen final de la asignatura. La detección de copias en las prácticas realizadas por diferentes alumnos conllevará el suspenso automático de las prácticas de todos ellos. En el tablón de anuncios del Departamento de Informática en la EUP se publicará la lista de alumnos que han entregado la práctica, junto con su calificación. La nota final de prácticas se calculará como una media ponderada de las notas de cada práctica.

Referencias

[1] David A. Patterson and John L. Hennessy. *Estructura y Diseño de Computadores*. Ed. Reverté, 1 edition, 2000. ISBN 8429126163.