

# ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA



UNIVERSIDAD CARLOS III DE MADRID

## **Práctica 2**

### **Ensamblador de MIPS e introducción al simulador PCSpim**

Octubre 2013



## **Contenido**

<b>OBJETIVOS DE LA PRÁCTICA.....</b>	<b>4</b>
<b>DESCRIPCIÓN DEL SIMULADOR SPIM.....</b>	<b>4</b>
<b>EJERCICIO 1.....</b>	<b>7</b>
<b>EJERCICIO 2.....</b>	<b>8</b>
<b>EJERCICIO 3.....</b>	<b>9</b>
<b>EJERCICIO 4.....</b>	<b>10</b>
<b>EJERCICIO 5.....</b>	<b>11</b>
<b>EJERCICIO 6.....</b>	<b>12</b>
<b>EJERCICIO 7.....</b>	<b>13</b>
<b>PROCEDIMIENTO DE EVALUACIÓN DE LA PRÁCTICA.....</b>	<b>14</b>
<b>PROCEDIMIENTO DE ENTREGA.....</b>	<b>15</b>

# Objetivos de la práctica

El objetivo de esta práctica es que el alumno se familiarice con la programación en ensamblador y la representación de distintos tipos de datos utilizando el ensamblador del MIPS32. Para el desarrollo de esta práctica se usará el emulador SPIM disponible en:

- <http://spimsimulator.sourceforge.net/>

SPIM es un simulador auto-contenido que ejecutará programas escritos en el lenguaje ensamblador del MIPS32. SPIM proporciona además un depurador y un conjunto mínimo de servicios del sistema operativo.

## Descripción del simulador SPIM

La versión para Windows de SPIM posee una interfaz similar a la que aparece en la siguiente figura. Existen cuatro paneles horizontales separados por tres líneas de color gris. Cada uno de estos paneles presenta un contenido distinto.

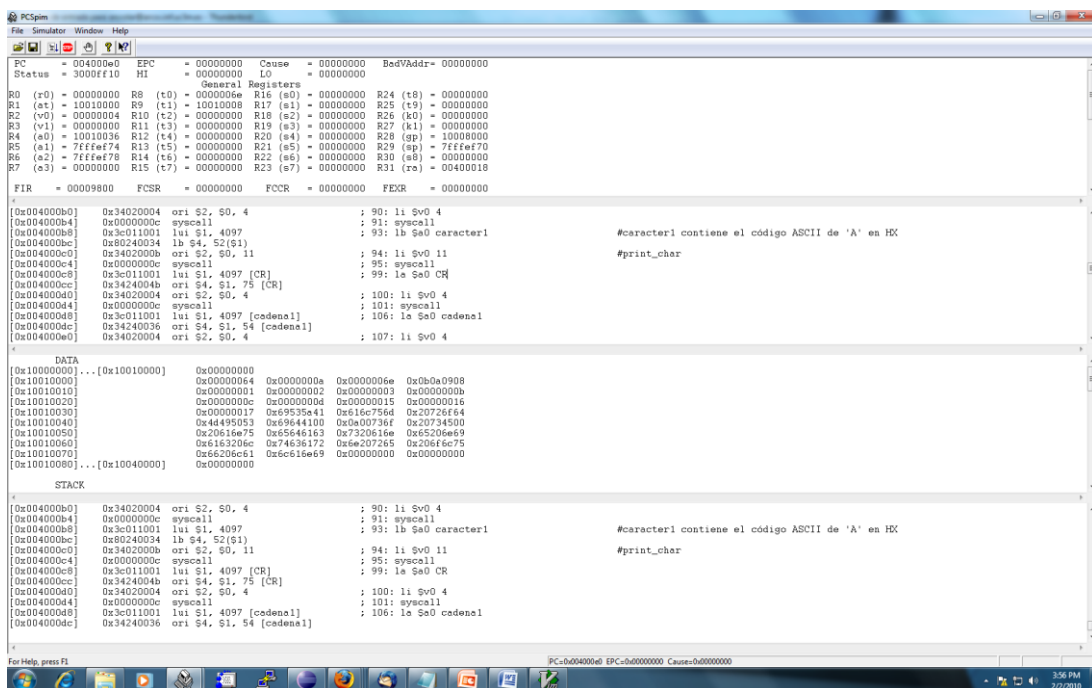


Figura 1. Interfaz del simulador SPIM.

## 1. PANEL 1. Registros de la máquina MIPS.

PC	=	00400020	EPC	=	00000000	Cause	=	00000000	BadVAddr	=	00000000
Status	=	3000fff10	HI	=	00000001	LO	=	0000004e			
General Registers											
R0 (r0)	=	00000000	R8 (t0)	=	00000009	R16 (s0)	=	00000000	R24 (t8)	=	00000000
R1 (at)	=	00000000	R9 (t1)	=	00000024	R17 (s1)	=	00000000	R25 (t9)	=	00000000
R2 (v0)	=	0000000a	R10 (t2)	=	00000009	R18 (s2)	=	00000000	R26 (k0)	=	00000000
R3 (v1)	=	00000000	R11 (t3)	=	0000009d	R19 (s3)	=	00000000	R27 (k1)	=	00000000
R4 (a0)	=	00000006	R12 (t4)	=	00000002	R20 (s4)	=	00000000	R28 (gp)	=	10008000
R5 (a1)	=	7ffff6dc	R13 (t5)	=	00000001	R21 (s5)	=	00000000	R29 (sp)	=	7ffff6d8
R6 (a2)	=	7ffff6ec	R14 (t6)	=	00000000	R22 (s6)	=	00000000	R30 (s8)	=	00000000
R7 (a3)	=	00000000	R15 (t7)	=	00000000	R23 (s7)	=	00000000	R31 (ra)	=	00400018
FIR	=	00009800	FCSR	=	00000000	FCCR	=	00000000	FEXR	=	00000000

Incluye el banco de registros de la máquina MIPS.

En la primera línea aparece el contador de programa (**PC**) y otros registros especiales como **HI** (almacena parte alta de registro) y **LO** (almacena la parte baja).

En las siguientes líneas aparecerán los 32 registros enteros de propósito general (**R0** a **R31**).

Finalmente aparecen los 32 registros usados para representar números en coma flotante, tanto en precisión simple, como en precisión doble (**FP0** a **FP31**).

## 2. PANEL 2. Segmento de Texto.

[0x00400000]	0x8fa40000	lw \$4, 0(\$29)	; 183: lw \$a0 0(\$sp)	# argc
[0x00400004]	0x27a50004	addiu \$5, \$29, 4	; 184: addiu \$a1 \$sp 4	# argv
[0x00400008]	0x24a60004	addiu \$6, \$5, 4	; 185: addiu \$a2 \$a1 4	# envp
[0x0040000c]	0x00041080	sll \$2, \$4, 2	; 186: sll \$v0 \$a0 2	
[0x00400010]	0x00c23021	addu \$6, \$6, \$2	; 187: addu \$a2 \$a2 \$v0	
[0x00400014]	0x0c100009	jal 0x00400024 [main]	; 188: jal main	
[0x00400018]	0x00000000	nop	; 189: nop	
[0x0040001c]	0x3402000a	ori \$2, \$0, 10	; 191: li \$v0 10	
[0x00400020]	0x0000000c	syscall	; 192: syscall	# syscall 10 (exit)
[0x00400024]	0x34080000	ori \$8, \$0, 0	; 7: li \$t0, 0	
[0x00400028]	0x34090000	ori \$9, \$0, 0	; 8: li \$t1, 0	
[0x0040002c]	0x3c011001	lui \$1, 4097	; 9: lw \$t2, n	
[0x00400030]	0x8c2a0024	lw \$10, 36(\$1)		

Incluye el código del programa ensamblador a ejecutar, identificado mediante la directiva *.text*.

Cada *pseudoinstrucción* ocupa una única línea, identificada por una dirección de memoria que aparece en la parte izquierda de la línea.

La primera dirección de memoria utilizada para almacenar el texto de un programa es 0x00400000.

### 3. PANEL 3. Segmento de Datos y pila.

DATA				
[0x10000000]...[0x10010000]	0x00000000			
[0x10010000]	0x0000000c	0x00000019	0x00000024	0x00000010
[0x10010010]	0x0000001c	0x00000025	0x0000007c	0x0000009c
[0x10010020]	0x0000009d	0x00000009	0x00000000	0x00000000
[0x10010030]...[0x10040000]	0x00000000			
STACK				
[0x7ffff6d8]	0x00000003	0x7ffff7b9		
[0x7ffff6e0]	0x7ffff7b3	0x7ffff79a	0x00000000	0x7ffffe1
[0x7ffff6f0]	0x7ffff7b9	0x7ffff7a0	0x7ffff5f	0x7ffff28
[0x7ffff700]	0x7ffffeec	0x7ffffebb	0x7ffffea4	0x7ffffe80
[0x7ffff710]	0x7ffffe6c	0x7ffffe5f	0x7ffffe48	0x7ffffe1d

Incluye el conjunto de datos definidos en el segmento de datos *.data* del programa a ejecutar. Estos datos se almacenan de manera lineal en memoria, a partir de la dirección **0x10000000**. A continuación se muestra el contenido de la pila (*stack*) que comienza en la dirección **0x7ffff6ff**.

### 4. PANEL 4. Pantalla para depuración de programas

```
See the file README for a full copyright notice.
Loaded: C:\Program Files (x86)\PCSpim\exceptions.s
C:\Users\Carlos\Documents\inversor.s successfully loaded
Memory and registers cleared and the simulator reinitialized.

SPIM Version 9.0.1 of January 2, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
Loaded: C:\Program Files (x86)\PCSpim\exceptions.s
C:\Users\Carlos\Documents\contador.s successfully loaded
```

El último panel es el depurador, destinado a visualizar mensajes de control o error del simulador SPIM.

# Ejercicio 1

En el material de apoyo de la práctica 2, que se puede descargar de aula global, se dispone del fichero *ejercicio1.s*.

Analícelo, cárguelo en el simulador PCSpim, ejecútelo y responda a las siguientes preguntas en la memoria:

- a) Indique la dirección de memoria donde comienza el main del *ejercicio1.s*.
- b) Indique la dirección de memoria que contiene el registro `$ra` después de ejecutar el programa. Indique además la instrucción que contiene esa dirección de memoria.
- c) La dirección **0x10010043** contiene un carácter de la cadena almacenada en el segmento de datos.  
Indique el valor que contiene esa dirección de memoria, y el carácter ASCII correspondiente.
- d) Indique en qué dirección está la segunda **'i'** de la cadena almacenada en el segmento de datos.
- e) Indique la dirección que corresponde a utilizar: **hueco+3**
- f) Indique la dirección que corresponde a utilizar: **cadena+2**
- g) Escriba las instrucciones necesarias para imprimir por pantalla el tercer byte de la cadena.

## Ejercicio 2

En un archivo con el nombre *max.s* está codificado un programa que tiene como objetivo calcular el valor máximo de un conjunto de datos de tipo entero de longitud 32 bits (.word).

El código presenta un **único** error que hace que el programa no funcione correctamente.

Se pide:

- Detecte y corrija dicho error para que el programa imprima correctamente el valor máximo del array.
- Indique para qué se utiliza cada uno de los registros usados en el programa.
- En cada iteración del bucle identificado por la etiqueta m2, indique el valor de cada uno de los registros usados. Complete una tabla como la siguiente:

REGISTROS	Iteración 1	Iteración 2	...	Iteración n
R8 (\$t0)				
R16 (\$s0)				
R18 (\$s2)				
PC				

## Ejercicio 3

En un archivo con el nombre *strLenCount.s*, desarrollar un programa escrito en código ensamblador de MIPS32 que calcule la longitud de una cadena de caracteres definida en la zona de datos y cuente el número de ocurrencias de un carácter definido.

La función **strLenCount** cuenta el número de caracteres que tiene la cadena. Para este cálculo no se tendrá en cuenta el carácter nulo. La función imprimirá por consola el valor calculado y el número de ocurrencias del carácter indicado.

La cadena de caracteres de prueba deberá declararse en el segmento de datos del fichero *strLenCount.s* como **.asciiz**, de la siguiente manera:

```
.data
string:      .asciiz "Innovation distinguishes between a leader and a follower."
char:        .byte 'w'
```

Donde:

- **string** es la cadena de caracteres que se quiere tratar.
- **char** es el carácter a contar

La salida por consola del programa indicado será la siguiente:

```
57
2
```

### NOTA:

- El orden de retorno de la función es importante, primero debe mostrarse siempre la longitud de la cadena y a continuación, y separado por un salto de línea, el número de ocurrencias.
- El programa debe funcionar independientemente de la cadena que se asigne a **string** y valor de **char**.

## Ejercicio 4

En un archivo con el nombre *toUpper.s*, desarrollar un programa escrito en código ensamblador de MIPS32 que dada una cadena de caracteres convierta todas las minúsculas a mayúsculas e imprima por pantalla el resultado de la conversión.

La función **toUpper** lee la cadena de caracteres almacenada en la región de datos y cambia las minúsculas a mayúsculas, imprimiendo por pantalla la cadena transformada. El resto de caracteres deben permanecer igual.

La cadena de caracteres debe declararse en la región de datos de la siguiente forma:

```
.data
string:      .asciiz "Innovation distinguishes between a leader and a follower."
```

Donde:

- **string** es la cadena de caracteres que se quiere tratar.

Para el ejemplo anterior el resultado esperado es el siguiente:

```
INNOVATION DISTINGUISHES BETWEEN A LEADER AND A FOLLOWER.
```

### NOTA:

- El programa debe funcionar independientemente de la cadena que se asigne a **string**.
- El programa debe modificar la cadena en memoria.

## Ejercicio 5

En un archivo con el nombre *get.s*, desarrollar un programa escrito en código ensamblador de MIPS32 que, dado un array y dos coordenadas, imprima por consola el valor almacenado en esa posición indicada.

La función **get** se encargará de desplazarse a la posición indicada por las coordenadas x e y para mostrar por consola el valor almacenado en dicha posición.

**Si las coordenadas quedaran fuera de los rangos admisibles del array, el valor retornado será -1.**

La región de datos asociada a este ejercicio será como la que se puede apreciar a continuación:

```
.data
array:      .word 1, 1, 1, 1
            .word 1, 1, 1, 1
            .word 1, 1, 1, 1
            .word 1, 1, 1, 1
columns:    .word 4
rows:       .word 4
column_index: .word 1
row_index:  .word 2
```

Donde:

- **array** es el array a tratar.
- **columns** son el número de columnas que tiene el array
- **rows** son el número de filas que tiene el array
- **column\_index** columna en la que se localiza el número buscado (0 a n-1)
- **row\_index** fila en la que se localiza el número buscado (0 a n-1)

La salida por consola del programa indicado será la siguiente:

1

## Ejercicio 6

En un archivo con el nombre *set.s*, desarrollar un programa escrito en código ensamblador de MIPS32 que, dado un array, dos coordenadas y un valor, reemplace el número existente en la posición indicada por el indicado.

La función **set** se encargará de desplazarse a la posición indicada por las coordenadas x e y y reemplazar el valor existente por el número indicado.

**Si las coordenadas quedaran fuera de los rangos admisibles del array no se realizará nada.**

La región de datos asociada a este ejercicio será como la que se puede apreciar a continuación:

```
.data
array:      .word 1, 1, 1, 1
            .word 1, 1, 1, 1
            .word 1, 1, 1, 1
            .word 1, 1, 1, 1
columns:    .word 4
rows:       .word 4
column_index: .word 1
row_index:  .word 2
number:     .word 5
```

Donde:

- **array** es el array a tratar.
- **columns** son el número de columnas que tiene el array
- **rows** son el número de filas que tiene el array
- **column\_index** columna en la que se localiza el número buscado (0 a n-1)
- **row\_index** fila en la que se localiza el número buscado (0 a n-1)
- **number** es número que se desea almacenar

## Ejercicio 7

En un archivo con el nombre *arrayCopy.s*, desarrollar un programa escrito en código ensamblador de MIPS32 que, dada un array, realice una copia en una zona de memoria reservada de forma dinámica.

La función **arrayCopy** calcula el tamaño total del array origen, reserva una región de memoria del tamaño calculado y copia en dicha región el array original.

La región de datos asociada a este ejercicio será como la que se puede apreciar a continuación:

```
.data
    array:          .word 1, 1, 1, 1
                   .word 1, 1, 1, 1
                   .word 1, 1, 1, 1
                   .word 1, 1, 1, 1
    columns:        .word 4
    rows:           .word 4
```

Donde:

- **array** es el array a tratar.
- **columns** son el número de columnas que tiene el array
- **rows** son el número de filas que tiene el array

# Procedimiento de evaluación de la práctica

La evaluación de la práctica se va a dividir en dos partes.

- **Código (8 puntos)**

- Ejercicio 1 (*0.75 punto*)
- Ejercicio 2 (*0.75 puntos*)
- Ejercicio 3 (*1 punto*)
- Ejercicio 4 (*1 punto*)
- Ejercicio 5 (*1.25 puntos*)
- Ejercicio 6 (*1.25 puntos*)
- Ejercicio 7 (*2 puntos*)

- **Memoria (2 puntos)**

Los ejercicios 1, 2, 3, 4, 5, 6 y la memoria son obligatorios. Deben entregarse para seguir la evaluación continua.

## Procedimiento de entrega

La entrega de la práctica 2 se realizará de la siguiente manera: los **ejercicios** de este cuaderno y la memoria completa podrá entregarse hasta el día **15 de Noviembre de 2013 a las 23:55 horas**.

**Entregador 1:** Se deberá entregar un único archivo comprimido en formato zip con el nombre `ec_p2_AAAAAAAAAA_BBBBBBBBBB.zip` donde A...A y B...B son los NIAs de los integrantes del grupo. El archivo zip debe contener:

- `ejercicio1.s`
- `max.s`
- `strLenCount.s`
- `toUpper.s`
- `get.s`
- `set.s`
- `arrayCopy.s`

**Entregador 2:** Se deberá entregar la memoria en un único archivo en formato pdf con el nombre `ec_p2_AAAAAAAAAA_BBBBBBBBBB.pdf` donde A...A y B...B son los NIAs de los integrantes del grupo. La memoria tendrá que contener al menos los siguientes apartados:

- Portada donde figuren los autores (incluyendo nombre completo, NIA y dirección de correo electrónico)
- Índice de contenidos.
- Descripción de los programas solicitados.
- Respuestas a las preguntas planteadas en los ejercicios debidamente justificadas.
- Batería de pruebas utilizadas y resultados obtenidos.
- Conclusiones y problemas encontrados.

### **NOTA: NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA.**

Aprobar la memoria es tan imprescindible para aprobar la práctica, como el correcto funcionamiento de la misma. Si al evaluarse la memoria de su práctica, se considera que no alcanza el mínimo admisible, su práctica estará suspensa.

**La longitud de la memoria no deberá superar las 10 páginas** (portada e índice incluidos)

- La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL 2 se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas.
- La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

## **Normas**

- 1) Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.
- 2) Un programa no comentado, obtendrá una calificación de 0.
- 3) La entrega de la práctica se realizará a través de los entregadores habilitados. No se permite la entrega a través de correo electrónico sin autorización previa.
- 4) Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, ambas obtendrán una calificación de 0.