

UNIVERSIDAD DE LA HABANA

FACULTAD DE MATEMÁTICA Y COMPUTACIÓN

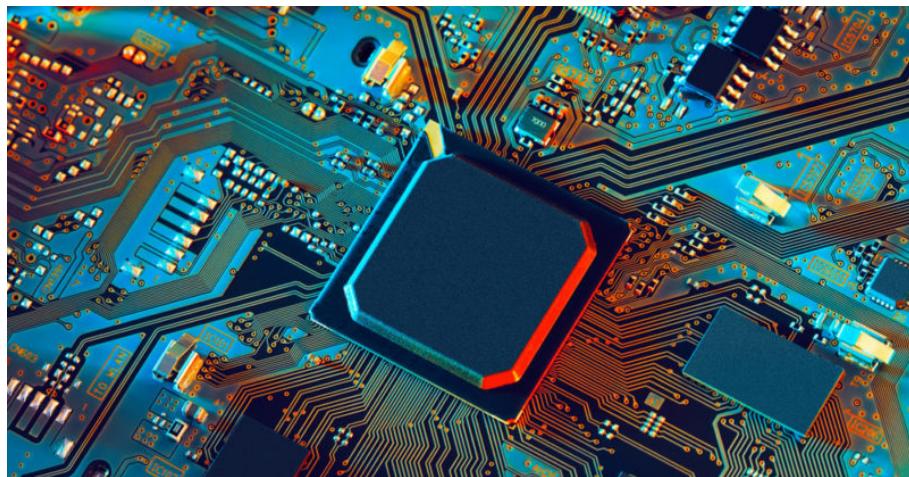
ARQUITECTURA DE COMPUTADORAS

---

## S-MIPS

Proyecto Final

---



Curso 2024

## ÍNDICE

1. Introducción	3
2. Instrucciones y sus formatos	3
2.1. Notas generales	3
3. Instrucciones Aritméticas	4
3.1. Addition: <b>add</b>	4
3.2. Subtract: <b>sub</b>	4
3.3. Multiply: <b>mult</b>	4
3.4. Unsigned multiply: <b>mulu</b>	4
3.5. Divide: <b>div</b>	4
3.6. Unsigned divide: <b>divu</b>	5
3.7. Addition inmediate: <b>addi</b>	5
4. Instrucciones Lógicas	5
4.1. Logical and: <b>and</b>	5
4.2. Logical or, nor, xor: <b>or, nor, xor</b>	5
4.3. Logical and immediate: <b>andi</b>	5
4.4. Logical or immediate & xor immediate: <b>ori, xori</b>	6
5. Instrucciones de Comparación	6
5.1. Set less than: <b>slt</b>	6
5.2. Unsigned set less than: <b>sltu</b>	6
5.3. Set less than immediate: <b>slti</b>	6
5.4. Set less than immediate unsigned: <b>sltiu</b>	6
6. Instrucciones de Copia de Registros	6
6.1. Move from Hi: <b>mfhi</b>	6
6.2. Move from Lo: <b>mflo</b>	7
7. Instrucciones de Salto	7
7.1. No operation: <b>nop</b>	7
7.2. Branch on equal: <b>beq</b>	7
7.3. Branch on not equal: <b>bne</b>	7
7.4. Branch on less than or equal zero: <b>blez</b>	8
7.5. Branch on greater than zero: <b>bgtz</b>	8
7.6. Branch on less than zero: <b>bltz</b>	8
7.7. Jump: <b>j</b>	8
7.8. Jump register: <b>jr</b>	9
8. Instrucciones de Memoria	9
8.1. Load word: <b>lw</b>	9
8.2. Store word: <b>sw</b>	9
8.3. Pop from stack: <b>pop</b>	9
8.4. Push to stack: <b>push</b>	10
9. Instrucciones especiales	10
9.1. Halt program: <b>halt</b>	10
9.2. Write to terminal: <b>tty</b>	10
9.3. Set random: <b>rnd</b>	10
9.4. Set key: <b>kbd</b>	11
10. Program Counter	11
11. Acceso a memoria	11
12. Interfaz con la memoria	12
13. Caché	13

## 1. INTRODUCCIÓN

El proyecto consiste en diseñar en *Logisim* un procesador que implemente la arquitectura de juegos de instrucciones S-MIPS (Simplified-MIPS).

S-MIPS es una arquitectura de 32 bits. El procesador tiene 32 registros de propósito general de 32 bits, nombrados de  $R_0$  a  $R_{31}$ . Por convenio,  $R_0$  siempre tiene el valor constante 0 independientemente de las operaciones que se realicen sobre él y  $R_{31}$  (también denominado *SP*) actúa como puntero de la pila (*Stack Pointer*). Cuenta con dos registros adicionales *Hi* y *Lo* donde se almacena el resultado de la división y la multiplicación. También se debe disponer de un registro (denominado *PC*) para el *Program Counter*, el cual almacenará la dirección en la memoria de la próxima instrucción a leer de la *RAM*.

El procesador debe implementarse en el módulo *S-MIPS* del circuito *S-MIPS Board* que se brinda. No se debe modificar el circuito *RAM* ni el circuito *S-MIPS Board*. Durante la evaluación se utilizarán estos circuitos tal y como se entregó. Además queda prohibido utilizar las componentes *RAM* y *ROM* que proporciona *Logisim*.

## 2. INSTRUCCIONES Y SUS FORMATOS

### 2.1. Notas generales.

- $R_s$ ,  $R_t$  y  $R_d$  especifican registros de propósito general.
- Un elemento entre corchetes ([ ]) indica “el contenido de”. Por ejemplo  $[R_3] + [R_{22}]$  se refiere a la suma de los valores almacenados en los registros  $R_3$  y  $R_{22}$ .
- $[PC]$  especifica la dirección de la instrucción en ejecución. Por ejemplo, saltar a la próxima instrucción es  $[PC] \leftarrow [PC] + 4$
- $I$  se refiere a los bits de la instrucción y el subíndice indica a cuáles de estos bits se refiere.  $[I_{15,\dots,0}]$  se refiere al contenido de los 16 primeros bits de la instrucción, que en el caso de las instrucciones de tipo I es la constante.
- $\parallel$  indica la concatenación de bits.
- Los sobre-índices indican la repetición de un valor binario.  $0^7$  se refiere a: 000 0000<sub>2</sub>.
- $M\{I\}$  se refiere a los 32 bits almacenados en la memoria RAM en la dirección divisible por 4 más cercana al valor de  $I$ , es decir, en la dirección  $I - I \% 4$ .

Las instrucciones de S-MIPS tienen una longitud constante de 32 bits. Hay 3 formatos de instrucciones distintos:

Tipo R:	Op-code	Rs	Rt	Rd	x	Func-code
---------	---------	----	----	----	---	-----------

Tipo I:	Op-code	Rs	Rt	Constante (complemento a2)
---------	---------	----	----	----------------------------

Tipo J:	Op-code	Destino-salto
---------	---------	---------------

Además se considera como el bit menos significativo (o bit 0) al bit más a la derecha, y como el bit más significativo (o bit 31) al bit más a la izquierda.

### 3. INSTRUCCIONES ARITMÉTICAS

#### 3.1. Addition: add.

<b>Tipo R:</b>	00 0000	$R_s$	$R_t$	$R_d$	0 0000	10 0000
----------------	---------	-------	-------	-------	--------	---------

Efectos de la instrucción:

$$R_d \leftarrow [R_s] + [R_t]$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

`add Rd, Rs, Rt`

#### 3.2. Subtract: sub.

<b>Tipo R:</b>	00 0000	$R_s$	$R_t$	$R_d$	0 0000	10 0010
----------------	---------	-------	-------	-------	--------	---------

Efectos de la instrucción:

$$R_d \leftarrow [R_s] - [R_t]$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

`sub Rd, Rs, Rt`

#### 3.3. Multiply: mult.

<b>Tipo R:</b>	00 0000	$R_s$	$R_t$	0 0000	0 0000	01 1000
----------------	---------	-------	-------	--------	--------	---------

Efectos de la instrucción:

$$Hi \parallel Lo \leftarrow [R_s] * [R_t]$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

`mult Rs, Rt`

#### 3.4. Unsigned multiply: mulu.

Idéntica a la instrucción `mult` excepto:

- Func-code = 01 1001
- El contenido de  $R_s$  y  $R_t$  se considera como enteros sin signo.

#### 3.5. Divide: div.

<b>Tipo R:</b>	00 0000	$R_s$	$R_t$	0 0000	0 0000	01 1010
----------------	---------	-------	-------	--------	--------	---------

Efectos de la instrucción:

$$Lo \leftarrow [R_s]/[R_t]$$

$$Hi \leftarrow [R_s]mod[R_t]$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

`div Rs, Rt`

### 3.6. Unsigned divide: divu.

Idéntica a la instrucción div excepto:

- Func-code = 01 1011
- El contenido de  $R_s$  y  $R_t$  se considera como enteros sin signo.

### 3.7. Addition immediate: addi.

Tipo I:	00 1000	$R_s$	$R_t$	constante
---------	---------	-------	-------	-----------

Efectos de la instrucción:

$$R_t \leftarrow [R_s] + ([I_{15}]^{16} \parallel [I_{15,\dots,0}])$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

addi  $R_t$ ,  $R_s$ , constante

Ejemplo: addi  $R_3$ ,  $R_8$ , 34

## 4. INSTRUCCIONES LÓGICAS

### 4.1. Logical and: and.

Tipo R:	00 0000	$R_s$	$R_t$	$R_d$	0 0000	10 0100
---------	---------	-------	-------	-------	--------	---------

Efectos de la instrucción:

$$R_d \leftarrow [R_s] \text{ and } [R_t]$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

and  $R_d$ ,  $R_s$ ,  $R_t$

### 4.2. Logical or, nor, xor: or, nor, xor.

Idéntica a la instrucción and excepto:

- Func-code = 10 0101 para la instrucción or,
- Func-code = 10 0111 para la instrucción nor,
- Func-code = 10 1000 para la instrucción xor.

### 4.3. Logical and immediate: andi.

Tipo I:	00 1100	$R_s$	$R_t$	constante
---------	---------	-------	-------	-----------

Efectos de la instrucción:

$$R_t \leftarrow [R_s] \text{ and } (0^{16} \parallel [I_{15,\dots,0}])$$

$$PC \leftarrow [PC] + 4$$

Ensamblador:

andi  $R_t$ ,  $R_s$ , constante

#### 4.4. Logical or immediate & xor immediate: ori, xori.

Idéntica a la instrucción andi excepto:

- Func-code = 00 1101 para la instrucción ori.
- Func-code = 00 1110 para la instrucción xori.

### 5. INSTRUCCIONES DE COMPARACIÓN

#### 5.1. Set less than: slt.

<b>Tipo R:</b>	00 0000	$R_s$	$R_t$	$R_d$	0 0000	10 1010
----------------	---------	-------	-------	-------	--------	---------

Efectos de la instrucción:

```
si  $[R_s] < [R_t]$  entonces  $R_d \leftarrow 0^{31} \parallel 1$ 
sino  $R_d \leftarrow 0^{32}$ 
 $PC \leftarrow [PC] + 4$ 
```

Ensamblador:

```
slt Rd, Rs, Rt
```

#### 5.2. Unsigned set less than: sltu.

Idéntica a la instrucción slt excepto:

- Func-code = 10 1011
- El contenido de  $R_s$  y  $R_t$  se considera como enteros sin signo.

#### 5.3. Set less than immediate: slti.

<b>Tipo I:</b>	00 1010	$R_s$	$R_t$	constante
----------------	---------	-------	-------	-----------

Efectos de la instrucción:

```
si  $[R_s] < ([I_{15}]^{16} \parallel [I_{15,...,0}])$  entonces  $R_t \leftarrow 0^{31} \parallel 1$ 
sino  $R_t \leftarrow 0^{32}$ 
 $PC \leftarrow [PC] + 4$ 
```

Ensamblador:

```
slti Rt, Rs, constante
```

#### 5.4. Set less than immediate unsigned: sltiu.

Idéntica a la instrucción slti excepto:

- Func-code = 00 1011.
- El contenido de  $R_s$  y  $R_t$  se considera como enteros sin signo.

### 6. INSTRUCCIONES DE COPIA DE REGISTROS

#### 6.1. Move from Hi: mfhi.

<b>Tipo R:</b>	00 0000	0 0000	0 0000	$R_d$	0 0000	01 0000
----------------	---------	--------	--------	-------	--------	---------

Efectos de la instrucción:

```
 $R_d \leftarrow [Hi]$ 
 $PC \leftarrow [PC] + 4$ 
```

Ensamblador:

`mfhi Rd`

## 6.2. Move from Lo: mflo.

Tipo R:	00 0000	0 0000	0 0000	R <sub>d</sub>	0 0000	01 0010
---------	---------	--------	--------	----------------	--------	---------

Efectos de la instrucción:

$$\begin{aligned}R_d &\leftarrow [Lo] \\PC &\leftarrow [PC] + 4\end{aligned}$$

Ensamblador:

`mflo Rd`

## 7. INSTRUCCIONES DE SALTO

### 7.1. No operation: nop.

Tipo R:	00 0000	0 0000	0 0000	0 0000	0 0000	00 0000
---------	---------	--------	--------	--------	--------	---------

Efectos de la instrucción:

$$PC \leftarrow [PC] + 4$$

Ensamblador:

`nop`

### 7.2. Branch on equal: beq.

Tipo I:	00 0100	R <sub>s</sub>	R <sub>t</sub>	offset
---------	---------	----------------	----------------	--------

Efectos de la instrucción:

si  $[R_s] == [R_t]$  entonces  $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15,\dots,0}] \parallel 0^2)$ ,  
o sea,  $PC \leftarrow [PC] + 4 + 4 * offset$   
sino  $PC \leftarrow [PC] + 4$

Ensamblador:

`beq Rs, Rt, offset`

### 7.3. Branch on not equal: bne.

Tipo I:	00 0101	R <sub>s</sub>	R <sub>t</sub>	offset
---------	---------	----------------	----------------	--------

Efectos de la instrucción:

si  $[R_s] <> [R_t]$  entonces  $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15,\dots,0}] \parallel 0^2)$ ,  
o sea,  $PC \leftarrow [PC] + 4 + 4 * offset$   
sino  $PC \leftarrow [PC] + 4$

Ensamblador:

`bne Rs, Rt, offset`

#### 7.4. Branch on less than or equal zero: blez.

<b>Tipo I:</b>	00 0110	$R_s$	$R_t$	offset
----------------	---------	-------	-------	--------

Efectos de la instrucción:

si  $[R_s] \leq 0$  entonces  $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15,\dots,0}] \parallel 0^2)$ ,  
 o sea,  $PC \leftarrow [PC] + 4 + 4 * offset$   
 sino  $PC \leftarrow [PC] + 4$

Ensamblador:

`blez Rs, offset`

#### 7.5. Branch on greater than zero: bgtz.

<b>Tipo I:</b>	00 0111	$R_s$	$R_t$	offset
----------------	---------	-------	-------	--------

Efectos de la instrucción:

si  $[R_s] > 0$  entonces  $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15,\dots,0}] \parallel 0^2)$ ,  
 o sea,  $PC \leftarrow [PC] + 4 + 4 * offset$   
 sino  $PC \leftarrow [PC] + 4$

Ensamblador:

`bgtz Rs, offset`

#### 7.6. Branch on less than zero: bltz.

<b>Tipo I:</b>	00 0001	$R_s$	$R_t$	offset
----------------	---------	-------	-------	--------

Efectos de la instrucción:

si  $[R_s] < 0$  entonces  $PC \leftarrow [PC] + 4 + ([I_{15}]^{14} \parallel [I_{15,\dots,0}] \parallel 0^2)$ ,  
 o sea,  $PC \leftarrow [PC] + 4 + 4 * offset$   
 sino  $PC \leftarrow [PC] + 4$

Ensamblador:

`bltz Rs, offset`

#### 7.7. Jump: j.

<b>Tipo J:</b>	00 0010	destino
----------------	---------	---------

Efectos de la instrucción:

$PC \leftarrow ([PC_{31,\dots,28}] \parallel [I_{25,\dots,0}] \parallel 0^2)$

Ensamblador:

`j destino`

## 7.8. Jump register: jr.

<b>Tipo R:</b>	00 0000	$R_s$	0 0000	0 0000	0 0000	00 1000
----------------	---------	-------	--------	--------	--------	---------

Efectos de la instrucción:

$$PC \leftarrow [R_s]$$

Ensamblador:

`jr Rs`

## 8. INSTRUCCIONES DE MEMORIA

### 8.1. Load word: lw.

<b>Tipo I:</b>	10 0011	$R_s$	$R_t$	offset
----------------	---------	-------	-------	--------

Efectos de la instrucción:

$$\begin{aligned} R_t &\leftarrow M\{[R_s] + ([I_{15}]^{16} \parallel [I_{15,\dots,0}])\} \\ PC &\leftarrow [PC] + 4 \end{aligned}$$

Ensamblador:

`lw Rt, offset(Rs)`

Ejemplo: `lw R3, 16(R0)`

### 8.2. Store word: sw.

<b>Tipo I:</b>	10 1011	$R_s$	$R_t$	offset
----------------	---------	-------	-------	--------

Efectos de la instrucción:

$$\begin{aligned} M\{[R_s] + ([I_{15}]^{16} \parallel [I_{15,\dots,0}])\} &\leftarrow [R_t] \\ PC &\leftarrow [PC] + 4 \end{aligned}$$

Ensamblador:

`sw Rt, offset(Rs)`

### 8.3. Pop from stack: pop.

<b>Tipo R:</b>	11 1000	0 0000	0 0000	$R_d$	0 0000	00 0000
----------------	---------	--------	--------	-------	--------	---------

Efectos de la instrucción:

$$\begin{aligned} R_d &\leftarrow M\{[SP]\} \\ SP &\leftarrow [SP] + 4 \\ PC &\leftarrow [PC] + 4 \end{aligned}$$

Ensamblador:

`pop Rd`

#### 8.4. Push to stack: push.

<b>Tipo R:</b>	11 1000	$R_s$	0 0000	0 0000	0 0000	00 0001
----------------	---------	-------	--------	--------	--------	---------

Efectos de la instrucción:

$$\begin{aligned} SP &\leftarrow [SP] - 4 \\ M\{[SP]\} &\leftarrow R_s \\ PC &\leftarrow [PC] + 4 \end{aligned}$$

Ensamblador:

```
push Rs
```

### 9. INSTRUCCIONES ESPECIALES

Las siguientes instrucciones no son realistas, pero son necesarias para simular una entrada de teclado y una terminal sin complicar la interfaz de simulación.

#### 9.1. Halt program: halt.

<b>Tipo R:</b>	11 1111	0 0000	0 0000	0 0000	0 0000	11 1111
----------------	---------	--------	--------	--------	--------	---------

Efectos de la instrucción:

Detiene la simulación del programa actual.

Ensamblador:

```
halt
```

#### 9.2. Write to terminal: tty.

<b>Tipo R:</b>	11 1111	$R_s$	0 0000	0 0000	0 0000	00 0001
----------------	---------	-------	--------	--------	--------	---------

Efectos de la instrucción:

Envía un carácter a la pantalla conectada al procesador (TTY). La pantalla es un circuito síncrono. Para enviar un carácter, se colocan los 7 bits menos significativos de  $R_s$  en la salida TTY DATA del procesador, se activa la salida TTY ENABLE, y en el próximo ciclo la pantalla mostrará el carácter *ASCII* correspondiente a los 7 bits de TTY DATA.

Ensamblador:

```
tty Rs
```

#### 9.3. Set random: rnd.

<b>Tipo R:</b>	11 1111	0 0000	0 0000	$R_d$	0 0000	00 0010
----------------	---------	--------	--------	-------	--------	---------

Efectos de la instrucción:

Almacena en  $R_d$  un número aleatorio.

Ensamblador:

```
rnd Rd
```

#### 9.4. Set key: kbd.

<b>Tipo R:</b>	11 1111	0 0000	0 0000	$R_d$	0 0000	00 0100
----------------	---------	--------	--------	-------	--------	---------

Efectos de la instrucción:

Esta instrucción lee un carácter del teclado y lo almacena en  $R_d$ . La entrada KBD AVAILABLE del procesador indica si hay algún carácter esperando en el *buffer* del teclado y la entrada KBD DATA, es un número de 7 bits que corresponde al código *ASCII* del carácter que está en la punta del *buffer*. El *buffer* del teclado funciona como una cola. Los caracteres se añaden a la cola cuando se teclea. En cada ciclo de reloj, si KBD ENABLE está activa, el teclado elimina el carácter que está en la punta de la cola. Si en el momento que esta instrucción se ejecuta, KBD AVAILABLE está desactivada, en  $R_d$  se almacena el valor -1.

Ensamblador:

**kbd  $R_d$**

#### 10. PROGRAM COUNTER

El valor del registro *Program Counter* (*PC*) representa la dirección de memoria de la próxima instrucción que se va a ejecutar. Como en S-MIPS las instrucciones ocupan 4 bytes, cada vez que el procesador ejecuta una instrucción que no sea de salto el valor de *PC* aumenta en 4. Las instrucciones de salto (**beq**, **bne**, **blez**,...) suman su argumento (en complemento a 2) al de *PC*. El argumento indica el número de instrucciones a saltarse, por tanto, es necesario multiplicar el argumento por 4 antes de sumarlo al *Program Counter*. Como resultado de esto, si un programa quisiera saltarse una instrucción, debe hacer un salto con argumento 1:

```
add R3, R0, 46
add R4, R0, 46
beq R3, R4, 1
halt
```

En este ejemplo la instrucción **halt** siempre se salta. Otro resultado de este comportamiento es que “**beq R<sub>0</sub>, R<sub>0</sub>, 0**” es lo mismo que **nop**, y “**beq R<sub>0</sub>, R<sub>0</sub>, -1**” es un ciclo infinito.

#### 11. ACCESO A MEMORIA

Las instrucciones de lectura y escritura en la *RAM* cuentan con 32 bits para direccionar la memoria. Sin embargo, la memoria para el programador de S-MIPS, es un *array* plano de 1 MB =  $2^{20}$  bytes, direccionable a nivel de 1 byte, por lo que solo se usarán los 20 primeros bits para el direccionamiento. Además las transferencias entre la *RAM* y el procesador ocurren siempre en bloques de 16 bytes (65536 bloques).

Por simplicidad para este proyecto académico, cada bloque de la *RAM* cuenta con 4 palabras de 4 bytes, por lo que solo se podrá direccionar a nivel de 4 bytes. Sin embargo, se desea respetar el indizado original (de 1 byte) de S-MIPS. Para ello, el procesador debe ignorar cualquier dirección de memoria que no sea divisible entre 4, e interpretarla como si fuera el número divisible por 4 más cercano por debajo. Por ejemplo, las instrucciones:

- **lw R<sub>3</sub>, 0(R<sub>0</sub>)**,
- **lw R<sub>3</sub>, 1(R<sub>0</sub>)**,
- **lw R<sub>3</sub>, 2(R<sub>0</sub>)**,
- **lw R<sub>3</sub>, 3(R<sub>0</sub>)**,

hacén lo mismo: copiarán los bytes 0, 1, 2 y 3 de la *RAM* al registro R<sub>3</sub>. En el caso de:

**sw R<sub>3</sub>, 0(R<sub>0</sub>)**,

se copiará a los bytes 0, 1, 2 y 3 de la *RAM* el valor guardado en el registro  $R_3$ .

S-MIPS es una arquitectura *Little-Endian*, lo que significa que, en cada palabra de 32 bits, el byte con dirección más pequeña es el menos significativo y el de dirección más grande es el más significativo. Esto luce “*al revés*” de cómo se escriben normalmente los números. Supongamos que los primeros bytes de la *RAM* tienen estos valores:

0	1	2	3	4	5	
1	2	0	0	1	2	...

Al hacer “`lw R3, 0(R0)`”, el byte 0 de la *RAM*, que tiene el valor 1 ( $0000\ 0001_2$ ) va hacia la parte menos significativa de  $R_3$  y el byte 3, con valor 0 ( $0000\ 0000_2$ ), va hacia la más significativa. El valor final de  $R_3$  sería:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0010\ 0000\ 0001_2 = 512_{10} + 1_{10} = 513_{10}.$$

Las escrituras se comportan de igual forma. Si se hace:

```
add R3, R0, 1027
sw R3, 0(R0)
```

se escribirán para el byte 0 de la *RAM* la parte menos significativa del valor 1027 y para el byte 3 la parte más significativa. Como 1027 tiene la siguiente representación binaria:

$$1027_{10} = 1024_{10} + 2_{10} + 1_{10} = 0000\ 0000\ 0000\ 0000\ 0100\ 0000\ 0011_2,$$

su byte menos significativo contiene  $0000\ 0011_2 = 3_{10}$ , y su byte más significativo contiene  $0000\ 0000_2 = 0_{10}$ , quedando en la *RAM*:

0	1	2	3	4	5	
3	4	0	0	1	2	...

## 12. INTERFAZ CON LA MEMORIA

El módulo *RAM* del circuito *S-MIPS Board* implementa una memoria asíncrona de 1 MB. Esta memoria está organizada en  $2^{16} = 65536$  bloques de 16 bytes. Esta *RAM* es más lenta que el procesador, y le toma varios ciclos leer y escribir datos. La cantidad de ciclos que toma hacer una lectura o una escritura están en las salidas **RT** (Read Time) y **WT** (Write Time). Estos valores no cambian una vez comienza la ejecución del procesador.

Aunque estas salidas sean constantes, no se deben usar sus valores “*a mano*” dentro del *CPU*. Hay que obtenerlos de la *RAM*. Los profesores vamos a probar distintas combinaciones de esos valores durante la verificación del procesador y este debe comportarse en concordancia.

La entrada **ADDR** (16 bits) indica el bloque de la *RAM* que se quiere indizar. La entrada **CS** (Chip Select) indica si se va a utilizar la *RAM* ( $CS = 1$ ), o no ( $CS = 0$ ). La entrada  $\neg R/W$  indica, en el caso de que  $CS = 1$ , si se leerá de la *RAM* ( $\neg R/W = 0$ ), o si se escribirá en la *RAM* ( $\neg R/W = 1$ ). Cuando  $CS = 0$ , se ignora el valor de  $\neg R/W$ .

En el momento que se desee acceder a la *RAM*, se debe activar **CS** y especificar el tipo de operación con  $\neg R/W$ , manteniendo persistentes estos valores durante **RT** o **WT** ciclos de la *CPU* (si se lee o se escribe respectivamente). Cuando se finaliza una lectura se proporcionan los 16 bytes del bloque seleccionado, a través las 4 salidas de 32 bits  $O_0$ ,  $O_1$ ,  $O_2$ , y  $O_3$ . Análogamente, cuando se va a realizar una escritura se deben enviar los valores por las entradas de 32 bits  $I_0$ ,  $I_1$ ,  $I_2$ , e  $I_3$ . Sin embargo, la escritura puede hacerse parcialmente, usando la entrada **MASK**.

La *RAM* está dividida en 4 bancos que actúan como columnas o *slices*. Cada bloque de 16 bytes de la *RAM* está dividido en 4 palabras de 4 bytes (32 bits). Los primeros 64 bytes de la *RAM* lucen así:

Banco 0	Banco 1	Banco 2	Banco 3	
00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	← Bloque 0
16 17 16 19	20 21 22 23	24 25 26 27	28 29 30 31	← Bloque 1
32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	← Bloque 2
48 49 50 51	52 53 54 55	56 57 58 59	60 61 62 63	← Bloque 3
:	:	:	:	:

La entrada **MASK** es una entrada de 4 bits que selecciona cuál o cuáles bancos van a ser modificados por una operación de escritura. El bit menos significativo de **MASK** selecciona el Banco 0, el más significativo selecciona el Banco 3. Por ejemplo, si la entrada **MASK** tiene el valor  $8_{10}$  ( $1000_2$ ) y la entrada **ADDR** es 0, la escritura solo afectaría los bytes 12, 13, 14 y 15 de la *RAM*, pues estos están en el bloque 0 y en el banco 3. En esos 4 bytes se escribiría el valor de la entrada **I<sub>3</sub>** (**Data Input 3**). Si la entrada **MASK** fuera  $12_{10}$  ( $1100_2$ ) y **ADDR** fuera 2, la escritura modificaría los bytes 40, 41, 42, 43 y 44, 45, 46, 47 (bloque 2, bancos 2 y 3). En esos 16 bytes se escribirían los valores de las entradas **I<sub>2</sub>** e **I<sub>3</sub>**.

La entrada **MASK** no afecta las operaciones de lectura. Las salidas **O<sub>0</sub>**, **O<sub>1</sub>**, **O<sub>2</sub>**, y **O<sub>3</sub>** siempre contienen el bloque completo solicitado en **ADDR**. Las escrituras se realizan cuando **CS** es 1 y  $\neg R/W$  es 1 y toman la cantidad de ciclos de *CPU* que indica la salida **WT**.

### 13. CACHÉ

La implementación del procesador S-MIPS debe incluir una caché para acelerar el proceso de lectura y/o escritura en la RAM. Se puede aprovechar el hecho de que a partir de una sola lectura de la RAM se obtiene un bloque completo de  $4 \times 32$  bits.

**IMPORTANTE:** Para optar por la nota máxima de 5 puntos debe tener en cuenta lo siguiente:

1. En la Primera Convocatoria de Entrega y Revisión (Ordinario) debe tener como mínimo una sola caché de mínimo cuatro líneas. De no cumplir esto la nota es en base a 4.
2. En la Segunda Convocatoria de Entrega y Revisión (Extraordinario) debe tener dos memorias caché, una para datos y otra para código. Ambas *cachés* pueden ser la misma implementación, siempre y cuando tengan cuatro líneas como mínimo. De no cumplir esto la nota es en base a 4.
3. En la Tercera Convocatoria de Entrega y Revisión (Mundial) debe tener dos memorias caché, una para datos y otra para código. Ambas *cachés* pueden ser la misma implementación, siempre y cuando tengan cuatro líneas como mínimo, y además implemente una función de *mappeo* distinta de **Direct Mapping**. De no cumplir esto la nota es en base a 4 si tiene las cachés, y en base a 3 sino no las tiene.