UNICEN
Universidad Nacional del Centro de la Provincia de Buenos Aires

Arquitectura de Computadoras I

Ingeniería de Sistemas – Curso 2017

Práctica de Laboratorio: Microprocesador MIPS Segmentado

El objetivo de esta práctica es implementar el microprocesador MIPS (visto en clase de teoría) en VHDL. En concreto, se va a realizar la versión **segmentada** del microprocesador, cuyos detalles se pueden encontrar en: "Computer Organization and Design: The Hardware/Software Interface ", por David A.Patterson y John L. Hennessy. Se recomienda seguir el libro para realizar esta práctica, ya que se sigue este libro con bastante fidelidad. En concreto, se sigue el modelo segmentado del MIPS, detallado en el capítulo 4.

El modelo de las memorias de datos y programas proporcionado (archivo memory.vhd) no introduce ciclos de espera y responde en el mismo ciclo. Además, utiliza dos archivos separados para el contenido inicial de cada memoria, archivo llamado “program1” para memoria de instrucciones y “data” para memoria de datos. Se proporcionan un archivo de ejemplo (program1.s) para utilizar junto con el testbench de la práctica, si bien se pueden generar otros archivos correspondientes a otros códigos para hacer más pruebas.

Se pide implementar el microprocesador MIPS en su versión segmentada. El resultado debe ser un micro capaz de realizar en el caso ideal (sin riesgos) una instrucción por ciclo de reloj. Para el ejercicio básico, no es necesario que el modelo soporte riesgos (salvo el estructural de acceso a memorias separadas de instrucciones y datos).

No se pide que el microprocesador soporte todo el juego de instrucciones completo, sino solamente las siguientes instrucciones: *add*, *sub*, *and*, *or*, *lw*, *sw*, *slt*, *lui*, *beq, jr* y *jal*. En cualquier caso, la instrucción *beq*, que implica riesgos de control por ser un salto, funcionará “anómalamente” en la versión básica del ejercicio obligatorio.

Se recomienda utilizar el esquema del libro, reflejado en la siguiente figura 1. Todos los registros deben resetearse asíncronamente y funcionar por flanco de subida del reloj. Las instrucciones a implementar tienen los siguientes códigos de operación. Los códigos de operación se pueden encontrar también en el apéndice A del libro (volumen 3).

ADD (Add Word)

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
SPECIAL 0 0 0 0 0 0	**rs**	**rt**	**rd**	0 0 0 0 0 0	**ADD** 1 0 0 0 0 0
6	5	5	5	5	6

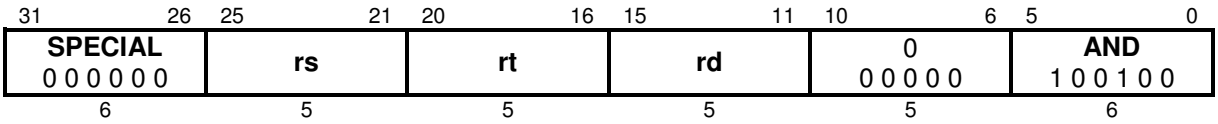
Formato: ADD rd, rs, rt
Descripción: rd ← rs + rt

SUB (Substract Word)

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
SPECIAL 0 0 0 0 0 0	**rs**	**rt**	**rd**	0 0 0 0 0 0	**SUB** 1 0 0 0 1 0
6	5	5	5	5	6

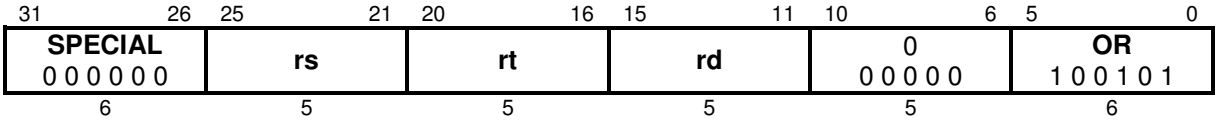
Formato: SUB rd, rs, rt
Descripción: rd ← rs - rt

AND



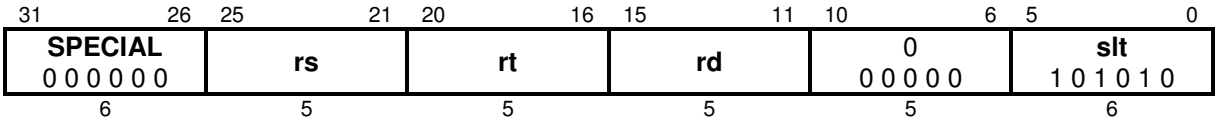
Formato: AND rd, rs, rt
Descripción: rd ← rs AND rt

OR



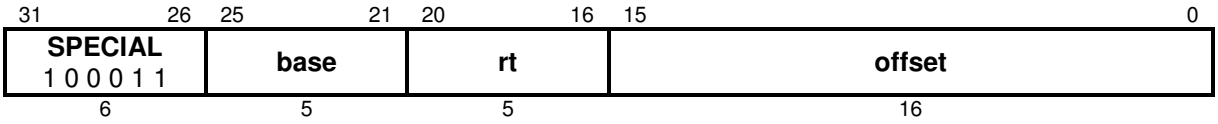
Formato: OR rd, rs, rt
Descripción: rd ← rs OR rt

SLT (Set on Less Than)



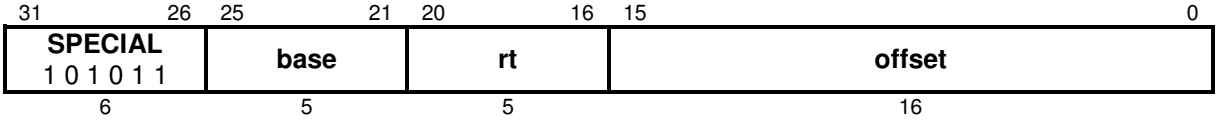
Formato: SLT rd, rs, rt
Descripción: rd ← (rs < rt)

LW (Load Word)



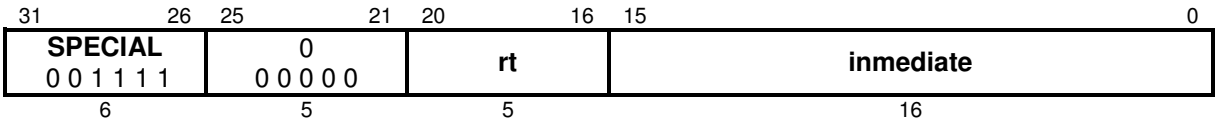
Formato: LW rt, offset(base)
Descripción: rt ← memory[base+offset]

SW (Store Word)



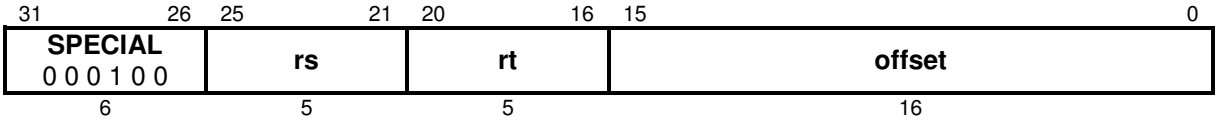
Formato: ST rt, offset(base)
Descripción: memory[base+offset] ← rt

LUI (Load Upper Inmediate)



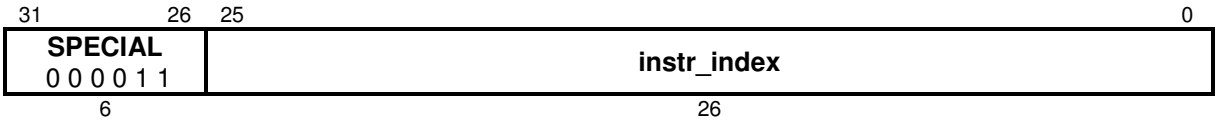
Formato: LUI rt, inmediate
Descripción: rt ← immediate & 0^{16} (rt ← immediate << 16)

BEQ (Branch on Equal)



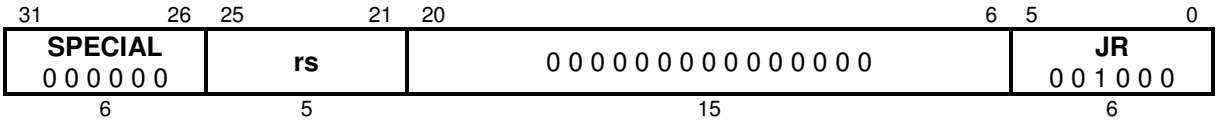
Formato: BEQ rs, rt, offset
Descripción: if (rs=rt) then PC ← PC + (offset << 2)

JAL (Jump And Link)



Formato: JAL target
Descripción: R[31] ← PC + 8
PC ← PC(31:28) & instr_index & 00

JR (Jump Register)



Formato: JR rs
Descripción: PC ← R[rs]

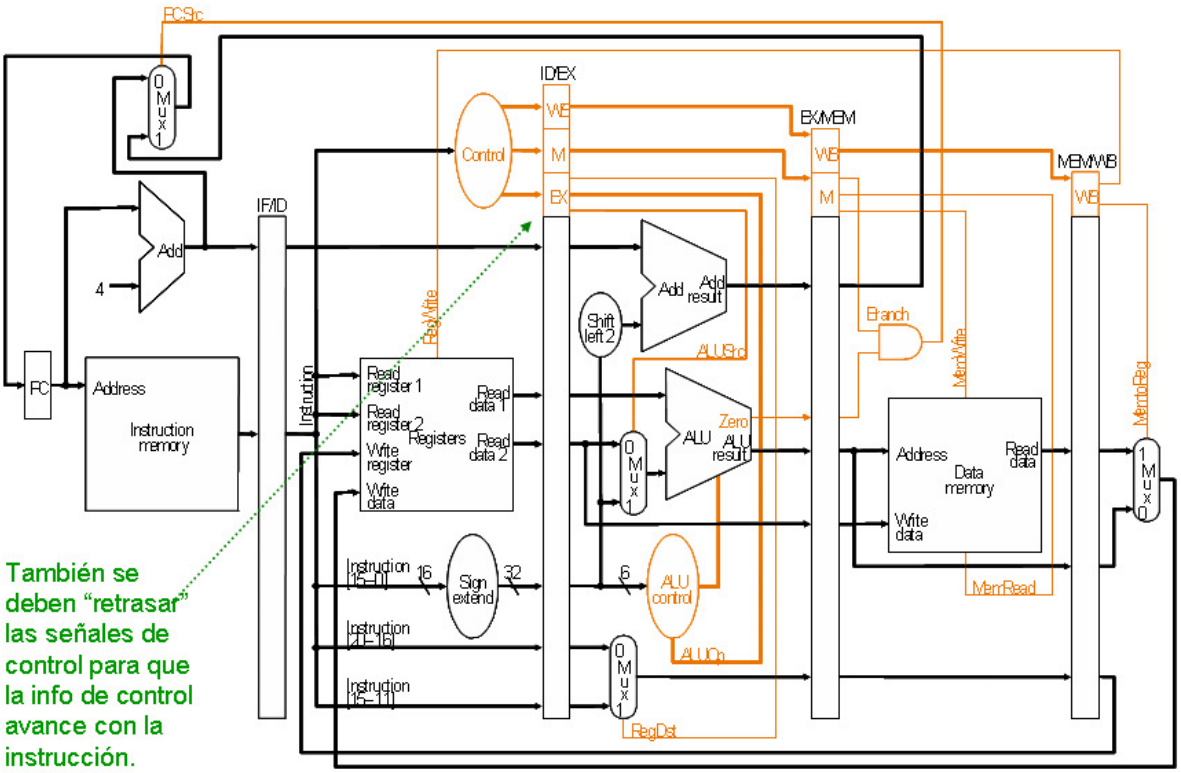


Figura 1. Modelo de microprocesador segmentado (faltan instrucciones JR y JAL).

Ejercicios

1. Verificar el diseño utilizando el archivo "program1" proporcionado por la cátedra.

2. Compilar y verificar el diseño utilizando el archivo "bit_counter.s" proporcionado por la cátedra. El programa implementa un algoritmo que cuenta el número de bits en estado alto ('1') de los 8 bits menos significativos de un dato almacenado en la dirección 4 de memoria. El programa sólo utiliza 3 registros del procesador y utiliza la memoria de datos para almacenamiento de datos temporales. El estado inicial de la memoria de datos es el siguiente:

Dirección 0x0000 0000:	0x000000001
Dirección 0x0000 0004:	0x0000000YY (YY = dato)

   El resultado se almacena en la posición 8 de la memoria.
   Se debe considerar que el programa posee riesgos de datos y de control, por lo cual deben incluirse instrucciones de espera (*nop*) para el correcto funcionamiento en la arquitectura. El número de ciclos de espera debe ser el mínimo.

3. Realizar una nueva versión del programa utilizando las instrucciones *jal* y *jr*. ¿Que ventajas presenta el uso de estas instrucciones?

4. ¿Qué desventaja introduce al procesador si la instrucción **JAL** realiza *R[31]* ← *PC + 4* ?

Material a entregar

- Archivos VHDL
- Program2.s corregido
- Archivos de memoria de instrucción y de datos

Ayudas y avisos

- Los archivos “programa” y “datos” que contienen las memorias de instrucciones y datos respectivamente deben estar en el directorio de trabajo. Si no es así, en el archivo procesador_TB.vhd se puede dar la ruta completa de dichos archivos cambiando las líneas de código:

```
C_ELF_FILENAME     => "programa",
...
C_ELF_FILENAME     => "datos",
```

  por otras donde indique la ruta completa a ambos archivos, por ejemplo:

```
C_ELF_FILENAME     => "D:\nombredirectorio\programa",
...
C_ELF_FILENAME     => "D:\nombredirectorio\datos",
```

- El programa de prueba “program.s” proporcionado en la práctica no incluye riesgos y prueba todas las instrucciones del ejercicio básico. El archivo “programa” es el resultado del ensamblado del “program.s” que se usará para probar el ejercicio básico. El archivo “datos” contiene los datos que se usaran por el “programa” en el ejercicio básico.

- El archivo *memory.vhd* contiene la memoria que se usará en el ejercicio básico. El archivo *processor.vhd* contiene la entidad del micro que se deberá implementar. El archivo *processor_tb.vhd* contiene el test bench para probar el ejercicio básico.

- La tabla contenida en el archivo “registers.html” proporcionada en la práctica muestra la traducción de los nombres de registros usados en ensamblador al número de registro en el micro, del 0 al 31.