Práctica 1: Componentes Microprocesador MIPS Segmentado

El objetivo de esta práctica es implementar varios componentes del microprocesador MIPS (visto en clase de teoría) en VHDL. Detalles de este microprocesador se pueden encontrar en el libro: "Estructura y Diseño de Computadores: interfaces circuitería/programación", 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 6 (volumen 2).

En la práctica 2 se podrán usar los componentes desarrollados aquí para construir un microprocesador MIPS, versión segmentada. Se debe usar siempre los tipos STD_LOGIC o STD_LOGIC_VECTOR para las entradas, salidas, señales y variables en las entidades.

IMPORTANTE: Los nombres de los archivos que se entregarán deben ser del estilo:

- P1X.VHD para el modelo pedido en el ejercicio.
- P1X_TB.VHD para su banco de pruebas (testbench).

Donde X es la letra del ejercicio. Por ejemplo, para el ejercicio c se entregarán dos archivos, P1C.VHD y P1C_TB.VHD. Se deberán usar los nombres especificados para las entidades y sus entradas/salidas. En el caso de los bancos de pruebas, como nombre de la entidad se usará el mismo que el del archivo (ejemplo: entity P1C_TB is...). Los nombres de las arquitecturas, tanto para el diseño como para el banco de pruebas, serán siempre PRACTICA. Es indispensable para la corrección respetar estas normas.

Ejercicios

a. Registro síncrono de 32 bits con reset síncrono y habilitación de reloj. Reloj activo en flanco de subida, reset activo a nivel alto y habilitación activo a nivel bajo. La señal de reset tiene prioridad sobre las demás.

   Entidad: Reg. Entradas: d(31 downto 0)**, clk, reset, ce. Salida:** q(31 downto 0). Probar el modelo con el testbench dado en la práctica.

b. Mutiplexor de 2 a 1, de 32 bits.

   Entidad: *Mux*. **Entradas:** *a*(31 downto 0), *b*(31 downto 0), se*l*. **Salida:** *q*(31 downto 0). Probar el modelo con el testbench dado en la práctica.

c. ALU del procesador MIPS. Esta ALU opera con datos de 32 bits, dispone del flag zero para indicar si el resultado es igual a cero. Sus operaciones aritmético/lógicas básicas son suma, resta, and, or, “menor que” y desplazamiento a la izquierda. En el caso de introducir un código de operación (control) que no corresponda con los prefijados en la tabla, la señal de salida result se pondrá a 0. Los códigos de operación se corresponderán con la siguiente tabla:

Control(2 downto 0)	Operación
000	A and B
001	A or B
010	A + B
110	A – B
111	A < B
100	B << 16 (logico)

La ALU debe ser completamente combinacional.

El flag "*zero*" se pondrá a 1 cuando el resultado de la operación es igual a cero. En caso contrario debe permanecer siempre a 0.

El código de operación 111 pondrá en la salida "*result*" un 1 si A es menor que B y un 0 en caso contrario.

Entidad: ALU. **Entradas**: *a*(31 downto 0), *b*(31 downto 0) y *control*(2 downto 0). **Salidas**: *result*(31 downto 0) y *zero*.

d. Banco de registros. En el procesador MIPS existe un banco de registros de 32 registros de propósito general, de 32 bits cada uno, de los cuales el primero (r0) está siempre a 0. Este banco permite el acceso simultáneo a tres registros, cuyas direcciones son: *reg1_rd*, *reg2_rd* y *reg_wr*. El acceso a los registros direccionados por *reg1_rd* y *reg2_rd* será de lectura, mientras que el asociado a *reg_wr* será de escritura. La escritura será síncrona, en flanco de bajada y habilitada con la señal *wr* a 1, mientras que la lectura será siempre combinacional. El reset del circuito es asíncrono y activo a nivel alto.
Entidad: *Registers*. **Entradas**: *clk*, *reset*, *wr*, *reg1_rd* (4 *downto* 0), *reg2_rd* (4 downto 0), *reg_wr*(4 downto 0) y *data_wr*(31 *downto* 0). **Salidas**: *data1_rd* (31 *downto* 0) y *data2_rd* (31 *downto* 0).
Empleando pseudocódigo, su funcionalidad se puede describir como:

```
If reg1_rd I= 0 then data1_rd <= Regs(reg1_rd) else data1_rd <= x"00000000"
If reg2_rd I= 0 then data2_rd <= Regs(reg2_rd) else data2_rd <= x"00000000"
If falling_edge(clk) and wr='1' then Regs(reg_wr) <= data_wr;
```