Guía práctica nro: 09 - VHDL - RISC-V

NOTA: Como en la guía anterior, se sugiere repasar las presentaciones de las clases de VHDL de Electrónica digital. Y también reutilizar código de los Language Templates del ISE Design de Xilinx.
Esto sin desmedro de los textos Guia practica de RISC-V de Patterson y Waterman, y Computer Organization and Design - The Hardware Software Interface [RISC-V Edition] de Patterson y Hennessy.

1. Desarrollo de las arquitecturas de los módulos de memoria.

1.1. Diseñe la arquitectura del módulo memInst (ROM)

Para crear una memoria rom, qué es de solo lectura, usamos un arreglo de vectores.[1] También recomiendo hacer un testbench. Un fragmento de código de ejemplo se encuentra en el código 1.

```
architecture Behavioral of memInstr is
type rom_type is array (0 to 15) of std_logic_vector (15 downto 0);
signal myROM : rom_type:= (X"200A", X"0300", X"8101",
                           X"0300", X"8602", X"2310",
                           X"8201", X"0500", X"4001",
                           others => X"0000");
begin
dataOut <= myROM(to_integer(unsigned(address)));
end Behavioral;
```

Código 1 - Código VHDL - memoria ROM

1.2. Diseñe la arquitectura del módulo memData (RAM)

Para crear una memoria ram, qué es de lectura y escritura, la escritura está sincronizada y debe tener su habilitación pertinente. Aquí también usamos un arreglo de vectores. Y también recomiendo hacer un testbench. Un fragmento de código de ejemplo se encuentra en el código 2.

```
architecture Behavioral of memData is
        type ram_type is array (0 to 15) of std_logic_vector (15 downto 0);
        signal myRAM : ram_type;
begin
   --Escritura
   process (clk, we, address, dataIn)
   begin
      if (clk'event and clk = '1') then
         if (cs = "01") then
            if (we = '1') then
                  myRAM(to_integer(unsigned(address))) <= dataIn;
            end if;
         end if;
      end if;
  end process;
  --Lectura
  dataOut <= myRAM(to_integer(unsigned(address)));
end Behavioral;
```

Código 2 - código VHDL - memoria Ram

[1] clases de VHDL 1 y 6 de Electrónica digital

2. Desarrollo de las arquitecturas de los módulos del camino de datos.

2.1. Diseñe la arquitectura del módulo registerBank

Ver Clase VHDL 6 de Electrónica Digital.

2.2. Diseñe la arquitectura del registro pc

Ver los Language Templates del ISE Design de Xilinx.

2.3. Diseñe la arquitectura de los multiplexores

Ver los Language Templates del ISE Design de Xilinx.

2.4. Diseñe la arquitectura de los signExt

Ver Guía 08 - Tabla 1 - Bits del campo inmediato; y los Language Templates del ISE Design de Xilinx.

2.5. Diseñe la arquitectura de la alu

Para el diseño de la **alu** debemos tomar en cuenta previamente cuáles operaciones realizaremos. En nuestro caso no vamos a realizar la multiplicación ni la división ya qué nuestra base es el RV32I. Además como nuestro diseño es inicialmente didáctico, nos limitaremos, por ahora, a las aritméticas Suma, Resta; las lógicas And, Or y la de comparación SLT.

Por último recuerden qué tienen una bandera *zero*, activa alto.

Para controlar estas operaciones respetamos la tabla 1.

$ALUControl_{2:0}$	Function
000	A & B
001	A | B
010	A + B
110	A - B
111	SLT

Tabla 1 - Términos de control de la ALU

3. Desarrollo de las arquitecturas del módulo de control.

3.1. Diseñe la arquitectura de la unitCtrl.

El módulo **unitCtrl** posee una entrada *ctrlIn* en bus de 9 bits y una salida *ctrlOut* en bus de 15 bits. Este módulo toma datos de la instrucción para setear en sus salidas las señales qué permiten dar sentido al camino de datos de acuerdo con dicha instrucción actual.

Como nuestro diseño es monociclo, nuestra unidad de control se resuelve como un simple combinacional. Para facilitar el diseño más aun, se puede optar por dividir el módulo en dos submódulos más específicos; el **mainDeco** y el **aluDeco.**

Como recordaran de Electrónica Digital la especificación de los combinacionales se puede hacer de acuerdo a una tabla.

3.2. Diseñe el submódulo mainDeco

El submódulo **mainDeco** reconoce en su entrada la instrucción, en particular vamos a distinguir el tipo de instrucción, pues como ya vimos un tipo involucra varias instrucciones similares.

De acuerdo con la Guia 08 recorrimos el camino de datos para las instrucciones de tipo R, Lw(I), Sw(S) y SB, está es nuestra entrada. Nuestras salidas corresponden con las señales de control para los componentes usados. Es decir qué el bus ctrlOut contiene las señales:

- *regWrite* de 1 bit, qué conecta con *we* del **registerBank**. para habilitar la escritura.
- InmSrc de 2 bits qué conecta con el selector del **signExt.**
- aluSrc de 2 bits, qué conecta con *sel* del **mux4x1**. para habilitar el operando 2.
- memWrite de 1 bit qué conecta con *we* de **memData**. para habilitar la escritura.
- resultSrc de 1 bit qué conecta con *sel* del **mux2x1**. para ingresar el *registerDestiny*.
- branch de 1 bit, qué conecta con *sel* del **mux2x1x16**. para ingresar el *pcNext*.

También tenemos una salida intermedia:

- aluOp de 2 bits, qué conecta con **aluDeco**.

Ver la tabla 2.

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	`lw`	1	00	1	0	1	0	00
35	`sw`	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	`beq`	0	10	0	0	X	1	01

Tabla 2 - Señales de mainDeco

3.3. Diseñe el submódulo aluDeco

El submódulo **aluDeco** reconoce las entradas de los campos *f3,* bit 5 de *f7,* el bit 5 de la instrucción y la señal *aluOp* del **mainDeco**. Con ellas determina los 3 bits de salida *aluCtrl* para la **alu.** Ver la tabla 3.

ALUOp	op_5	funct3	$funct7_5$	Instruction	$ALUControl_{2:0}$
00	X	X	X	`lw, sw`	010 (add)
01	X	X	X	`beq`	110 (subtract)
10	X	000	0	`add`	010 (add)
	1	000	1	`sub`	110 (subtract)
	X	010	0	`slt`	111 (set less than)
	X	110	0	`or`	001 (or)
	X	111	0	`Slt`	000 (and)

Tabla 3 - Señales de aluDeco

4. **Diseñe los bancos de prueba de los módulos en general**
   Una vez concretado el diseño de los módulos resulta imprescindible realizar test bench de cada uno de ellos pues así evitamos arrastrar errores qué posteriormente son difíciles de encontrar. En la presentación de la clase 2 de VHDL de electrónica digital, hice hincapié en un ejemplo de tests de autocomprobación. Sin embargo en algunos casos la sentencia for loop resulta más útil para situaciones donde se requiere un uso intensivo.

```
-- insert stimulus here
--valor de configuración
entrada1 <= '1';
--valor de estímulos
for i in 0 to 15 loop
        entrada2<=(std_logic_vector(to_unsigned(i,8)));
        wait for 5 ns;
end loop;
wait;
end process;
```

5. **Diseñe el camino de datos para la instrucción addi y la instrucción jal**
   Diseñe los módulos necesarios, modifique los caminos y las señales necesarias a fin de poder procesar dichas señales.
