DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2019)

Entrega 2

Entrega: Lunes 14 de Octubre | 10:59:59 a.m.

Requisitos

- Esta entrega es de carácter grupal. Cualquier tipo de falta a la honestidad académica será sancionada con la **reprobación** del curso con la nota mínima.
- Los nombre de archivos y el cómo deben ser ejecutados son parte del formato, no respetarlo será penalizado.
- El programa de la **placa** deberá ser realizado en `VHDL`.
- La **documentación** deberá ser realizada en un archivo `Markdown` y subirlo junto a su tarea, de nombre README.md, en el mismo repositorio.
- Esta entrega deberá ser subida a su repositorio grupal de *GitHub* antes de la fecha y hora dada.
- **La entrega de la placa debe ser realizada previa o al inicio de la hora de ayudantía del curso. El no cumplimiento, no solo perjudicará su nota, sino también a sus compañeros**

Objetivo

Para esta entrega tendrán que diseñar y armar su propio computador básico de 16 bits, el que utilizarán posteriormente para ejecutar programas hechos en lenguaje *assembly*. Lo que deberán armar en Vivado es su propia CPU, la cual deberá cumplir con una serie de requisitos descritos más abajo, dentro de los cuales se encuentra soportar una lista determinada de instrucciones en *assembler*.

Adicionalmente, deberán escribir en lenguaje máquina un programa en la ROM de la implementación de su computador básico que multiplique dos números cualquiera (que no sea el algoritmo de multiplicación rusa visto en clases) y deberán entregar un informe grupal sobre su trabajo en esta entrega. A continuación, los detalles correspondientes:

Especificaciones CPU

Sin entrar todavía en detalles, el computador que deberán armar debe contar con:

- Dos registros de 16 bits (registros A y B), los que deben tener sus ocho bits menos significativos conectados al Display de la placa.
- Una memoria ROM de instrucciones de 4096 palabras de 36 bits cada una.
- Una memoria RAM de 4096 palabras de 16 bits cada una. Esta memoria es de lectura asíncrona y escritura síncrona.
- **Una unidad aritmético lógica** (ALU) capaz de realizar ocho operaciones (Se especificarán mas abajo).
- Un *program counter* (PC) de 12 bits de direccionamiento.
- Dos multiplexores de cuatro entradas de 16 bits.
- Una **unidad de control** de lógica combinacional (Control Unit).
- Un registro de status (C, Z, N).

Se deben tener en cuenta las siguientes indicaciones para la implementación de los componentes de su computador básico en Vivado:

- **Unidad Aritmética Lógica:** No puede utilizar operadores matemáticos de Vivado. Este componente debe ser construido **solo con half adders y compuertas lógicas**. Se recomienda encarecidamente revisar con cuidado la tabla de verdad de este componente.
- **Unidad de Control:** La palabra de control debe ser diseñada por ustedes. Esta debe tener un diseño que tenga sentido y no puede estar construida solo por multiplexores.

En la Figura 1 se muestra el diagrama del computador básico recién descrito, donde se pueden apreciar cada uno de los bloques mencionados junto con las interconexiones pertinentes (además de la indicación del tamaño de cada bus). Cabe mencionar que todos los bloques síncronos de la Figura 1 son de flanco de subida.

Nota: Tomar en cuenta que este esquema es solo representativo y requiere modificaciones leves para implementar toda la ISA requerida para esta entrega.

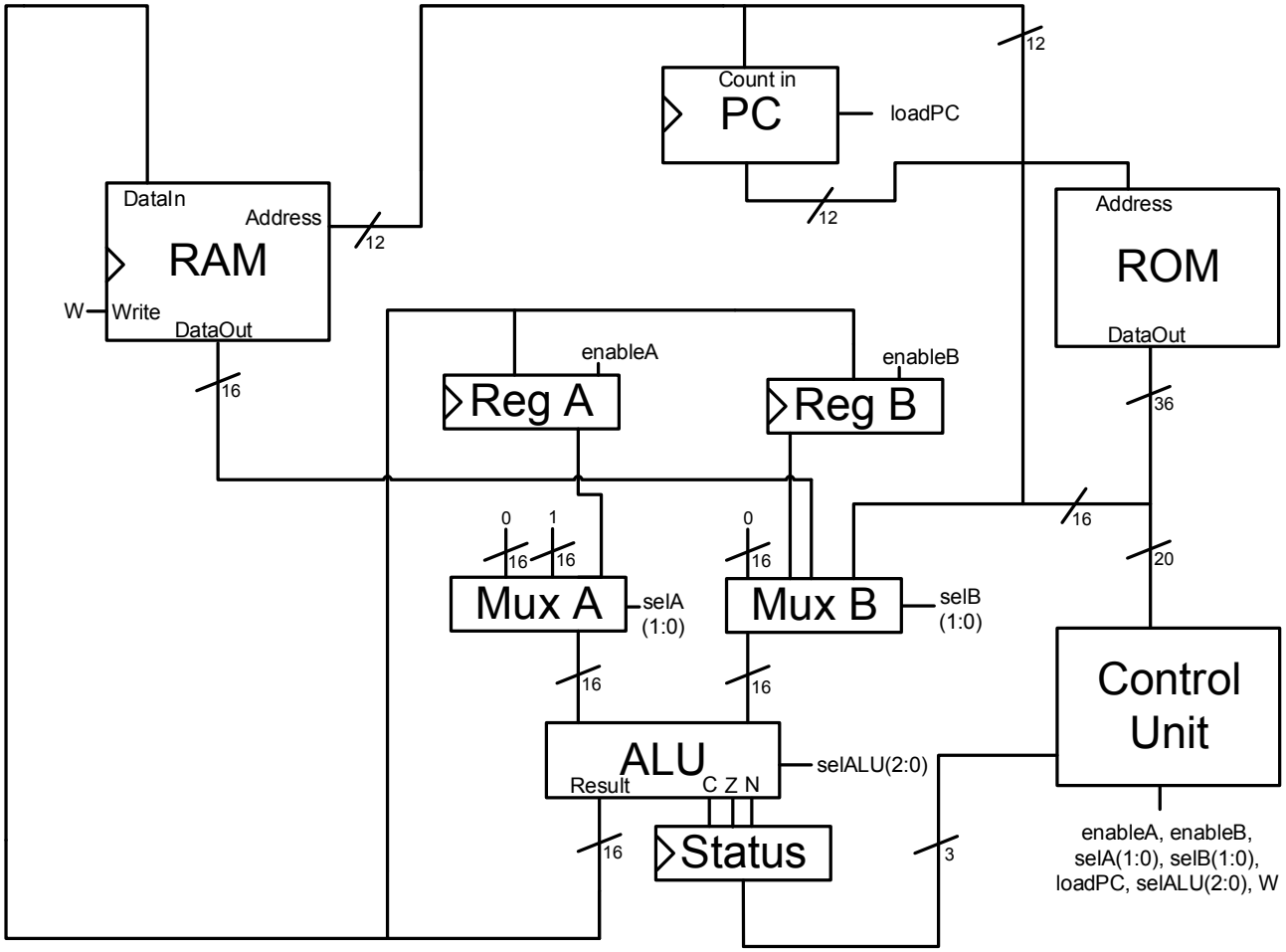

Figura 1: Computador básico.

A continuación, la lista de instrucciones que su computador básico deberá soportar:

Entrega 2

MOV	A,B	guarda B en A
	B,A	guarda A en B
	A,Lit	guarda un literal en A
	B,Lit	guarda un literal en B
	A,(Dir)	guarda Mem[Dir] en A
	B,(Dir)	guarda Mem[Dir] en B
	(Dir),A	guarda A en Mem[Dir]
	(Dir),B	guarda B en Mem[Dir]
ADD SUB AND OR XOR	A,B	guarda A op B en A
	B,A	guarda A op B en B
	A,Lit	guarda A op literal en A
	B,Lit	guarda A op literal en B
	A,(Dir)	guarda A op Mem[Dir] en A
	B,(Dir)	guarda A op Mem[Dir] en B
	(Dir)	guarda A op B en Mem[Dir]
NOT SHL SHR	A	guarda op A en A
	B,A	guarda op A en B
	(Dir),A	guarda op A en Mem[Dir]
INC	A	incrementa A en una unidad
	B	incrementa B en una unidad
	(Dir)	incrementa Mem[Dir] en una unidad
DEC	A	decrementa A en una unidad
CMP	A,B	hace A-B
	A,Lit	hace A-Lit
	A,(Dir)	hace A-Mem[Dir]
JMP	Ins	carga Ins en PC
JEQ	Ins	carga Ins en PC si en el status Z = 1
JNE	Ins	carga Ins en PC si en el status Z = 0
NOP		Sin acciones

Módulos entregados

Se les entregarán módulos VHDL para facilitar su entrega. A continuación, una lista de los módulos que deben ser usados de manera obligatoria:

- `Display_Controller.vhd`

  Este módulo recibe 4 vectores y un valor lógico: `dis_a[3:0]`, `dis_b[3:0]`, `dis_c[3:0]`, `dis_d[3:0]` y `clk`. Entrega como salida los vectores `seg[7:0]` y `an[3:0]`.

  Su función es decodificar los valores que quieres entregar en los display 7 segmentos correspondientes:

  

  Para que este módulo funcione, debe conectar la señal de entrada de la Basys3 `clk`, a la señal de entrada `clk` de este módulo.

  La salida de este módulo genera las señales de salida de la Basys3: `seg[7:0]` y `an[3:0]` para encender el display.

  Como se indicó anteriormente, los regitros A y B deberán estar conectados al Display. El registro A deberá mostrar sus 8 bits en base hexadecimal utilizando los Display A y B de la placa, en cambio, el registro B deberá mostrar sus ocho bits menos significativos en base hexadecimal en los Display C y D de la placa.

- `ROM.vhd`

  Este módulo recibe un vector de doce bits, que representará una dirección de memoria, y entrega como salida un vector de 36 bits, que representará el valor guardado en dicha dirección. Dentro de este componente habrá una arreglo de vectores, que representaran una memoria de lectura.

  Los valores de dicho arreglo son explicitados, y serán utilizados para representar las instrucciones del programa en su código de máquina. A continuación, un ejemplo de la ROM de 4096 palabras de 36 bits que tiene solo 9 instrucciones:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  USE IEEE.NUMERIC_STD.ALL;
5
6  entity ROM is
7      Port (
8          address   : in  std_logic_vector(11 downto 0);
9          dataout   : out std_logic_vector(35 downto 0)
10           );
11 end ROM;
12
13 architecture Behavioral of ROM is
14
15 type memory_array is array (0 to ((2 ** 12) - 1) ) of std_logic_vector (35 downto 0);
16
17 signal memory : memory_array:= (
18         "000000000000000000010000111000000010",    -- instruccion 1
19         "000000000000000000000000011000000011",    -- instruccion 2
20         "000000000000000001000000111000000010",    -- instruccion 3
21         "000000000000000001000000011000000011",    -- instruccion 4
22         "000000000000000001100000111000000010",    -- instruccion 5
23         "000000000000000001000000011000000011",    -- instruccion 6
24         "000000000000000010000000111000000010",    -- instruccion 7
25         "000000000000000001100000011000000011",    -- instruccion 8
```

```
26          "00000000000000000000000111000000010",    -- instruccion 9
27          "00000000000000000000000000000000000",    -- el resto de las
28          "00000000000000000000000000000000000",    -- instrucciones estan
29          "00000000000000000000000000000000000",    -- en blanco
```

```
4112        "00000000000000000000000000000000000",    -- instruccion 4095
4113        "00000000000000000000000000000000000"     -- instruccion 4096
4114        );
4115    begin
4116
4117       dataout <= memory(to_integer(unsigned(address)));
4118
4119    end Behavioral;
```

Recordar que la estructura de cada una de las instrucciones de 36 bits queda a criterio de cada grupo.

- `RAM.vhd`

  Este módulo recibe un vector de doce bits, que representará una dirección de memoria, y entrega como salida un vector de 16 bits, que representará el valor guardado en dicha dirección. Dentro de este componente habrá una arreglo de vectores, que representaran una memoria que se puede escribir en flanco de subida.

  Además recibirá como entrada un vector de 16 bits `datain`, junto con un valor lógico `write`, tal que cuando este último tenga un valor igual a uno y se encuentre en un flanco de subida, se guardará el vector `datain` el la dirección de memoria indicada en la entrada del vector de doces bits.

  Para que este módulo funcione de la forma esperada para esta entrega, debe conectar la señal de salida del componente `Clock Divider` a la señal `clock` de este módulo.

- `Reg.vhd`

  Este módulo recibe un vector de 16 bits y 4 valores lógicos `clock`, `load`, `up` y `down`. Este componente funciona como un registro 16 bits de flanco de subida.

  Cuando la entrada `load` este activada y se encuentre la entrada `clock` en un flanco de subida, se guardara en el registro el vector de entrada. Por otro lado, si la entrada `up` este activada y se encuentre la entrada `clock` en un flanco de subida, se guardara en el registro el sucesor del valor previamente almacenado en el registro. Finalmente, si la entrada `down` este activada y se encuentre la entrada `clock` en un flanco de subida, se guardara en el registro el antecesor del valor previamente almacenado en el registro. Inicialmente el valor almacenado en este registro es igual a cero.

  Para que este módulo funcione de la forma esperada para esta entrega, debe conectar la señal de salida del componente `Clock Divider` a la señal `clock` de este módulo.

- `Clock_Divider.vhd`

  Este componente recibe como entrada un valor lógico `clk`, y un vector de dos bits y da como salida un valor lógico `clock`.

  Su función es reducir la frecuencia de un clock a distintas frecuencias dependiendo del vector de dos bits. Esto ayuda a la hora de visualizar los cambios que ocurren en el computador, debido a que si se ocupará la frecuencia del reloj `clk` de la Basys3 no se podría visualizar los cambios que ocurren en cada instrucción porque ocurrirían en fracciones de segundo.

  Tomando como entrada la señal `clk` el cual tiene una frecuencia de 100 Mhz, y dependiendo del vector de entrada la señal de salida tendrá distintas frecuencias como se indican a continuación:

  1. Si el vector es igual a `"00"`, entonces la frecuencia del reloj de salida será 25 Mhz.

2. Si el vector es igual a `"01"`, entonces la frecuencia del reloj de salida será 8 Mhz.
3. Si el vector es igual a `"10"`, entonces la frecuencia del reloj de salida será 2 Mhz.
4. Si el vector es igual a `"11"`, entonces la frecuencia del reloj de salida será 0.5 Mhz.

Como se menciono antes, para que este módulo funcione, debe conectar la señal de entrada de la Basys3 `clock`, a la señal de entrada `clk` de este módulo.

Por otra parte se le entregará módulos VHDL podrán usar de manera opcional:

- `Debouncer.vhd`

  Este módulo permite el correcto funcionamiento de los botones, dado que estos generan una señal vibratoria al ser pulsados.

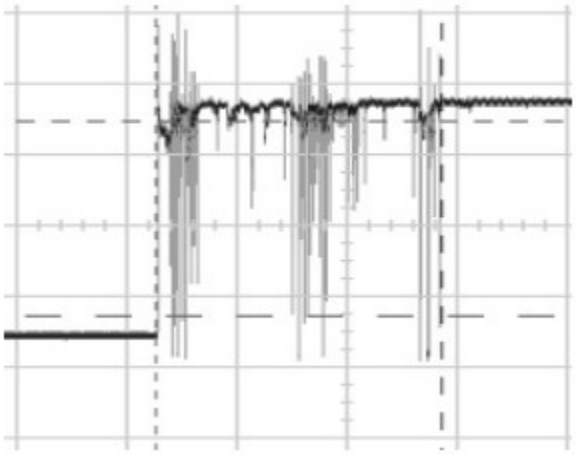

Figura 2: Señal al pulsar un botón

Recibe la señal de un botón y entrega la misma señal corregida.

Para que este módulo funcione, debe conectar la señal de entrada de la Basys3 `clk`, a la señal de entrada `clk` de este módulo.

Tablas de verdad componentes

A continuación se mostrará la tabla de verdad de algunos de los componentes del computador de esta entrega.

- `ALU`

Entradas			Salidas			
a(15:0)	b(15:0)	sel	result(15:0)	c	z	n
*	*	add	a + b	result < a + b	result = 0	0
*	*	sub	a - b	a >= b	result = 0	b > a
*	*	and	a and b	0	result = 0	0
*	*	or	a or b	0	result = 0	0
*	*	xor	a xor b	0	result = 0	0
*	*	not	not a	0	result = 0	0
*	*	shr	0 & a(15 downto 1)	a(0)	result = 0	0
*	*	shl	a(14 downto 0) & 0	a(15)	result = 0	0

Figura 3: Tabla de verdad de la ALU.

- `Reg`

Entradas					Salidas
clock	load	up	down	datain	dataout
Flanco Subida	1	*	*	*	datain
Flanco Subida	0	1	*	*	dataout + 1
Flanco Subida	0	0	1	*	dataout - 1
*	*	*	*	*	dataout

Figura 4: Tabla de verdad de Reg.

- `RAM`

Entradas				Salidas
clock	write	address	datain	dataout
Flanco Subida	1	*	*	Mem[address] = datain
*	*	*	*	Mem[address]

Figura 5: Tabla de verdad de la RAM.

- `ROM`

Entradas	Salidas
address	dataout
*	Mem[address]

Figura 6: Tabla de verdad de la ROM.

Bitácora de proyecto

Adicionalmente, para esta entrega su grupo deberá escribir una bitácora **en formato Markdown** que hable de los siguientes puntos:

- Explicación del funcionamiento de cada componente de su implementación del computador básico y con especial detalle en lo que se refiere la unidad de control (arquitectura interna y señales).
- La especificación de la estructura de las instrucciones de su CPU, es decir, la codificacíon de cada uno de los bits de su palabra de instrucción.
- El trabajo realizado por cada uno de los miembros de su grupo. Se debe indicar específicamente que hizo cada integrante del grupo y deben explicar que fue lo mas difícil para el grupo en la entrega.
- El código en *assembly* del algoritmo de multiplicación que introdujeron en la ROM de su computador básico.
- Una tabla o referencia donde se indique que instrucciones son soportadas por su implementación de acuerdo a la especificación dada anteriormente.

Requerimientos

- Crear el proyecto
  - Seleccionar las opciones correctas para crear el proyecto en Vivado, que funcione con la placa correspondiente.
  - Importar correctamente el archivo `Basys3.xdc` .
  - Configura correctamente las *constrains* del archivo `Basys3.xdc`. Descomentando las líneas correctas del archivo.
- Crea la arquitectura necesaria:
  - Importar correctamente los módulos entregados en el enunciado.
  - Crear sources para cada componente cada una conteniendo la arquitectura correspondiente
  - Se pueden crear *sources* adicionales para facilitar el problema
- **Incluir el README.md, el Informe y los archivos correspondientes con lo solicitado.**

Entrega

La entrega se realizará a través de GitHub. El repositorio debe contener una carpeta con su proyecto de Vivado y el archivo `.bit`. En el caso de la carpeta del proyecto, deben subir solo la carpeta `.srcs`, el archivo `.xpr` y el archivo `Basys3.xdc` (las carpetas y archivos restantes se recomienda evitar su subida configurando el archivo `.gitignore`).

Evaluación

La evaluación de su entrega se enfocará en los siguientes puntos:

- El uso correcto de Vivado y escritura en VHDL, es decir, que los archivos se desplieguen correctamente al abrir el archivo .xpr y que estos sean capaces de compilar el .bit
- El diseño de su palabra de control. Que este diseño cumpla con lo pedido en las especificaciones de la CPU.
- El correcto funcionamiento del archivo .bit cargado en la placa y la correcta escritura de la ROM.
- El contenido de la bitácora. Este debe explicar con claridad los elementos que se pidieron.
- **Evaluación de Pares.** Deberán contestar un form semanalmente respecto al trabajo que han realizado juntos. Deberán indicar si han trabajado como grupo y quienes han trabajado. No contestar este form significará un descuento en su nota.

Integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1,1 en el curso y se solicitará a la Dirección de Docencia de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente.

Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.