Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (I/2020)

Proyecto Semestral: Entrega Práctica 02

Computador primigenio

Fecha de entrega: Viernes 8 de Mayo a las 20:00 horas

1. Objetivo

Para esta entrega tendrán que diseñar y armar su propio computador básico **de 16 bits**, el cual será utilizado 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 adelante, dentro de los cuales se encuentra soportar una lista determinada de instrucciones en assembler.

Tendrán que crear su propia conversión de assembler a binario, la cual estará dada por la estructura que escojan para las palabras de instrucción. Para esto tendrán que **decidir** entre programar su propio *assembler* o hacer la conversión de sus palabras de instrucción manualmente. Es importante destacar que **el programa *assembler* no será evaluado** en la entrega, pero su realización facilitará enormemente las pruebas que deberán realizar en su CPU (la automatización que les dará su *assembler* será relevante para ahorrar tiempo). La documentación para realizar su *assembler* se entregará al mismo tiempo que este enunciado.

2. Especificaciones CPU

El computador que deberán armar debe contar con:

- Dos registros de 16 bits (registros A y B).
- Una memoria ROM de instrucciones de 4096 palabras de 33 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 (todas las de la Entrega 1).
- Un *program counter* (PC) de 12 bits de direccionamiento.
- Dos multiplexores de cuatro entradas de 16 bits.
- Una unidad de control de lógica combinacional.

- Un registro de status (C, Z, N).

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 porte de cada bus). Cabe mencionar que todos los bloques síncronos de la Figura 1 son de flanco de subida.

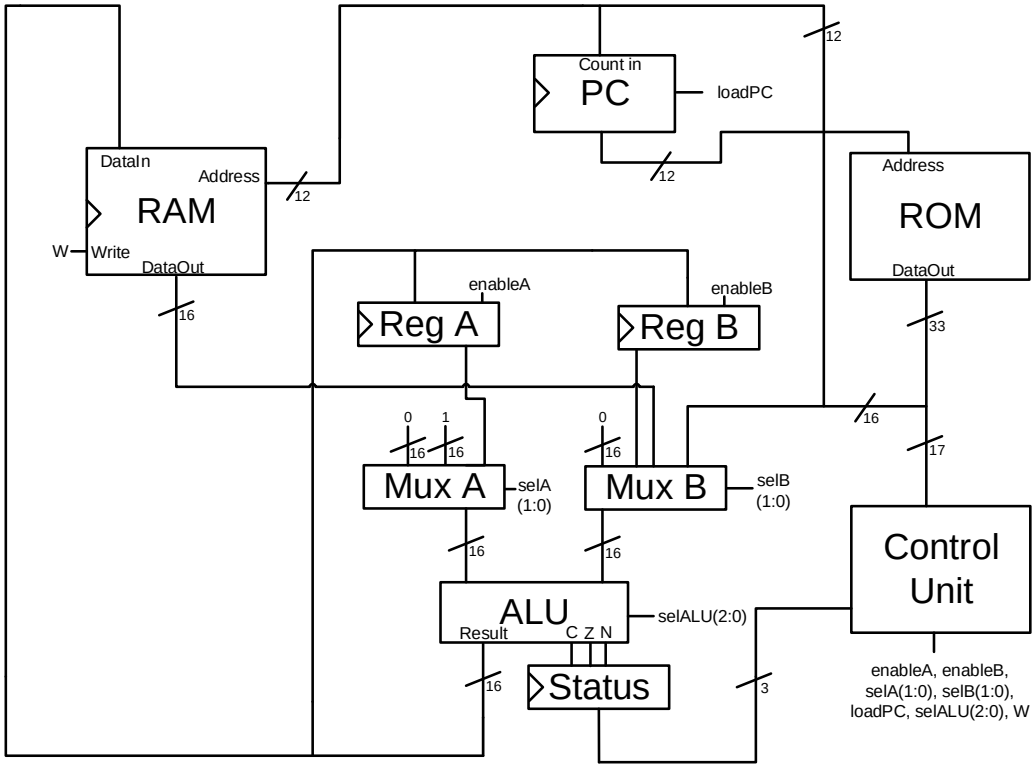


Figura 1: Computador básico.

2.1. Módulos entregados

Se les entregarán varios módulos VHDL que deberán utilizar para desarrollar su entraga:

- `ROM.vhd`

  Este módulo recibe un vector de doce bits, que representará una dirección de memoria, y entrega como salida un vector de 33 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.

  Recordar que la estructura de cada una de las instrucciones de 33 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 `clk`.

- `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 `clk`.

2.2. 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 2: 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 3: Tabla de verdad de Reg.

- `RAM`

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

Figura 4: Tabla de verdad de la RAM.

- `ROM`

Entradas	Salidas
address	dataout
*	Mem[address]

Figura 5: Tabla de verdad de la ROM.

3. Desarrollo

Para desarrollar esta entrega se recomienda a los grupos realizar los siguientes pasos.

1. **Componentes:** Crear e importar los componentes que se especifican en el enunciado en el archivo Basys3, procurando conectar correctamente las señales de entradas y salidas.

2. **Opcode y Señales:** Analizar y crear la forma como la *ROM* entrega las instrucciones a la *Control Unit* para posteriormente generar las señales. Se recomienda usar tablas que identifiquen el código de operación y sus respectivas señales asociadas.

3. **Comportamiento de los Componentes:** Completar los comportamientos específicos de cada uno de los componentes a desarrollar. Estos son: *ALU*, *Control Unit* y *MUX*.

   a) *ALU*: componente que resuelve los problemas aritméticos lógicos del computador. En esta entrega tiene las mismas características que en la entrega 1.

   b) *Control Unit*: componente que procesa las palabras provenientes de la *ROM* para generar señales que afectan el comportamiento de todos los componentes de la CPU. La *Control Unit* procesa la instrucción de la *ROM* y los valores del registro *Status* para elaborar las señales. La forma en que transforma el código de operación de la *ROM* a señales la deberán implementar ustedes.

   c) *MUX*: componente que actúa de selector al igual que en en la entrega pasada.

4. **De Assembly a Binario:** Transformar los test a evaluar de lenguaje assembly a binario, generando un archivo ROM para cada uno de los test. Esto se podrá hacer manualmente o automáticamente, es decisión suya elegir como hacerlo.

5. **Simulación:** Finalmente, teniendo todo lo anterior listo, deberán simular y observar si los valores transitivos y finales de los display corresponden a los valores esperados o no. Para esto se le entregará un archivo de simulación listo con explicaciones de como deberán probar sus test.

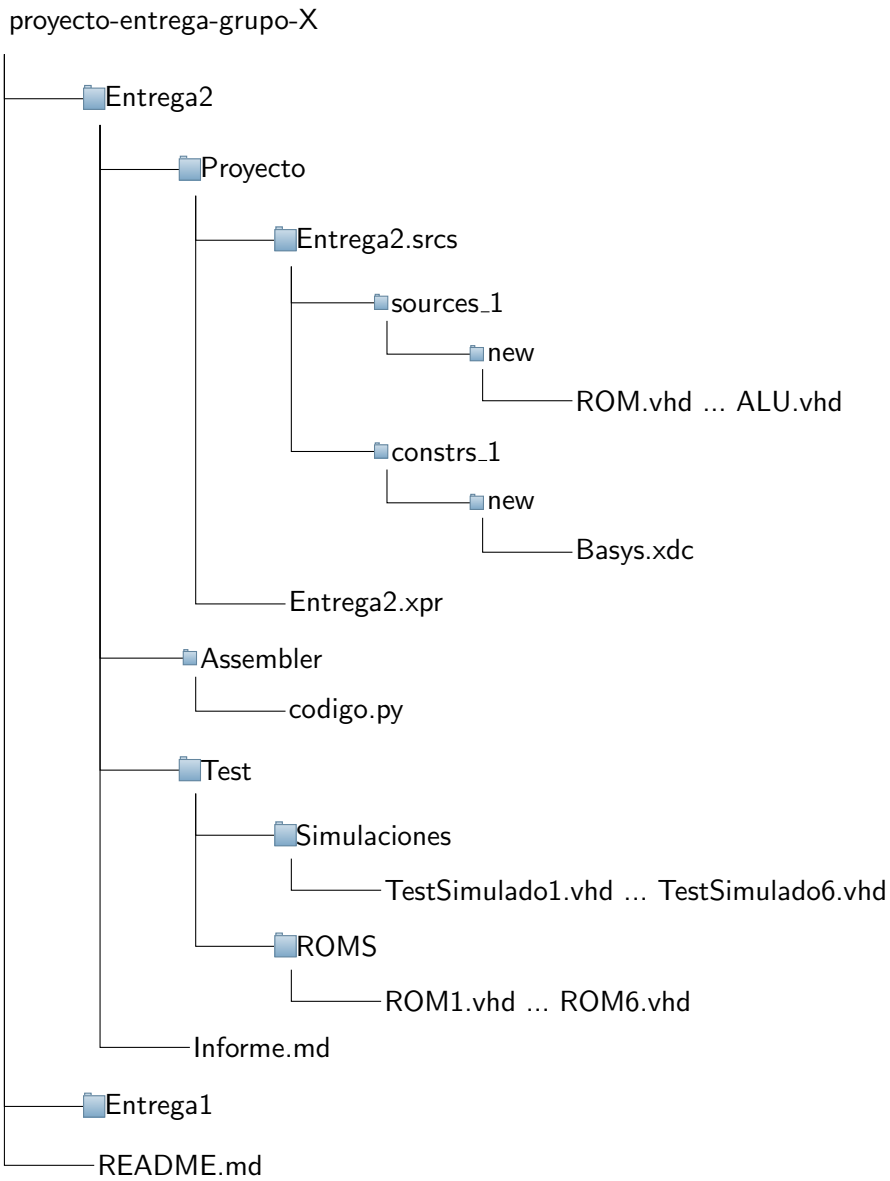
4. Requerimientos

A continuación se describen los requerimientos de esta entrega y el puntaje asociado a cada item:

- Para implementar declaraciones condicionales **solamente** se permite hacer uso de bloques `with/select`. El uso de los *statements process*, `case` e `if/else` quedan absolutamente prohibidos. Esto porque se privilegia el uso de selectores y operaciones lógicas básicas para el desarrollo de esta tarea. **Para esta entrega, queda estrictamente prohibido utilizar cualquier tipo de librería aritmética que simplifique la tarea de la suma**. ***Nota:*** Esto solo aplica a los componentes a desarrollar, en los archivos de simulación se permite el uso de process.
- 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` .
  - Configurar correctamente las *constrains* del archivo `Basys3.xdc`. Descomentando las líneas correctas del archivo.
- (6 pts) Se evaluara el contenido dentro de la entidad `Basys3.vhd`
  - (3 pts) Test correctos, bien visualizados y bien simulados en la Basys3:
    - (1 pts) Test 1
    - (0.5 pts) Test 2
    - (0.5 pts) Test 3
    - (0.5 pts) Test 4
    - (0.5 pts) Test 5
  - (0.5 pts) Correcta arquitectura de la CPU.
  - (0.5 pts) Uso correcto de todos los componentes de la CPU.
  - (0.5 pts) Correcta implementación de la *Control Unit* con las tablas de opcode y señales consistentes.
  - (0.5 pts) Correcta conexión de los comportamiento de la entidad Basys3 para desarrollar el problema.
  - (1 pts) Contenido del informe.
  - Puede crear más *sources* para facilitar el problema.
  - Habiendo completado la creación del modulo `Basys3.vhd` ustedes ya podrán optar por la nota 7 en la entrega.
- Bonus
  - (0.5 pts) Test 6
  - Habiendo realizado el bonus ustedes obtendrán 0.5 ptos extras en esta entrega, pudiendo obtener como nota máxima de 7.5
- **Incluir el un archivo Infome.md :** En este debe describir 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.

Además, se deberá incluir resultados de los archivos de simulación, junto con los archivos ROM.vhd asociados, mostrando que su entidad efectivamente resuelve el problema. Para esto basta con mostrar el valor del display al final de la simulación que deberá mostrar los 8 bits menos significativos de cada registro (disA y disB serán el registro A, disC y disD serán el registroB. Los 4 bits menos significativos de cada registro son mostrados en disB y disD). **Nota:** se recomienda hacer más casos para probar su circuito, pero a la hora de escribir el informe solo basta con mostrar cada uno de los Test.

- **Formato del repositorio:** El formato del respositorio debe ser el siguiente:



NOTA: La carpeta Assembler es opcional, solo en caso de que hagan un programa para ello. Si traspasan los test a código de máquina manualmente no se debe subir, y lo deben especificar en su informe. **EL NO SEGUIR ESTE FORMATO DE REPOSITORIO PUEDE SIGNIFICAR UN DESCUENTO EN SU CALIFICACIÓN FINAL.**

5. Entrega

La entrega se realizará a través de GitHub y deberán entregar en el repositorio:

- La carpeta con su proyecto de Vivado. En el caso de la carpeta del proyecto, deben subir solo la carpeta `.srcs`, el archivo `.xpr` y el archivo `Basys3.xdc` dentro de la carpeta new de los constrains (ver el diagrama de más arriba).
- Una carpeta Test que contenga dos carpetas. Una carpeta de *Simulaciones* con sus archivos de simulaciones para cada test y una carpeta *ROMS* con sus archivos ROM para cada test
- Su archivo *informe.md*

6. Evaluación de pares

La evaluación de pares estará activa durante hasta el miércoles 06 de Mayo a las 23 horas. El link es el siguiente:

`https://forms.gle/nXXjdczLSKSwVHpF9`.

Esto con el objetivo que nos informen si algún integrante de su grupo no se comunica o no se compromete con el proyecto, para que como equipo docente poder intervenir antes de la entrega y poder encontrar soluciones posibles para el correcto desarrollo de la misma.

7. Ejemplos

Considere los siguientes ejemplos que corresponden a diversos programas que su CPU deberá soportar:

- Programa 1:

```
DATA:
CODE:           // Canten 'La Farolera':
MOV A,2         // 2
MOV B,2         // y 2
ADD A,B         // son 4
NOP             // 4
NOP             // y 2
NOP             // son
ADD A,2         // 6
NOP             //
NOP             // 6
NOP             // y 2
ADD A,B         // son 8
MOV B,A         // y 8
ADD A,B         // 16
NOP             //
NOP             //
NOP             // A = 10h , B =8h
```

- Programa 2:

```
DATA:

CODE:           // Swaps

MOV A,3         // A = 3
MOV B,5         // B = 5

MOV (0),A       // |
MOV A,B         // |
MOV B,(0)       // | Swap con MOV y variable auxiliar

SUB A,B         // A = 2

XOR A,B         // |
XOR B,A         // |
XOR A,B         // | Swap con XOR
```

- Programa 3:

```
DATA:           // Variables a sumar

 a 5
 b Ah

CODE:           // Sumar variables

MOV A,0         // 0 a A
ADD A,(a)       // A + a a A
ADD A,(b)       // A + b a A
MOV B,A         // Resultado a B

end:
 DEC A          // A--
JMP end
```

- Programa 4:

```
DATA:

 a E5h          // 11100101b
 b B3h          // 10110011b
 bits 0b

CODE:           // Contar bits en 1 compartidos
```

```
8
9  MOV A, ( a)      // a a A
10 AND A , ( 1d )   // A & b a A
11 JMP loop         // Empieza desde loop
12
13 bit:
14  INC  (2h)       // bits ++
15  loop:
16   CMP A ,0b      // Si A == 0
17   JEQ  end       // Terminar
18   SHR A          // Si A >> 1 genera carry
19   JCR  bit       // Siguiente desde bit
20                  // Si no
21   JMP  loop      // Siguiente desde loop
22
23 end:
24  MOV A,(10b)     // Resultado a A
25  JMP end
```

- Programa 5:

```
1  DATA:
2
3   varA 8
4   varB 3
5
6  CODE:            // Restar sin SUB ni ADD:
7
8  MOV A,(varB)     // varB a A
9  NOT (varB),A     // A Negado a varB
10 INC (varB)       // Incrementar varB
11
12 suma:
13
14 MOV A,(varA)     // Resultado a A
15
16 end:
17  NOP
18  JMP end
```

- Programa 6:

```
1  DATA:
2
3  CODE:            // No debe saltar
4
```

```
5  JMP start
6
7  error:
8   MOV A,FFh      // FFh a A
9   JMP error
10
11 start:
12  MOV B,1
13  MOV A,B
14  INC A
15  CMP A,B
16  JEQ error
17
18  INC B
19  CMP A,2
20  JNE error
21
22  MOV (0),A
23  INC B
24  CMP A,2
25  JGT error
26  CMP A,(0)
27  JGT error
28
29  INC B
30  INC (0)
31  CMP A,(0)
32  JGE error
33
34  INC B
35  CMP A,2
36  JLT error
37
38  CMP A,1
39  JLT error
40  INC B
41  DEC A
42  CMP A,0
43  JLE error
44
45  INC B
46  SHL A
47  JCR error
48
49  SUB A,3
50  JCR error
51
52  MOV A,11h      // 11h a A
```

8. Assembly

Esta es la lista de instrucciones soportadas para esta entrega.

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
JGT	Ins	carga Ins en PC si en el status N = 0 y Z = 0
JGE	Ins	carga Ins en PC si en el status N = 0
JLT	Ins	carga Ins en PC si en el status N = 1
JLE	Ins	carga Ins en PC Ins si en el status N = 1 o Z = 1
JCR	Ins	carga Ins en PC Ins si en el status C = 1
NOP		no hace cambios

9. Contacto

Cualquier pregunta sobre el proyecto, ya sean de enunciado, contenido o sobre aspectos administrativos deben comunicarse con los ayudantes creando issues en el Syllabus del Github del curso o directamente con los ayudantes:

- Felipe Valenzuela: frvalenzuela@uc.cl
- Matías López: milopez8@uc.cl
- Cristóbal Herreros: ceherreros@uc.cl
- Raúl Del Río Jara: rjdelrio@uc.cl

10. 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 esta 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.

Esta 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.