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

IIC2343 – Arquitectura de Computadores (II/2015)

Proyecto Semestral: Entrega II

Computador básico y compilador assembly

Fecha de entrega: Domingo 4 de Octubre

1. Objetivo

Para esta entrega tendrán que diseñar y armar su propio computador básico, el cual utilizarán posteriormente para ejecutar programas hechos en lenguaje assembler. En una primera instancia tendrán que armar en Vivado 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.

A continuación tendrán que crear su propia conversión de assembler a binario, la cual estará dada por la estructura que escojan para las palabras de instrucción. Además, deberán programar su propio *assembler* que soporte las instrucciones y especificaciones detalladas más adelante.

Por último tendrán que realizar unos programas simples para corroborar el correcto funcionamiento de su computador.

2. Especificaciones CPU

Sin entrar todavía en muchos detalles, 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 sumar y restar.
- 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 (Z, N, C).

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.

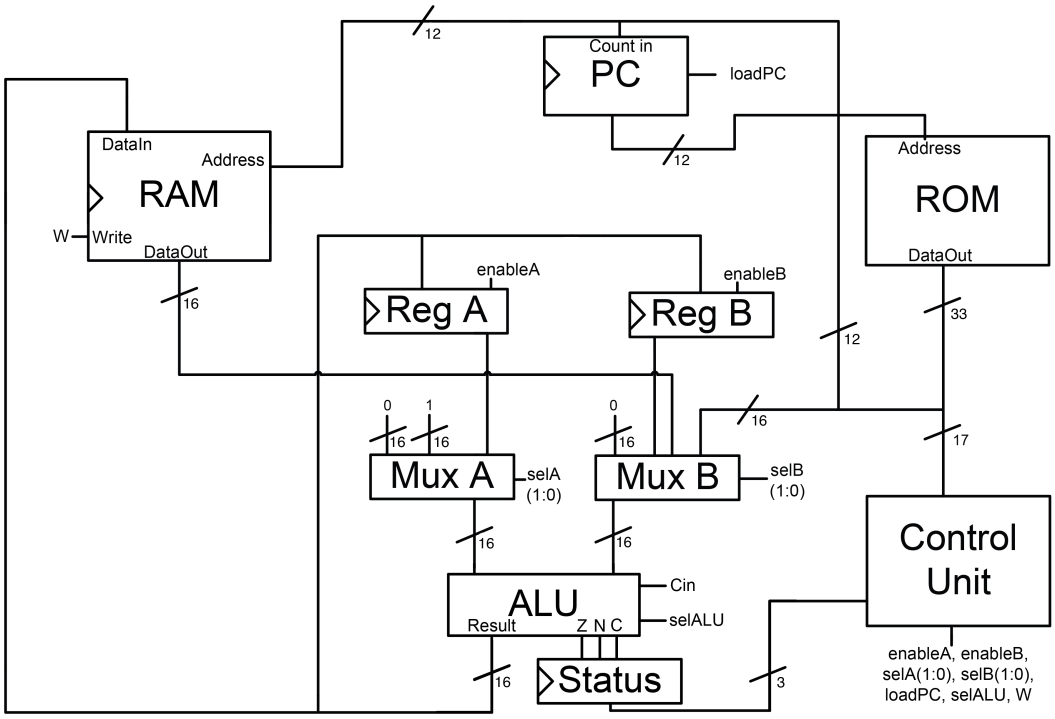

Figura 1: Computador básico.

A continuación se presenta el listado de instrucciones que debe soportar la CPU:

```
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 A,B       (guarda A+B en A)
    B,A       (guarda A+B en B)
    A,Lit     (guarda A+literal en A)
    B,Lit     (guarda A+literal en B)
    A,(Dir)   (guarda A+Mem[Dir] en A)
    B,(Dir)   (guarda A+Mem[Dir] en B)
    (Dir)     (guarda A+B en Mem[Dir])
SUB A,B       (guarda A-B en A)
    B,A       (guarda A-B en B)
    A,Lit     (guarda A-literal en A)
    B,Lit     (guarda A-literal en B)
    A,(Dir)   (guarda A-Mem[Dir] en A)
```

```
    B,(Dir)   (guarda A-Mem[Dir] en B)
    (Dir)     (guarda A-B en Mem[Dir])
AND A,B       (guarda A and B en A)
    B,A       (guarda A and B en B)
    A,Lit     (guarda A and literal en A)
    B,Lit     (guarda A and literal en B)
    A,(Dir)   (guarda A and Mem[Dir] en A)
    B,(Dir)   (guarda A and Mem[Dir] en B)
    (Dir)     (guarda A and B en Mem[Dir])
OR  A,B       (guarda A or B en A)
    B,A       (guarda A or B en B)
    A,Lit     (guarda A or literal en A)
    B,Lit     (guarda A or literal en B)
    A,(Dir)   (guarda A or Mem[Dir] en A)
    B,(Dir)   (guarda A or Mem[Dir] en B)
    (Dir)     (guarda A or B en Mem[Dir])
XOR A,B       (guarda A xor B en A)
    B,A       (guarda A xor B en B)
    A,Lit     (guarda A xor literal en A)
    B,Lit     (guarda A xor literal en B)
    A,(Dir)   (guarda A xor Mem[Dir] en A)
    B,(Dir)   (guarda A xor Mem[Dir] en B)
    (Dir)     (guarda A xor B en Mem[Dir])
NOT A         (guarda not A en A)
    B,A       (guarda not A en B)
    (Dir),A   (guarda not A en Mem[Dir])
SHL A         (guarda shift left A en A)
    B,A       (guarda shift left A en B)
    (Dir),A   (guarda shift left A en Mem[Dir])
SHR A         (guarda shift right A en A)
    B,A       (guarda shift right A en B)
    (Dir),A   (guarda shift right 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       (ir a la instrucción Ins)
JEQ Ins       (ir a la instrucción Ins si es que A=B)
JNE Ins       (ir a la instrucción Ins si es que A!=B)
JGT Ins       (ir a la instrucción Ins si es que A>B)
JGE Ins       (ir a la instrucción Ins si es que A>=B)
JLT Ins       (ir a la instrucción Ins si es que A<B)
JLE Ins       (ir a la instrucción Ins si es que A<=B)
JCR Ins       (ir a la instrucción Ins si es que se produce un carry out en la ALU)
NOP           (no hace cambios)
```

La estructura de cada una de las instrucciones de 33 bits queda a criterio de cada grupo, teniendo que definir según eso el número binario correspondiente a cada operación.

Para llevar a cabo esta entrega considere lo siguiente:

- Se les entregará ya hecho los archivos de la RAM, de la ROM y del *program counter*.
- Utilizando como base el proyecto de la entrega anterior, debe incorporar los módulos antes mencionados y mantener conectados directamente al Led_Driver las salidas de los registros A y B. La idea es que constantemente se muestre en los displays de siete segmentos los 8 bits menos significativos de los números almacenados en ambos registros en formato hexadecimal.
- Los 3 leds de más a la derecha deben estar encargados de prenderse de acuerdo a la salida del registro status, uno por cada señal. Para incluir los leds en su CPU debe descomentar las lineas en Basys3.xdc y agregar las salidas en Basys3.vhd.

Otro punto importante que debe tener en cuenta es que la CPU solo opera con números positivos de 16 bits (o sea, no en complemento de 2), lo que por ahora implica que a nivel de *assembly* la CPU no tiene que soportar números negativos.

Para utilizar y modificar la ROM entregada (que corresponde a un archivo en VHDL) veamos el siguiente ejemplo donde se muestra el código de una ROM de 4096 palabras de 33 bits que tiene sólo 9 instrucciones:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity ROM is
    Port (
        address   : in  std_logic_vector(11 downto 0);
        dataout   : out std_logic_vector(32 downto 0)
          );
end ROM;

architecture Behavioral of ROM is

type memory_array is array (0 to ((2 ** 12) - 1) ) of std_logic_vector (32 downto 0);

signal memory : memory_array:= (
    "000000000000000100000111000000010",   -- instrucción 1
    "000000000000000000000011000000011",   -- instrucción 2
    "000000000000000100000111000000010",   -- instrucción 3
    "000000000000000100000011000000011",   -- instrucción 4
    "000000000000001100000111000000010",   -- instrucción 5
    "000000000000001000000011000000011",   -- instrucción 6
    "000000000000001000000111000000010",   -- instrucción 7
    "000000000000001100000011000000011",   -- instrucción 8
    "000000000000000000000111000000010",   -- instrucción 9
    "000000000000000000000000000000000",    -- el resto de las
```

```
    "00000000000000000000000000000000",    -- instrucciones están
    "00000000000000000000000000000000",    -- en blanco
    "00000000000000000000000000000000",
    .
    .
    .
    "00000000000000000000000000000000",
    "00000000000000000000000000000000",
    "00000000000000000000000000000000",    -- instrucción 4095
    "00000000000000000000000000000000"     -- instrucción 4096
);
begin

    dataout <= memory(to_integer(unsigned(address)));

end Behavioral;
```

Como se puede apreciar, para introducir una instrucción deseada en la ROM basta con escribir la palabra de 33 bits en la dirección de memoria que queramos. Las direcciones de memoria que no se utilicen deben ser llenadas con ceros. La indización de la instrucción es decreciente de izquierda a derecha, es decir, el bit más significativo es el 32 y el menos significativo, el 0.

Si bien la RAM y el *program counter* entregados también corresponden a archivos en VHDL, ustedes no tendrán que modificarlos, limitándose a utilizarlos como bloques. Considere que inicialmente la RAM contiene valores desconocidos en cada una de su direcciones, motivo por el cual sólo pueden hacer operaciones utilizando posiciones de memoria en las que hayan guardado algún dato conocido previamente.

3. Especificaciones *assembler*

A continuación se describen y detallan las características y especificaciones que debe soportar el *assembler* creado por cada grupo:

1. Debe ser capaz de soportar *labels* de la forma:

   ```
   label1:

   label2:
   ```

   donde cada *label* representa una dirección en la ROM de instrucciones, luego su *assembler* tiene que asignar automáticamente cada *label* a la dirección de ROM de instrucciones correcta. Pueden asumir que los *labels* comienzan con una letra (mayúscula o minúscula) y que solo pueden tener letras o números entre sus caracteres.

2. Se debe poder definir variables con valores iniciales al principio del programa. Para inicializar las variables definidas al principio del programa su *assembler* debe generar las instrucciones necesarias para cargar el literal en la memoria RAM, cosa que puede requerir más de un ciclo de instrucción.

3. Los literales deben poder ingresarse como decimal (de la forma 102d o 102), binario (de la forma 01010b) y hexadecimal (de la forma AAh). Si no se especifica el formato del literal, el programa debe asumir que se trata de un decimal.

4. Debe soportar espacios en blanco y comentarios (los cuales se anteceden con //).

5. Su compilador **NO** puede necesitar elementos adicionales como una línea *END* al final del código para poder compilar.

6. Su programa debe tomar un archivo .txt (cuya ubicación debe ser especificada por el usuario) que contiene el código en assembly, para luego retornar un archivo output.txt (en la misma carpeta del archivo de origen) de la forma:

   ```
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_UNSIGNED.ALL;
   USE IEEE.NUMERIC_STD.ALL;

   entity ROM is
       Port (
           address   : in  std_logic_vector(11 downto 0);
           dataout   : out std_logic_vector(32 downto 0)
             );
   end ROM;

   architecture Behavioral of ROM is

   type memory_array is array (0 to ((2 ** 12) - 1) ) of std_logic_vector (32 downto 0);
   ```

```
signal memory : memory_array:= (
    "00000000000000010000111000000010",   -- instrucción 1
    "00000000000000000000011000000011",   -- instrucción 2
    "00000000000000100000111000000010",   -- instrucción 3
    "00000000000000010000011000000011",   -- instrucción 4
    "00000000000001100000111000000010",   -- instrucción 5
    "00000000000000100000011000000011",   -- instrucción 6
    "00000000000001000000111000000010",   -- instrucción 7
    "00000000000001100000011000000011",   -- instrucción 8
    "00000000000000000000111000000010",   -- instrucción 9
    "00000000000000000000000000000000",    -- el resto de las
    "00000000000000000000000000000000",    -- instrucciones están
    "00000000000000000000000000000000",    -- en blanco
    "00000000000000000000000000000000",
    .
    .
    .
    "00000000000000000000000000000000",
    "00000000000000000000000000000000",
    "00000000000000000000000000000000",    -- instrucción 4095
    "00000000000000000000000000000000"     -- instrucción 4096
);
begin

    dataout <= memory(to_integer(unsigned(address)));

end Behavioral;
```

esto de manera de poder copiar y pegar el archivo resultante en ROM.vhd. Pueden asumir que al archivo de entrada viene correcto, es decir, no deben manejar errores.

7. No es necesario que el compilador soporte líneas que contengan *labels* e instrucciones a continuación (del tipo `label123: MOV A,B`).

4. Especificaciones de la funciones a programar

Para probar el computador hecho por cada grupo tendrán que hacer un programa que multiplique los valores guardados en los registros A y B, teniendo en consideración de que $A * B$ no puede superar el máximo número representable con dos dígitos hexadecimales, de manera que el resultado se pueda apreciar correctamente en el display de 7 segmentos.

Además deberán hacer otro programa que tome los valores almacenados en ambos registros y realice la operación A^B, es decir, A elevado a B, teniendo siempre en cuenta las mismas condiciones anteriores.

5. Ejemplos

Considere los siguientes ejemplos, los cuales corresponden a diversos programas que su *assembler* debe ser capaz de compilar:

```
Programa 1:

DATA: //esto es un comentario, solo se permiten comentarios de una linea
var1 3 //se declara la variable var1 cuyo valor inicial es 3
var2 4 //se declara la variable var2 cuyo valor inicial es 4

CODE: //para separar la zona de definicion de variables del codigo
MOV A,(var1) //guarda un 3 en A
MOV B,(var2) //guarda un 4 en B
INC A //A=3+1=4
SUB A,B //A=A-B=0
ADD A,B //A=A+B=4
JMP hola
INC B
hola:
JMP hola //muestra 4 en A y 4 en B (si esta bueno) y 5 en B si esta malo

Programa 2:

DATA:

CODE:
MOV A,2 //A=2
MOV B,5 //B=5
MOV (2),A //Mem[2]=2
MOV (3),B //Mem[3]=5
MOV (4),A //Mem[4]=2
// espacio en blanco
MOV A,(2) //A=2
MOV B,(4) //B=2
ADD B,A //B=4
MOV A,(3) //A=5
fin:
JMP fin
```

Programa 3:

```
DATA:
var1 9

CODE:

MOV A,(var1)
MOV B,(var1)
ADD A,1 // A=10
SUB B,A // B=A-B=10-9=1
SUB A,(var1) //A=A-Mem[var1]=10-9=1
DEC A //A=0
fin:
JMP fin
```

6. Entrega y consultas

Específicamente cuando se cumpla el plazo deben entregar:

- El proyecto completo de la CPU, es decir, todos los archivos involucrados en su proyecto en un **archivo comprimido**.
- La placa de desarrollo con la CPU cargada y funcionando el día de la entrega en horario de ayudantía.
- Un solo archivo ejecutable (.exe o .jar) que contenga su *assembler*. **NO** se recibirá código.
- Un archivo de texto por cada una de las funciones a programar (el multiplicador y la potencia) escrito en *assembly*.
- Un breve informe (incluyendo el número de grupo y el nombre de los integrantes) que contenga la especificación de la estructura de las instrucciones de su CPU (función de cada uno de los 33 bits de una instrucción), más una tabla con todas las instrucciones soportadas por la CPU y su implementación de acuerdo a la especificación indicada anteriormente. Además, este informe debe contener la explicación de cómo ocupar su *assembler*, detallando paso a paso cómo usar su programa para ensamblar un archivo en *assembly* y generar el archivo de salida para insertar en la ROM. Si bien todos los *assembler* entregados debiesen ocuparse de la misma manera, de todas formas puede ser que existan pequeñas variaciones entre unos y otros, motivo por el cual deben explicar la forma de uso. Adicionalmente, este informe debe indicar específicamente qué hizo cada integrante del grupo durante la entrega (Un párrafo por persona).

Para entregar sus proyectos deben dejar todo lo solicitado en un archivo comprimido (.zip o .rar) en la carpeta compartida en Dropbox correspondiente a su grupo. La fecha de entrega vence el Domingo 4 de Octubre a las 23:59 horas, entregas atrasadas serán **penalizadas con 0.5 puntos** por cada hora (o fracción) de atraso.

Además se realizará una corrección presencial de esta entrega, en la que se probarán sus *assembler* con distintos programas en la hora de ayudantía del Lunes 5 de Octubre. Adicionalmente, y en base al informe entregado, se podrá interrogar a algunos alumnos. En caso que el alumno no demuestre los conocimientos que dice o debería tener de acuerdo a su trabajo, se le podrá aplicar un **descuento de hasta 1.0** punto en su nota individual de la entrega.

Cualquier pregunta sobre el proyecto, ya sean de enunciado, contenido o sobre aspectos administrativos deben comunicarse con Francesca Lucchini, al mail flucchini@uc.cl.

7. Evaluación

Cada una de las entregas del proyecto se evaluará de forma grupal y se ponderará por un porcentaje de coevaluación para calcular la nota de cada alumno.

Dado lo anterior, dentro de las primeras **24 horas** posteriores a cada entrega, **todos los alumnos** deberán enviar de forma **individual y obligatoria** un mail a los ayudantes repartiendo hasta 4 puntos, con hasta un decimal, entre sus compañeros. La suma de todos los puntos obtenidos por el integrante, *sp*, será utilizada para el cálculo de la nota de cada entrega, lo que puede hacer que este repruebe el curso.

La nota de cada entrega se calcula de la siguiente forma:

$$NotaEntrega_{individual} = \text{mín}(k_g \times NotaEntrega_{grupal}, NotaEntrega_{grupal} + 0,5)$$

donde,

$$k_g = \frac{sp+3}{7}$$

Los alumnos que no cumplan con enviar la coevaluación antes del Lunes 5 de Octubre a las 23:59 horas tendrán un **descuento de 0.5 puntos** en su nota de la entrega correspondiente.

8. 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.