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

IIC2343 – Arquitectura de Computadores (II/2015)

Proyecto Semestral: Entrega IV

Máquina programable con LCD externo

Fecha de entrega: Domingo 22 de Noviembre

1. Objetivo

Para esta entrega tendrán que mejorar una vez más el computador básico desarrollado durante las entregas pasadas, extendiendo la cantidad de inputs y agregando la capacidad de output. Además deberán agregar soporte para un LCD externo y el manejo de todos los leds disponibles.

En la Figura 1 se muestra el diagrama del computador básico modificado que tendrán que diseñar para esta entrega.

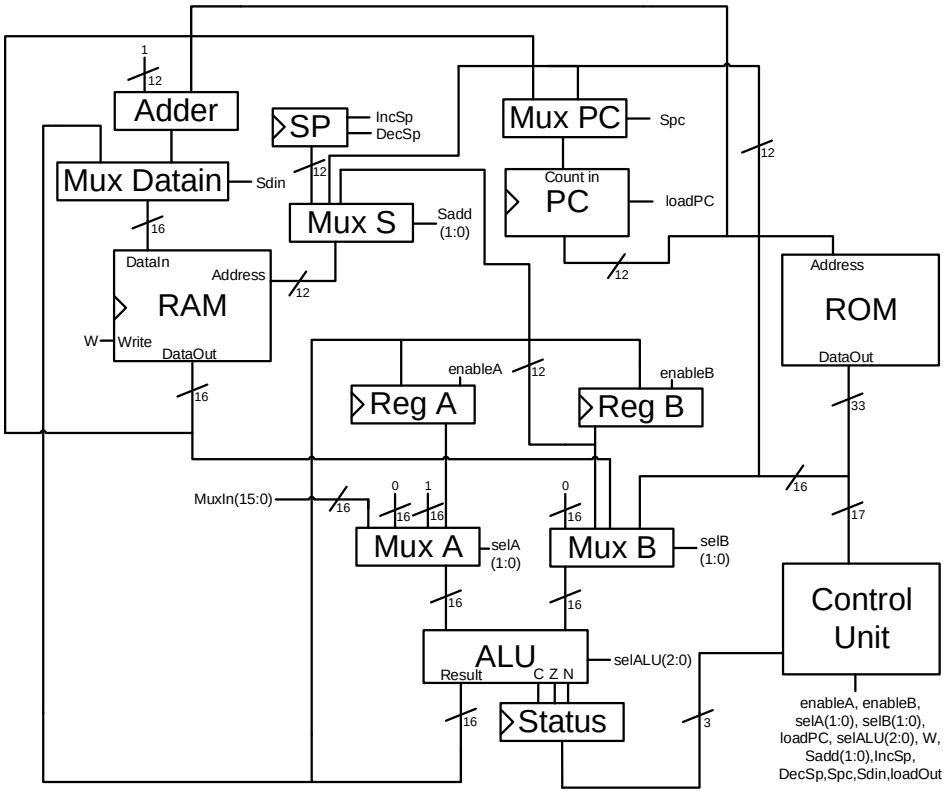

Figura 1: Computador básico.

Como se puede apreciar en la Figura 2, la implementacin del *MuxIn* cambia un poco, incluyendo un componente *Timer* que entrega 3 enteros de 16 bits, con la información de segundos, milisegundos y nanosegundos. Tambin se agrega un *Decoder OUT* y registros para almacenar la información a mostrar a través del display de 7 segmentos y leds de la placa.

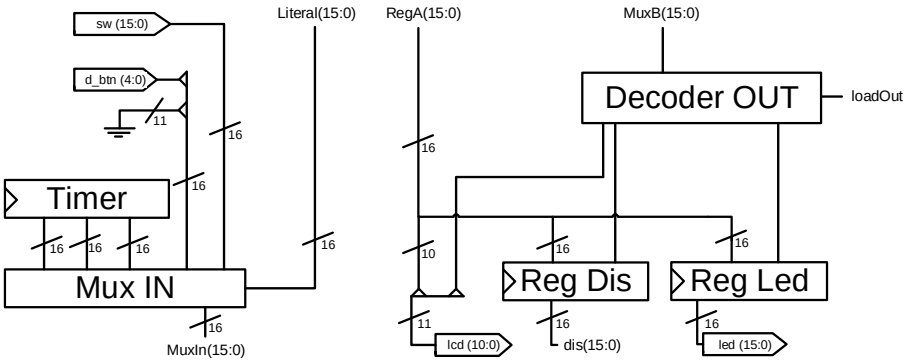

Figura 2: I/O detallado.

Para un manejo más sencillo del LCD externo, se les facilitará una librería y los manuales de hardware correspondientes.

2. Especificaciones CPU

A continuación se presenta el listado de las instrucciones nuevas que debe poder soportar la CPU (y por supuesto su *assembler*):

```
OUT A,B       (dirige A al puerto B)
    A,(B)     (dirige A al puerto Mem[B])
    A,(Dir)   (dirige A al puerto Mem[Dir])
    A,Lit     (dirige A al puerto Lit)
```

De esta forma, el listado completo de las instrucciones que debe poder soportar la CPU es:

```
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])
    A,(B)     (cargar Mem[B] en A)
    B,(B)     (cargar Mem[B] en B)
    (B),A     (guarda A en Mem[B])
    (B),Lit   (guarda Lit en Mem[B])
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])
    A,(B)     (guarda A+Mem[B] en A)
    B,(B)     (guarda A+Mem[B] en B)
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])
    A,(B)     (guarda A-Mem[B] en A)
    B,(B)     (guarda A-Mem[B] en B)
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])
    A,(B)     (guarda A and Mem[B] en A)
    B,(B)     (guarda A and Mem[B] en B)
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])
    A,(B)     (guarda A or Mem[B] en A)
    B,(B)     (guarda A or Mem[B] en B)
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])
    A,(B)     (guarda A xor Mem[B] en A)
    B,(B)     (guarda A xor Mem[B] en B)
NOT A         (guarda not A en A)
    B,A       (guarda not A en B)
    (Dir),A   (guarda not A en Mem[Dir])
    (B),A     (guarda not A en Mem[B])
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])
    (B),A     (guarda shift left A en Mem[B])
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])
    (B),A     (guarda shift right A en Mem[B])
INC A         (incrementa A en una unidad)
    B         (incrementa B en una unidad)
    (Dir)     (incrementa Mem[Dir] en una unidad)
    (B)       (incrementa Mem[B] 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])
    A,(B)     (hace A-Mem[B])
PUSH A         Mem[SP] = A, SP--
     B         Mem[SP] = B, SP--
POP  A         SP++, A = Mem[SP]
     B         SP++, B = Mem[SP]
CALL Dir       Mem[SP] = PC, PC = Dir, SP--
RET            SP++, PC = Mem[SP]
IN   A,Lit     A = Input[Lit]
     B,Lit     B = Input[Lit]
     (B),Lit   Mem[B] = Input[Lit]
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)
OUT A,B       (dirige A al puerto B)
    A,(B)     (dirige A al puerto Mem[B])
    A,(Dir)   (dirige A al puerto Mem[Dir])
    A,Lit     (dirige A al puerto Lit)
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 incluir el resto de los *leds* y el *LCD* en su CPU debe descomentar las lineas en Basys3.xdc, extender la salida *led* a 16 bits y crear la salida *lcd* de 11 bits 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.

3. Especificaciones *assembler*

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

1. Al soporte de literales se debe agregar que puedan ser ingresados como caracteres, de la forma 'c'.
2. Se debe poder definir caracteres como literal y arreglos de caracteres al principio de cada programa, los que deben ser de la forma

   ```
   DATA:

   letrac 'c'

   palabra "hola"
   ```

   Internamente, las frases se deben transformar en arreglos de caracteres terminados por un 0, por ejemplo [h, o, l, a, 0].

4. Especificaciones de los puertos a utilizar

Para estandarizar el uso de los componentes, se reservarán los siguientes puertos para entradas y salidas:

```
INPUT
0 Switches
1 Botones
2 Segundos
3 MSegundos
4 uSegundos

OUT
0 Display
1 Leds
2 LCD
```

5. Especificaciones de la funciones a programar

Para probar el computador hecho por cada grupo tendrán que generar un programa con interacción del usuario a través de los dispositivos de entrada y salida de la placa, usando el codigo assembly base que se pondrá a su disposición. Este programa es un pequeño juego que llamaremos MegaMental, en el cual dos jugadores se desafían mutuamente a decifrar el código de su oponente. Van jugando por turnos, donde primero un jugador crea un código y el otro intenta adivinarlo, y luego intercambian los roles.

Para su implementación en la placa, el juego se desarrolla de la siguiente manera:

1. El juego comienza pidiendo el nombre de ambos jugadores, donde cada uno puede ser de un máximo de 4 caracteres. El ingreso de estos nombres se debe hacer con los botones arriba, abajo, izquierda y derecha, y su confirmación se realiza con el botón central, mostrando el puntero solo durante la edición. Luego se debe mostrar en todo momento sus nombres y puntajes en la linea superior y los mensajes en la inferior.

2. Se le pide al jugador que no está de turno que ingrese un código secreto a través de los 16 switches de la placa, confirmándolo con el botón central. Luego, se le debe dar la oportunidad de cambiar la posición de los switches, antes de que el jugador de turno presiones el botón central para comenzar a adivinar.

3. El jugador de turno contará con 16 oportunidades para adivinar el código de su oponente. Estas "vidas"deben representarse con los leds de la placa.

4. Para adivinar el código del oponente, el jugador de turno cuenta con 5 herramientas a su favor, estas son las operaciones ADD, SUB, AND, OR y XOR. Seleccionando alguna de ellas a través de los botones, el jugador podrá operar entre el código de su oponente y una configuración de los switches que desee probar.

5. Luego de cada operación que el jugador ejecute, se debe mostrar en el display de 7 segmentos la cantidad de unos que tenga el resultado de la misma.

6. El turno termina declarando como ganador al jugador si este logra descifrar el código de su oponente antes de quedarse sin vidas. Si se le acaban las vidas, entonces se declarará como ganador del turno a su oponente. El ganador gana un punto.

7. Una vez que termina cada turno, se debe preguntar si se desea continuar. En caso que si lo deseen, los jugadores cambian de roles. En caso contrario se debe volver a iniciar el programa, solicitando los nombres de dos nuevos jugadores.

6. Ejemplos

A continuación se muestran unos cuantos ejemplos de código en *assembly* que los compiladores de cada grupo debiesen ser capaces de soportar[1]:

```
Programa 1:


DATA:

CODE:

MOV B,1
MOV A,ABCDh
OUT A,0
MOV A,1010101010101010b
OUT A,B

end:
JMP end
```

```
Programa 2:

DATA:

CODE:

loop:
IN A,2
OUT A,0
IN A,3
OUT A,1
JMP loop
```

[1]Esto quiere decir que si copian y pegan el código tal cual en sus compiladores, estos debiesen generar el archivo de salida correspondiente.

7. Entrega y consultas

Específicamente cuando se cumpla el plazo indicado al comienzo de este enunciado, 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 con el juego a programar 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).

Además se realizará una corrección presencial de esta entrega, en la que deberán presentar con su placa de desarrollo el o los programas que se les solicite en la ayudantía del Lunes 23 de Noviembre. 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.

Para entregar sus proyectos deben dejar todos los archivos solicitados en la carpeta compartida en Dropbox correspondiente a su grupo. La fecha de entrega vence el Domingo 22 de Noviembre a las 23:59 horas, entregas atrasadas serán **penalizadas con 0.5 puntos** por cada hora (o fracción) de atraso.

Cualquier pregunta sobre el proyecto, ya sean de enunciado, contenido o sobre aspectos administrativos deben comunicarse con los ayudantes:

- Francesca Lucchini: flucchini@uc.cl
- Jurgen Heysen: jdheysen@uc.cl
- Nicolás Elliott: ngelliot@uc.cl

8. Evaluación

Cada entrega 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 completar de forma **individual y obligatoria** el formulario web que los ayudantes pondrán a su dispocición, repartiendo un máximo de 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 en el plazo asignado tendrán un **descuento de 0.5 puntos** en su nota de la entrega correspondiente.

9. 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.