Taller de Microarquitectura

Organización del Computador 1

Primer Cuatrimestre 2019 - Turno Mañana

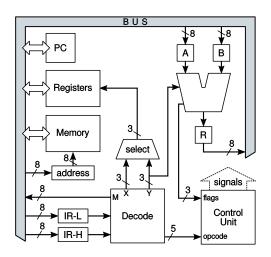
El presente taller consiste en analizar y extender una micro-arquitectura diseñada sobre el simulador *Logisim*. Se buscará codificar programas simples en ensamblador, modificar parte de la arquitectura y diseñar nuevas instrucciones.

El simulador se puede bajar desde la página http://www.cburch.com/logisim/ o de los repositorios de Ubuntu. Requiere Java 1.5 o superior. Para ejecutarlo, teclear en una consola java -jar logisim.jar.

Para este taller utilizaremos mayoritariamente el modo simulación. Sobre todo las opciones de "Enable clock ticks".

Además utilizaremos el componente memoria RAM, que inicia con todas sus posiciones en 0 y dos memorias ROM cuyo valor puede ser cargado desde un archivo desde la opción "Contents—Open...".

Procesador OrgaSmall



- Arquitectura von Neumann, memoria de datos e instrucciones compartida.
- 8 registros de propósito general, R0 a R7.
- 1 registro de propósito específico PC.
- Tamaño de palabra de 8 bits y de instrucciones 16 bits.
- Memoria de 256 palabras de 8 bits.
- Bus de 8 bits.
- Diseño microprogramado.

Se adjunta como parte de este taller las hojas de detalles del procesador OrgaSmall.

Ejercicios

- (1) Introducción Recorrer la máquina y la hoja de datos y responder:
 - a) ¿Cuál es el tamaño de la memoria?

- b) ¿Cuántas instruciones sin operandos se podrían agregar respetando el formato de instrucción actual?
- c) Sin respetar el formato de instrucción, ¿cuántas instrucciones sin operandos se pueden agregar?
- d) ¿Qué tamaño tiene el PC?
- e) ¿Dónde se encuentra y qué tamaño tiene el IR?

Charlengint 1 _	
Checkpoint 1 —	

- (2) **Analizar** Estudiar el funcionamiento de los circuitos indicados y responder las siguientes preguntas:
 - a) PC (Contador de Programa): ¿Qué función cumple la señal inc?
 - b) microOrgaSmall (DataPath): ¿Para qué sirve la señal DE_enOutImm? ¿Qué parte del circuito indica que índice del registro a leer y escribir?
 - c) ALU (Unidad Aritmético Lógica): ¿Qué función comple la señal opW?
 - d) ControlUnit (Unidad de control): ¿Cómo se resuelven los saltos condicionales? Describir el mecanismo.
 - e) Realizar un *data-path* que no incluya la unidad de control pero en la cual los componentes tengan las señales de las operaciones que realizan.

Cl1: 0	
Checkboint 2	

(3) Ensamblar y correr - Escribir en un archivo, compilar, cargar el siguiente programa:

JMP seguir
seguir:
SET RO, OxFF
SET R1, Ox11
siguiente:
ADD RO, R1
JC siguiente

halt:
JMP halt

Para ensamblar un archivo .asm el comando a ejecutar es: python assembler.py NombreDeArchivo.asm Esto generará un archivo .mem que puede ser cargado en la memoria RAM de la máquina.

a) Antes de correr el programa, identificar el comportamiento esperado.

- b) ¿Qué lugar ocupará cada instrucción en la memoria? Detallar el por qué valor se reemplazarán las etiquetas.
- c) Ejecutar y controlar ¿cuántos ciclos de clock son necesarios para que este codigo llegue a la instrucción JMP halt?
- d) ¿Cuántas microinstrucciones son necesarias para realizar el ADD?¿Cuántas para el salto?
- e) Asumiendo que el clock oscila a 32Hz. El simulador cuenta con un contador de clocks reiniciable manualmente para facilitar dicha tarea. ¿Cuánto tarda cada instrucción (en ms) y cuánto la ejecución?

Cll : t 2	
Спескропи 5	, -

- (4) **Analizar** Cada componente tiene una X sobre un cable. Para cada uno de ellos indicar qué sucedería si ese cable:
 - a) Se corta
 - b) Vale siempre 0 (la cantidad de bits necesarios)
 - c) Vale siempre 1 (la cantidad de bits necesarios)

01 1 1 4	
——— Checkpoint 4 ———	

- (5) **Programar** Escribir en ASM los siguientes programas:
 - a) Escribir un programa que calcule la suma de los primeros n números naturales. El valor resultante debe guardarse en R1, y se espera que el valor de n sea leido de R0.

Charles sint 5	
— Cneckpoint 5	

Para generar un nuevo set de micro-instrucciones, generar un archivo .ops y traducirlo con el comando:

python buildMicroOps.py NombreDeArchivo.ops

Esto generará un archivo .mem que puede ser cargado en la memoria ROM de la unidad de control.

- (6) **Ampliando la máquina** Agregar las siguientes instrucciones listadas a continuación a la máquina teniendo en cuenta que para cada una:
 - se debe plantear el funcionamiento esperado en RTL (Register transfer language).
 - se debe implementar la funcionalidad pedida.
 - se debe presentar con un código de ejemplo que pruebe la funcionalidad agregada.
 - a) Sin agregar circuitos nuevos, agregar la instrucción SIG que dado un registro, aumenta su valor en 1. Esta operación **no modifica** los flags. Utilizar como código de operación el 0x09.

- b) Sin agregar circuitos nuevos, agregar la instrucción NEG que obtenga el inverso aditivo de un número sin modificar los flags. Nota: el inverso aditivo de un número se puede obtener como xor(XXXX, 0xFFFF)+0x0001. Utilizar como código de operación el 0x0A.
- c) Implementar un cirtuito que dados dos números A_{7-0} y B_{7-0} los combine de forma que el resultado sea B_1 A_6 B_3 A_4 B_5 A_2 B_7 A_0 . Apartir de este componente agregar una instrucción que permita tener esta funcionalidad.

Cl
 Checkboint o

(7) Optativos - Otras modificaciones interesantes:

- a) Modificar las instrucciones JC, JZ, JN y JMP, para que tomen como parámetro un registro.
- b) Agregar las instrucciones CALL y RET. Considerar que uno de los parámetros de ambas instrucciones es un índice de registro que se utilizará como Stack Pointer. La instrucción CALL toma como parámetro, además, un inmediato de 8 bits que indica la dirección de memoria donde saltar.
- c) Modificar el circuito agregando las conexiones necesarias para codificar las instrucciones STR [RX+cte5], Ry y LOAD Ry, [RX+cte5]. Este par de instrucciones son modificaciones a las existentes, considerando que cte5 es una constante de 5 bits en complemento a dos.
- d) Agregar instrucciones que permitan leer y escribir los flags. Describir las modificaciones realizadas sobre el circuito.