Taller de Microprogramación

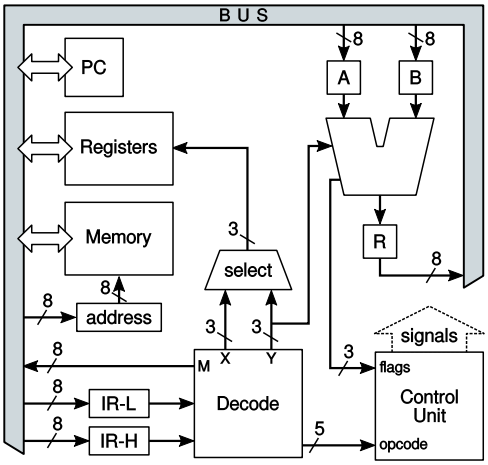
Organización del Computador 1

Cuatrimestre Verano 2019

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`.

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) **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) `ALU` (Unidad Aritmético Lógica): ¿Qué función comple la señal `opW`?

c) `ControlUnit` (Unidad de control): ¿Cómo se resuelven los saltos condicionales? Describir el mecanismo.

d) `microOrgaSmall` (`DataPath`): ¿Para qué sirve la señal `DE_enOutImm`? ¿Qué parte del circuito indica que índice del registro a leer y escribir?

(2) **Programar** - Escribir en ASM los siguientes programas:

a) Calcular el máximo común divisor. (referencia 10 instrucciones)

```
mcd(a, b)
  while a != b
    if a > b
      a = a - b
    else
      b = b - a
  return a
```

b) Calcular el i-esimo número de Fibonacci. (referencia 17 instrucciones)

```
fib(n)
  a=1
  b=1
  for i in 0 to n:
    tmp = a + b
    a = b
    b = tmp
  return a
```

(3) **Modificar** - Agregar las siguientes nuevas instrucciones:

a) Sin agregar circuitos nuevos, agregar una instrucción que obtenga el inverso aditivo de un número sin modificar los flags.

b) Modificando el circuito de la `ALU`, agregar una instrucción que combine dos números $\texttt{A}_{7-0}$ y $\texttt{B}_{7-0}$, tal que el resultado sea $\texttt{B}_1$ $\texttt{A}_6$ $\texttt{B}_3$ $\texttt{A}_4$ $\texttt{B}_5$ $\texttt{A}_2$ $\texttt{B}_7$ $\texttt{A}_0$.

Nota: Cada instrucción debe ser utilizada en un código de ejemplo que se presentará como parte de la solución del taller.

(4) **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.