

## Segundo Cuatrimestre 2023

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...”.

- 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.
- Direcciones de 8 bits.
- Direccionamiento a Byte.
- Bus de 8 bits.
- Diseño microprogramado.

Tiene internamente 3 ejemplares del componente llamado **registro-salida-restringida**, que se encuentra incompleto. Este componente tiene la capacidad de almacenar un valor interno. Completarlo de manera que su comportamiento sea:

- Cuando la señal de entrada **clk** pasa a su flanco alto, si la señal de entrada **W** vale 1, entonces se debe almacenar internamente el valor de la entrada **in**. En caso de que no se cumpla alguna condición de las antes nombradas el valor interno no se modifica.
- Cuando la señal de entrada **en\_out** vale 1, en la salida **out** se debe poder leer el valor interno almacenado. Caso contrario esta salida debe estar en alta impedancia (**Hi-Z**).
- Por motivos didácticos se pide también una salida de debug para poder observar el valor interno almacenado (salida **out\_debug**) todo el tiempo.

Una vez completado el componente anterior, analizar el componente **ej-2** y responder:

- a) ¿Cuáles son y qué representa cada entrada y cada salida del componente? ¿Cuáles entradas deben ser consideradas como de control?
- b) Las entradas **input\_bit** y **en\_input\_bit** sirven para poder introducir en el circuito un valor arbitrario. Escribir una secuencia de activación y desactivación de entradas para que el registro R1 pase a tener el valor 1.
- c) Dar una secuencia de activaciones que inicialmente ponga un valor arbitrario en R0 (suponer para un valor y luego generalizarlo), luego que este valor se transfiera a R1, luego que el valor de R2 pase a R0 y finalmente el valor de R1 a R2.

---

Checkpoint 1

---

- (2) **Analizar** - Recorrer la máquina y la hoja de datos, estudiar el funcionamiento de los circuitos indicados y responder las siguientes preguntas:

- a) ¿Cuál es el tamaño de la memoria?
- b) ¿Qué tamaño tiene el PC?
- c) ¿Cuántas instrucciones se podrían agregar respetando el formato de instrucción actual?

**Mirando los módulos indicados de hardware:**

- a) **PC (Contador de Programa):** ¿Qué función cumple la señal **inc**?
- b) **ALU (Unidad Aritmético Lógica):** ¿Qué función cumple la señal **opW**?
- c) **microOrgaSmall (DataPath):** ¿Para qué sirve la señal **DE\_enOutImm**? ¿Qué parte del circuito indica que registro se va a leer y escribir?
- d) **ControlUnit (Unidad de control):** ¿Cómo se resuelven los saltos condicionales? Describir el mecanismo.

---

Checkpoint 2

---

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

```

    JMP seguir

seguir:
    SET R0, 0xFF
    SET R1, 0x11

siguiente:
    ADD R0, R1
    JC siguiente

halt:
    JMP halt

```

Para ensamblar un archivo `.asm` ejecutar el comando:  
`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 por qué valor se reemplazarán las etiquetas.
- c) Ejecutar y controlar ¿cuántos ciclos de clock son necesarios para que este código llegue a la instrucción `JMP halt`?
- d) ¿Cuántas microinstrucciones son necesarias para realizar el `ADD`? ¿Cuántas para el salto?

---

Checkpoint 3

---

(4) **Ampliando la máquina** - Agregar las siguientes instrucciones nuevas:

Nota1: Los siguientes ítems deben ser presentados mediante un código de ejemplo que pruebe la funcionalidad agregada. Pueden agregar las instrucciones al programita `asm` del checkpoint anterior.

Nota2: Tener en cuenta que si se agrega una operación, será necesario agregar el nombre mnemotécnico y el opcode en el archivo `"assembler.py"` y si se agregan operaciones en la ALU, de la misma manera agregarlas en el archivo `"buildMicroOps.py"`.

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.

- 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) Implementar un circuito que dados dos números  $A_{7-0}$  y  $B_{7-0}$  los combine de forma tal que el resultado sea  $B_1 A_6 B_3 A_4 B_5 A_2 B_7 A_0$ . Agregar la instrucción **MIX** que aplique dicha operación entre dos registros, asignándole un código de operación a elección.

---

Checkpoint 4

---

(5) **Optativos** - Otras modificaciones interesantes:

- a) 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(XX, 0xFF)+0x01`. Utilizar como código de operación el 0x0A.
- b) Modificar las instrucciones **JC**, **JZ**, **JN** y **JMP** para que tomen como parámetro un registro.
- c) 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**. Además, la instrucción **CALL** toma como parámetro un inmediato de 8 bits que indica la dirección de memoria a donde saltar.
- d) 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.
- e) Agregar instrucciones que permitan leer y escribir los **flags**. Describir las modificaciones realizadas sobre el circuito.