Auxiliar 8 - Assembler + Circuitos

Otoño 2025

Profesores: Alexandra Ibarra
Luis Mateu
Rodrigo Urrea

Auxiliares: Luciano Márquez
Tomás Vergara

Resumen

Compuertas lógicas

Las compuertas lógicas son operaciones entre dos entradas de bits, que pueden ser AND, OR, XOR, etc. En el caso del NOT, la entrada es única.

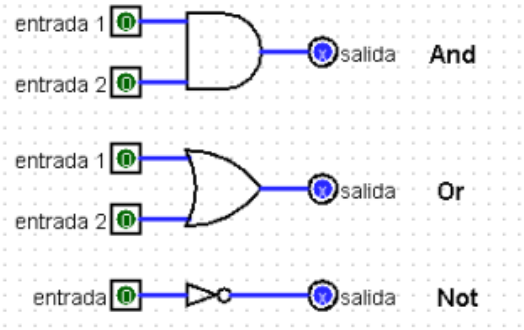


Figura 1: Compuertas

Módulos aritméticos

Los módulos aritméticos aplican operaciones entre dos entradas. A estos se les debe asignar la cantidad de bits que reciben en la entrada. En algunos módulos, como el *shift*, el orden de las entradas es importante para definir un resultado.

Figura 2: Algunos módulos aritmeticos

Módulos de memoria

Registros: Permiten almacenar datos, tienen las siguientes conexiones:

- **Pin D**: el dato que se quiere guardar
- **Pin en**: si está en 1 o no conectado, almacena el dato desde el pin D. si está en 0, se queda intacto.
- **Pin clk**: La entrada del reloj, que oscila entre 0 y 1. El disparador debe estar en modo `falling edge`.
- **Pin Q**: Salida que muestra el último dato almacenado en el registro.

ROM y RAM: las memorias ROM y RAM funcionan como una tabla de datos. En el caso de la RAM, también se puede escribir, para lo cual se necesita una dirección, datos de entrada y un reloj. También retorna el valor almacenado en esa dirección.

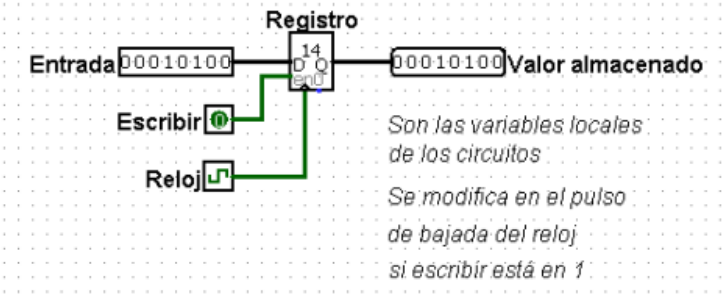


Figura 3: Un registro y sus conexiones

Multiplexor

El multiplexor funciona como un `if`, con tres pines: dos entradas de datos y un selector. El selector decide cuál de las entradas será el valor de la salida.

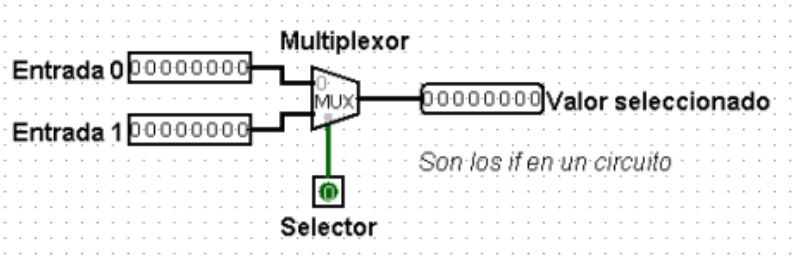


Figura 4: Un multiplexor

Preguntas

P1 [P2.a Control 2 2023-2)]

La función `f` de la derecha está programada en assembler Risc-V. Considere que se invoca `f` recibiendo en `a1` el entero 5 y en `a0` la dirección de `d` de un arreglo de 4 enteros con los valores 3, 8, 1 y 0. La siguiente tabla muestra la ejecución de la función hasta la 2da ejecución de `lw`

```
1   f:                                  Risc-V
2       mv      a4,a0
3   .L1
4       lw      a5,0(a4)
5       addi    a4,a4,4
6       bge     a5,a1,.L2
7       sw      a5,0(a0)
8       addi    a0,a0,4
9   .L2
10      bne     a5,zero,.L1
11      ret
```

Instrucción	a0	a1	a4	a5	arreglo d
	d	5			3 8 1 0
`mv a4,a0`			d		
`lw a5,0(a4)`				3	
`addi a4,a4,4`			d+4		
`bge a5,a1,.L2`			no salta		
`sw a5,0(a0)`					3 8 1 0
`addi a0,a0,4`	d+4				
`bne a5,zero,.L1`			salta		
`lw a5,0(a4)`				8	

Prosiga llenando la tabla a partir de la ejecución de `addi a4,a4,4` hasta que se ejecute la instrucción `ret`.

P2.

a) Diseñe el circuito `SumN` de la figura que recibe un número entero **x** de 8 bits y le suma la cantidad **n** de 8 bits en **varios** ciclos del reloj, dejando el resultado en la salida **z**. La entrada **start** del circuito determina el comienzo de una nueva suma. Por otro lado, la salida **rdy** debe permanecer en 0 y sólo cambiar a 1 cuando se haya terminado la operación. **No se puede usar un sumador que sume directamente x con n**.

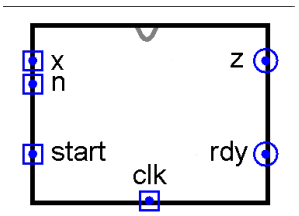


Figura 5: Circuito SumN

b) Para el circuito diseñado en la parte anterior, complete la tabla a partir del primer ciclo con los valores de las siguientes salidas, con las entradas **x=0b00101100** y **n=0b00000011**:

Ciclo	1	2	3	4	5	6
start	0	1	0	0	0	0
RegX	0	0	101100			
RegN						
sumador						
rdy						

Recuerde que un ciclo del reloj inicia con el cambio de 1 a 0 de clk y termina con el siguiente cambio de 1 a 0 de clk.

P3.

El circuito `maxHex` debe buscar en su entrada **x**, un entero de 32 bits, la mayor cifra hexadecimal siguiendo el siguiente algoritmo:

```c
typedef unsigned int uint32;
uint32 maxHex(uint32 x){
  uint32 mask = ~(-1U << 4);
  uint32 max = 0;
  while(x){
      uint32 n = (x & mask);
      if(n > max)
          max = n;
      x >>= 4;
  }
  return max;
}
```

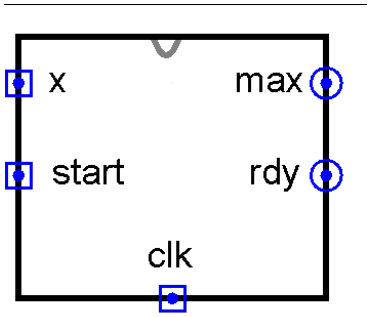
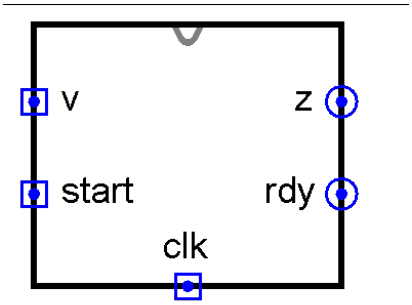


Figura 6: Circuito MaxHex

La búsqueda comienza cuando se detecta que **start** es 1 justo en el momento en que la entrada **clk** pasa de 1 a 0. En ese instante la salida **rdy** debe ir a 0 y permanecer en 0 mientras se realiza el cálculo. La entrada **x** permanecerá constante hasta que se solicite una nueva búsqueda llevando **start** a 1.

P4. (Propuesto)

En la entrada **v**, de 32 bits, el circuito `VectorSum` recibe cuatro vectores, de 8 bits cada uno. En donde los 4 bits más significativos corresponden a la coordenada x y los 4 menos significativos a y. Cuando **start** se pone en 1, debe llevar la salida **rdy** a 0 y calcular en varios ciclos del reloj el resultado de sumar los cuatro vectores. Al terminar el cálculo debe colocar **rdy** en 1 y entregar el resultado de la suma en la salida **z**. **Asuma que el valor máximo de cada coordenada es de 15**



Por ejemplo, la entrada *0b0001 0010 0110 0100*, los vectores son (1, 2) y (6, 4). El resultado de esta operación sería la salida *0b0111 0110* o el vector (7, 6).