DCC

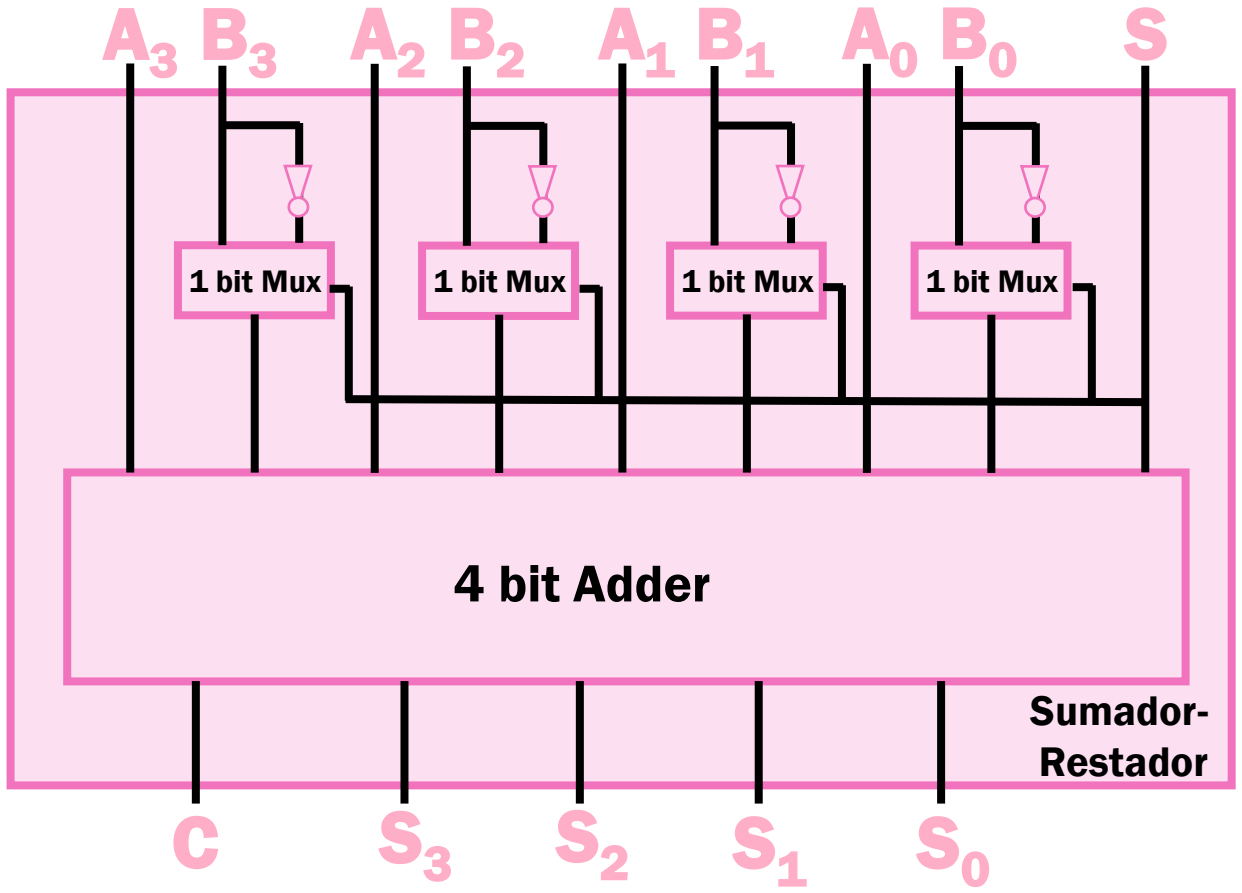
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 4 – Almacenamiento de Datos

Susana Figueroa

Sumador-Restador



- Usaremos S para indicar si la operación que se quiere realizar es una suma o resta.
  - S = 0 indica suma
  - S = 1 indica resta

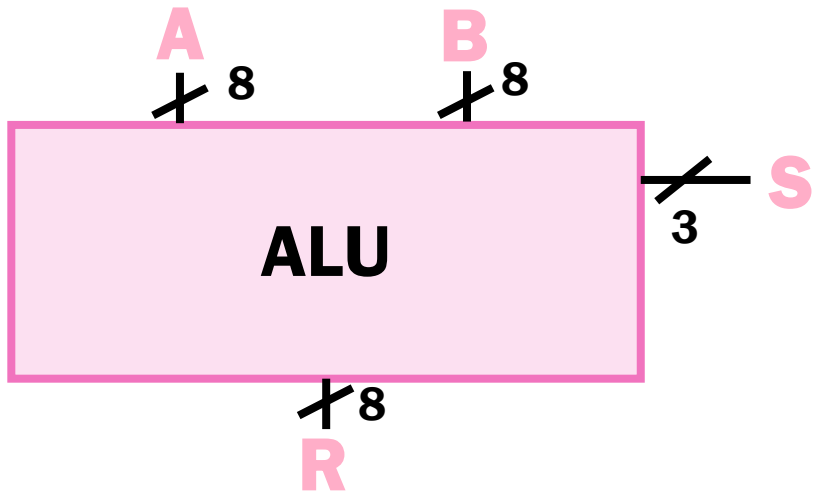
- El S nos sirve como selector para los multiplexores. Si S = 1 se usara la negación de B como input para el sumador.
- También el S nos indica el carry inicial para el sumador.

ALU

- Que es la ALU? Es la unidad aritmética lógica de un computador.
- Es un circuito que se encarga de realizar operaciones lógicas (como AND y OR) y aritméticas (como la suma y resta) sobre dos operandos.

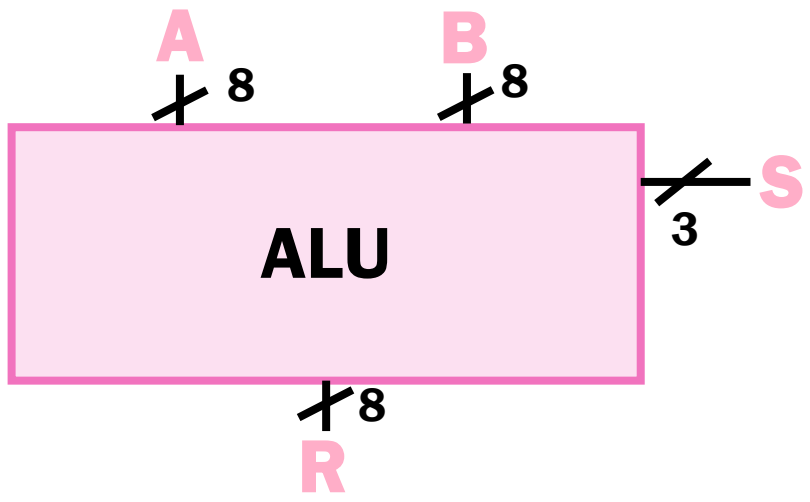
- Los operandos que utiliza: A y B, van a ser de n bits cada uno.
- El resultado R de la operación es también de n bits.
- Tiene una señal de control S de m bits que permite seleccionar entre máximo 2^m operaciones.

Por ahora vamos a considerar que nuestro computador básico tiene solamente 8 operaciones (m=3) y opera sobre números de 8 bits (n=8).



ALU

- Con esto tenemos lista nuestra ALU de 8 operaciones.



S_2	S_1	S_0	Operación
0	0	0	Suma
0	0	1	Resta
0	1	0	AND
0	1	1	OR
1	0	0	NOT
1	0	1	XOR
1	1	0	SHL
1	1	1	SHR

- El R corresponderá al resultado de la operación que se “ejecutó” según lo que indica S.
  - S = 100, entonces R = NOT A
  - S = 001, entonces R = A - B

... Overflow

- Hay que tener ciertas consideraciones al sumar y restar números en binario.
- Como vamos a tener números de una cantidad fija de bits (por ejemplo 8 bits), el resultado de las operaciones también debe ser del mismo tamaño. Pero van a haber veces donde el resultado real no va a caber en la cantidad de bits que tenemos para representar.

```
  11001011
+ 01001010
----------
 100010101
```

- Cuál es el problema?

... Overflow

- Hay que tener ciertas consideraciones al sumar y restar números en binario.
- Como vamos a tener números de una cantidad fija de bits (por ejemplo 8 bits), el resultado de las operaciones también debe ser del mismo tamaño. Pero van a haber veces donde el resultado real no va a caber en la cantidad de bits que tenemos para representar.

```
  11001011
+ 01001010
----------
 X00010101
```

- Cuál es el problema? El número que “verá” el computador es el número de 8 bits, pero este número no “calza” con la magnitud esperada del resultado.
- Esto se vuelve aún más problemático cuando estamos trabajando con números con signo.

... Overflow

- Consideremos la siguiente suma de números **con** signo.
- Se ve todo bien...

```
  01001011   (75)
+ 01001010   (74)
  ---------
  10010101   (149)
```

... Overflow

- Consideremos la siguiente suma de números **con** signo.
- Se ve todo bien… Noooo!!
- Como estamos trabajando con números con signo, el resultado también tiene signo.

```
  01001011   (75)
+ 01001010   (74)
  ---------
  10010101   (149)
```

... Overflow

- Consideremos la siguiente suma de números **con** signo.
- Se ve todo bien… Noooo!!
- Como estamos trabajando con números con signo, el resultado también tiene signo.

```
  01001011   (75)
+ 01001010   (74)
----------
  10010101   (-107)
```

- Qué pasó? Sumamos dos números positivos y el resultado nos dio negativo.
- Cada vez que se produzca un cambio de signo inesperado, lo llamaremos ***overflow***.
- Esto puede pasar tanto en la suma como en la resta.

... Overflow

- Consideremos la resta de 109 con -19

```
   01101101   (109)
-  11101101   (-19)
```

... Overflow

- Consideremos la resta de 109 con -19
- Es correcto el resultado?

```
  01101101   (109)
+ 00010011   (19)
----------
  10000000   (128)
```

... Overflow

- Consideremos la resta de 109 con -19
- Es correcto el resultado? Nuevamente tenemos que considerar el signo del resultado.

```
  01101101   (109)
+ 00010011   (19)
----------
  10000000   (-128)
```

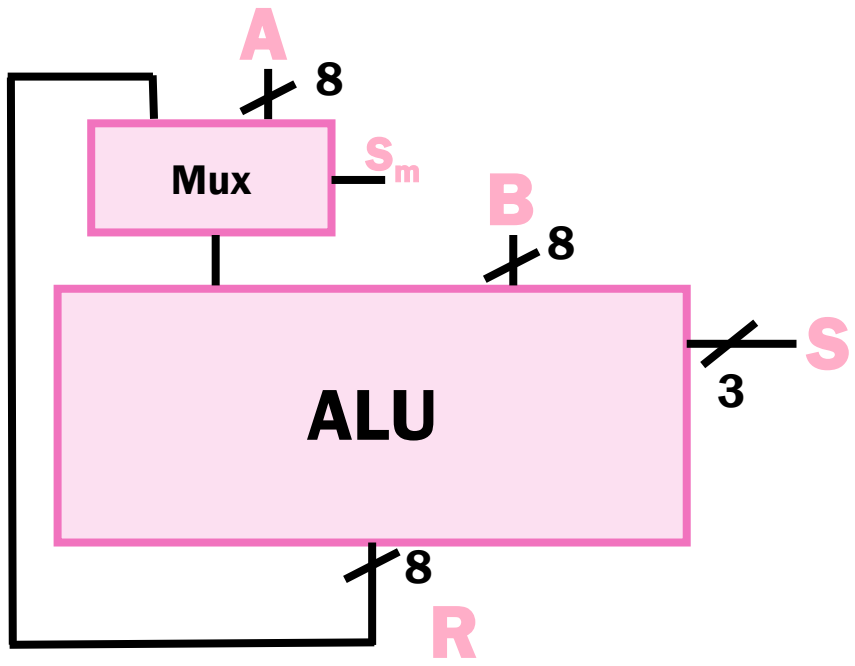
- Ahora restamos un numero positivo grande, con un numero negativo chico y nos dio un numero negativo de resultado. Esto corresponde a un ***overflow***.
- Hay más casos de overflows, pueden intentar buscarlos por su cuenta :)

ALU

- Con lo que tenemos hasta ahora somos capaces de realizar 8 operaciones diferentes sobre 2 números de 8 bits, pero que pasa si queremos usar los resultados anteriores para una operación?... Hasta ahora no podemos.

- Que podríamos hacer para lograr esto? Podríamos conectar la salida de la ALU a uno de los operandos a través de un multiplexor.

Qué problemas nos vamos a encontrar?

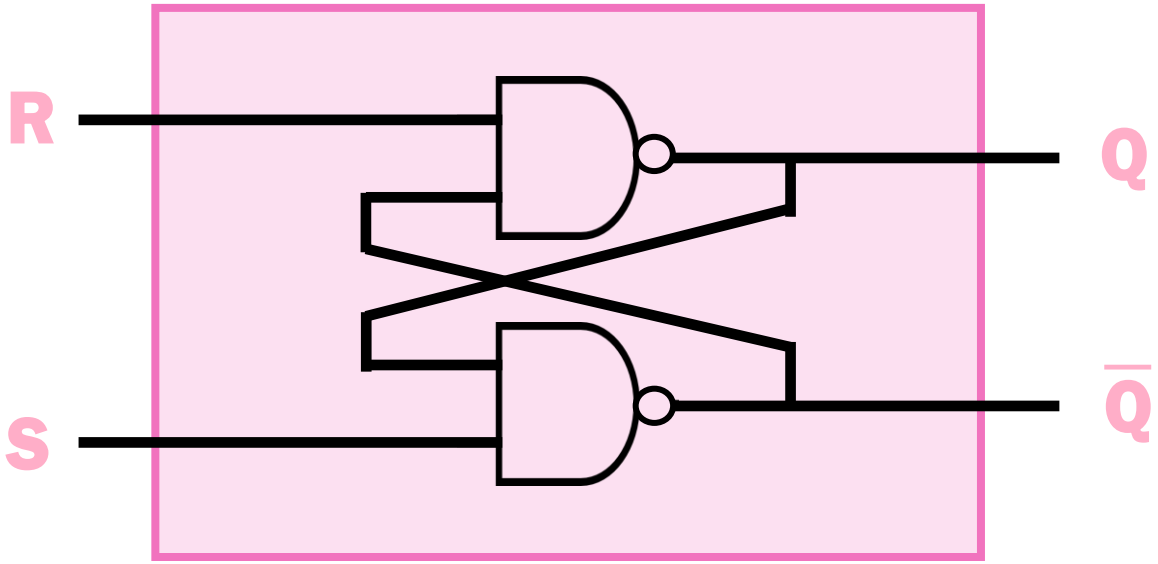


Latch

- Para lograr que valores perduren, necesitamos agregar memoria a nuestro computador. Pero es más fácil decirlo que hacerlo.
- Necesitamos construir un circuito que nos permita mantener información en el tiempo.
- El circuito más básico que veremos que permite preservar estados anteriores es el **latch**. Este se construye solamente con 2 compuertas NAND.

Latch RS

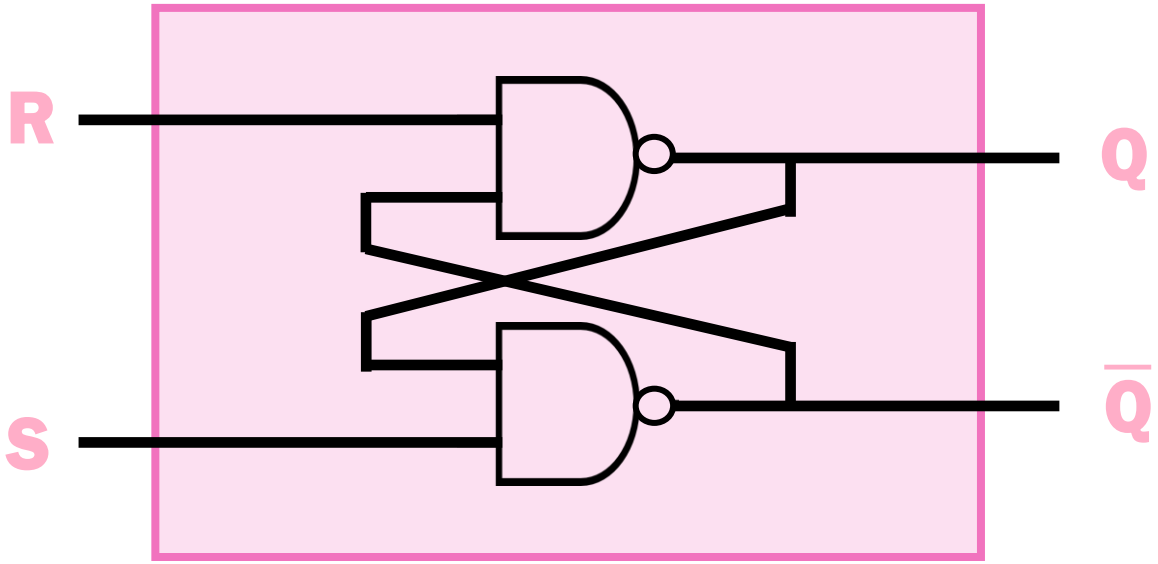
- Este tipo de **latch** es de **reset-set.**
- Nos permite *setear* el valor. Cuando set = 1, $Q_t = 1$.
- También podemos *resetear* el valor. Cuando reset = 1, $Q_t = 0$.
- O podemos conservar el estado anterior. Si set = reset = 0 entonces $Q_t = Q_{t-1}$.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

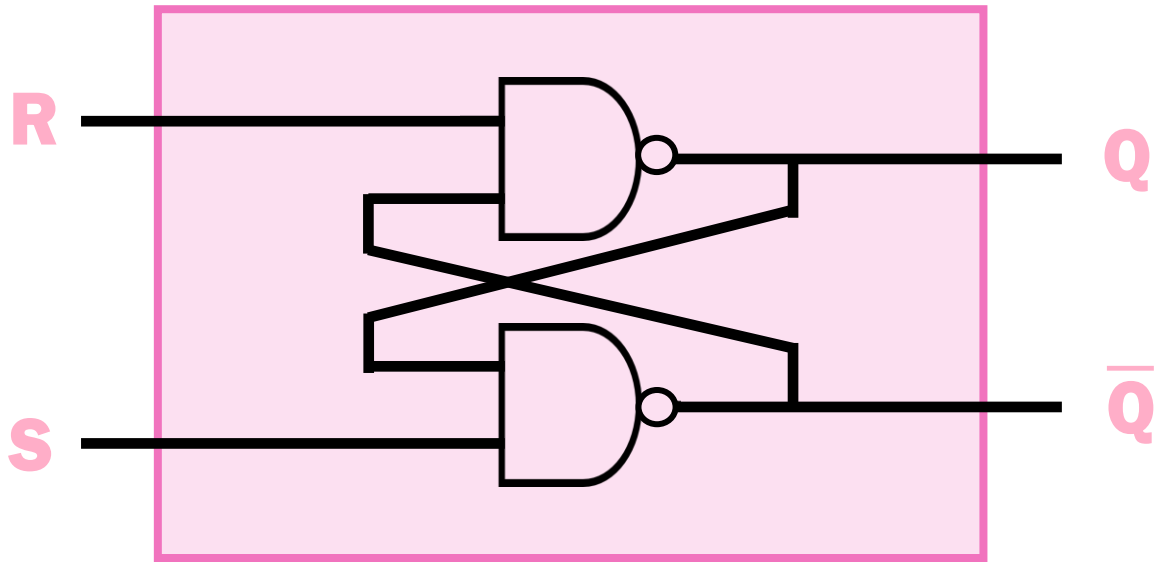
- Veamos paso a paso cómo funciona.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

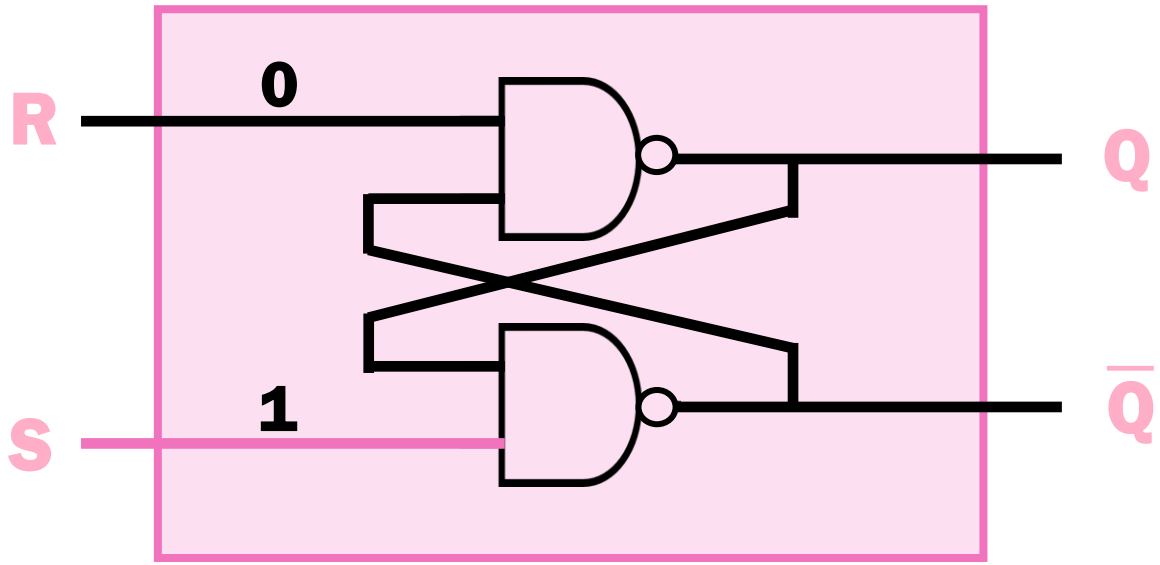
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

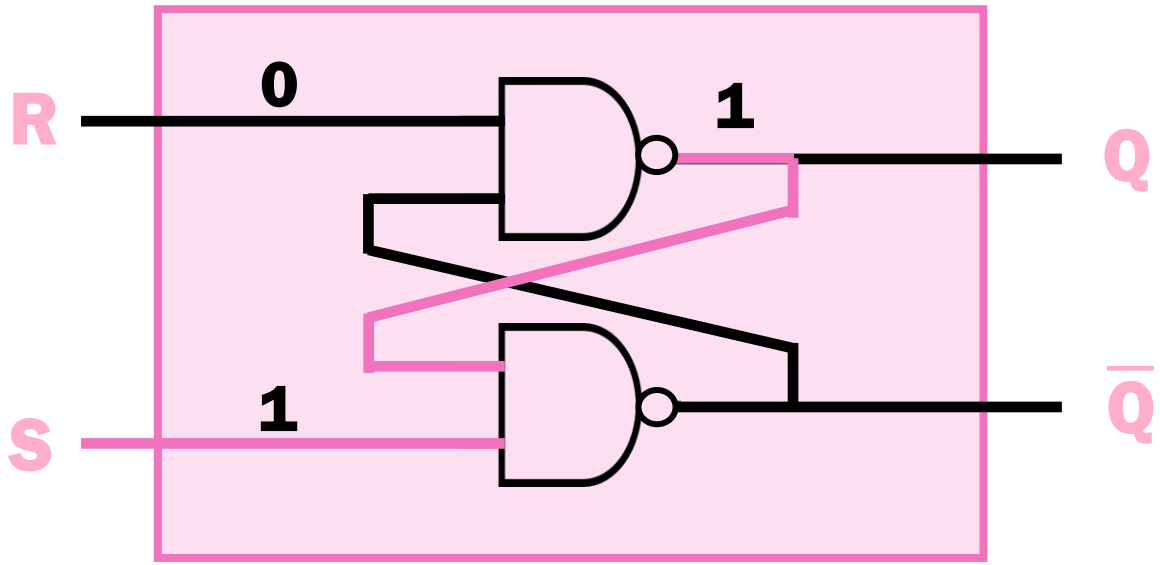
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

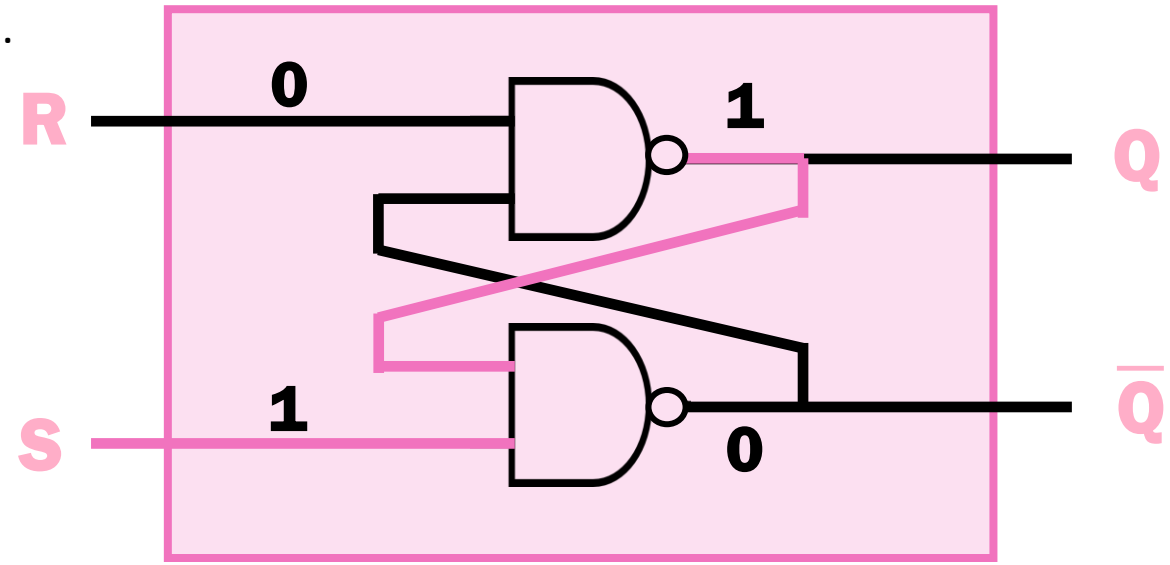
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

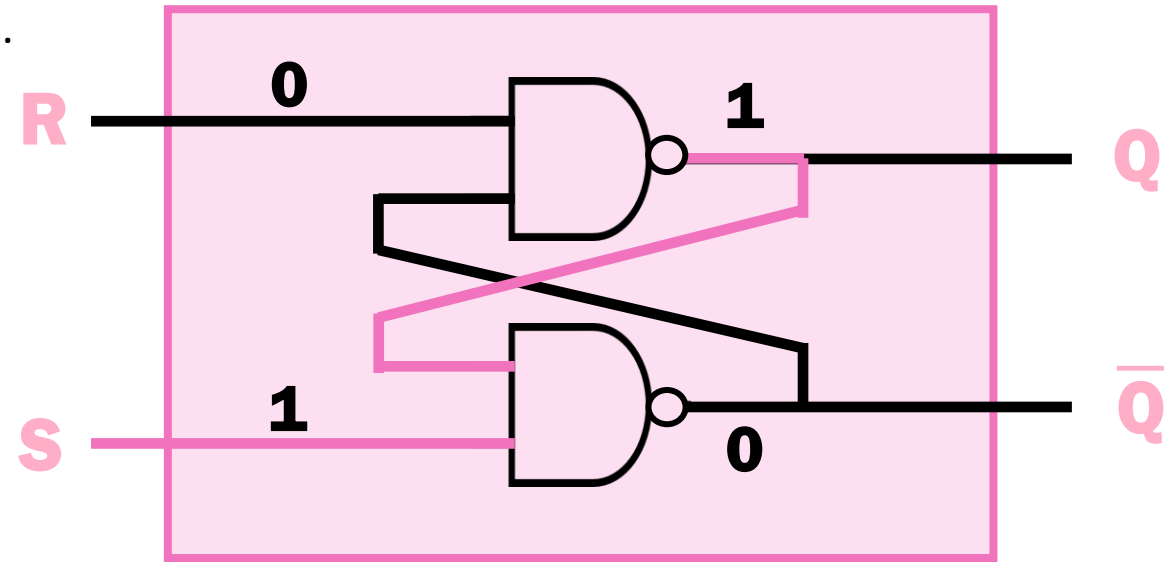
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

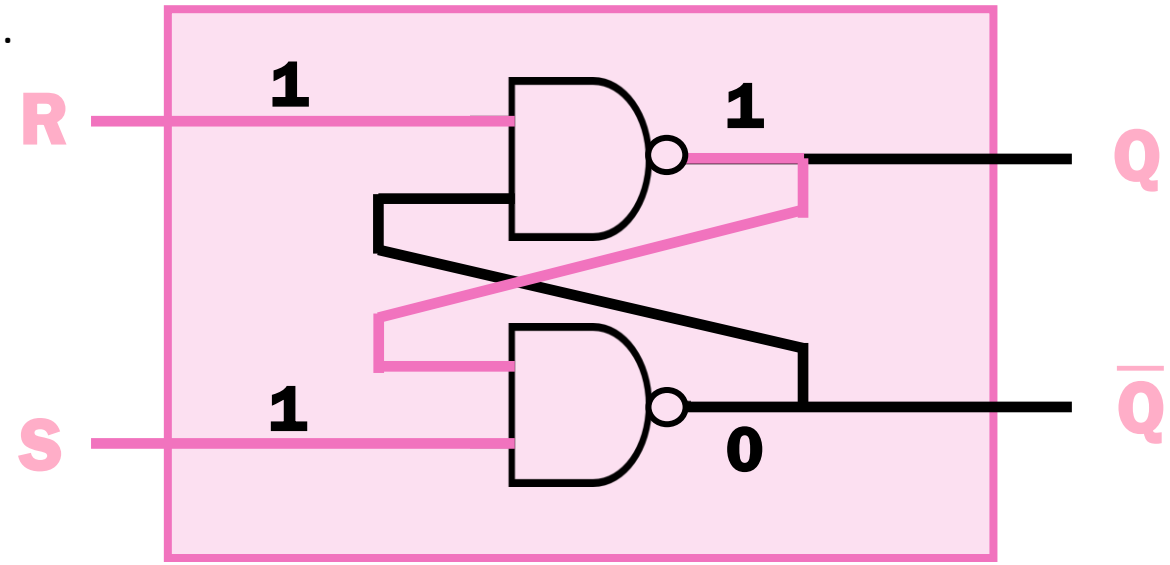
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

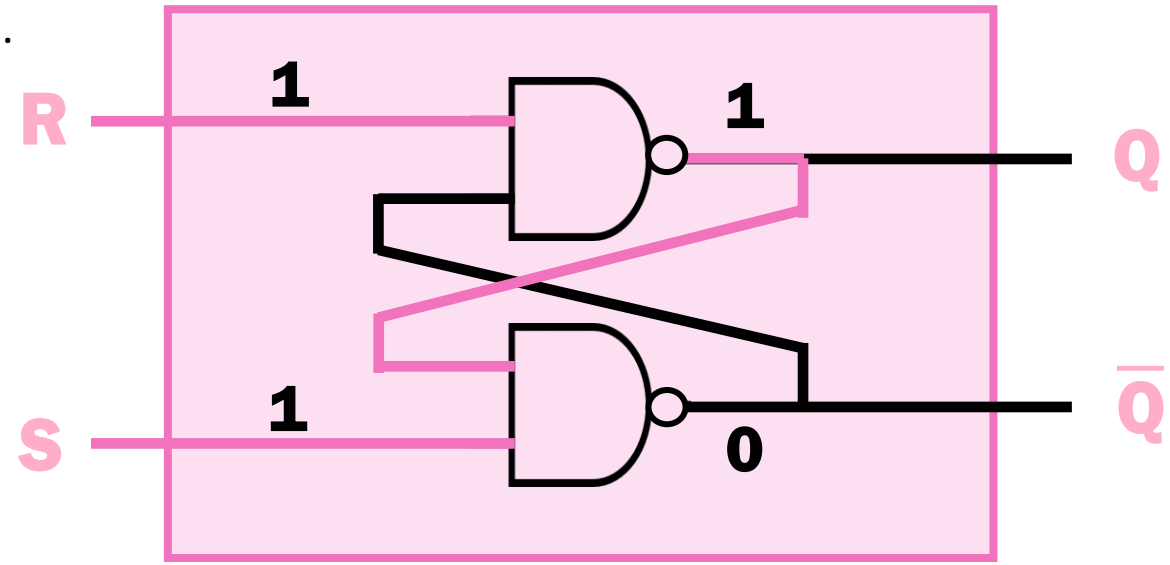
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

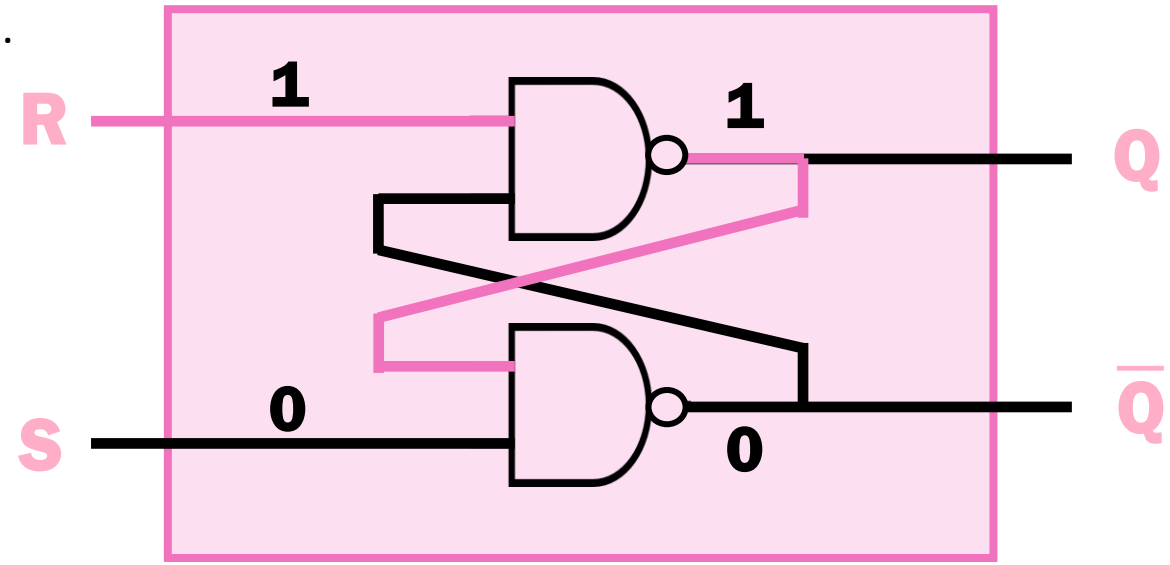
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

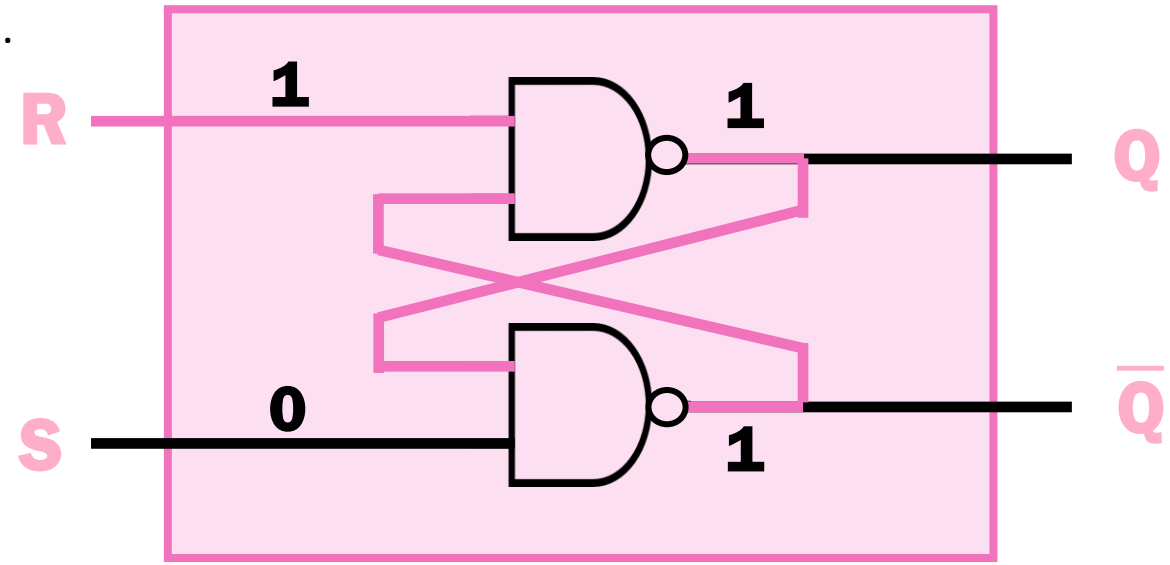
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

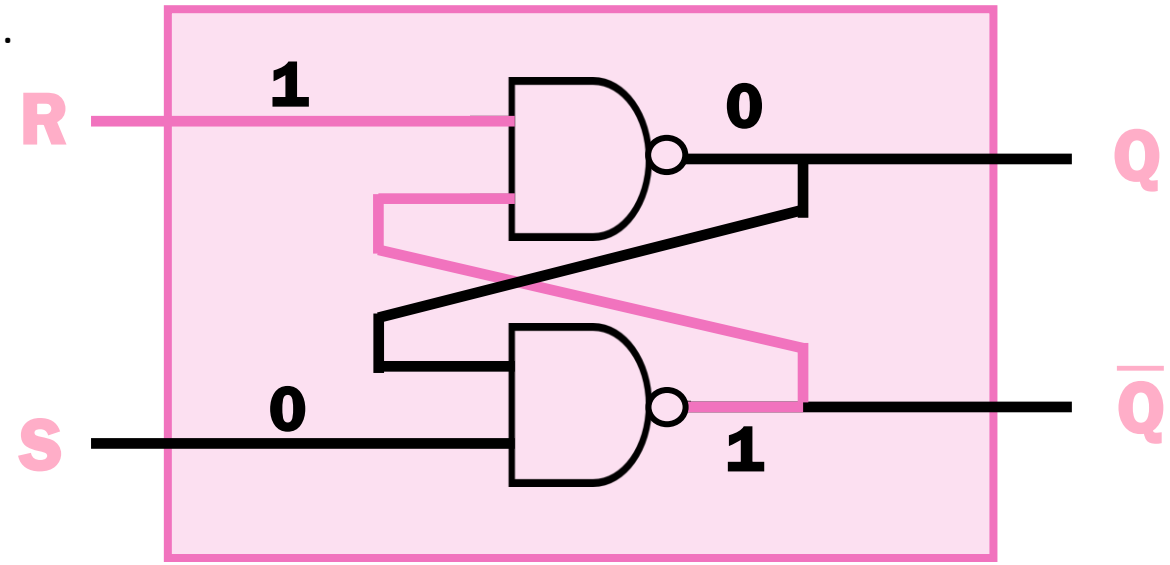
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0.



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0. Q toma el valor de 0.

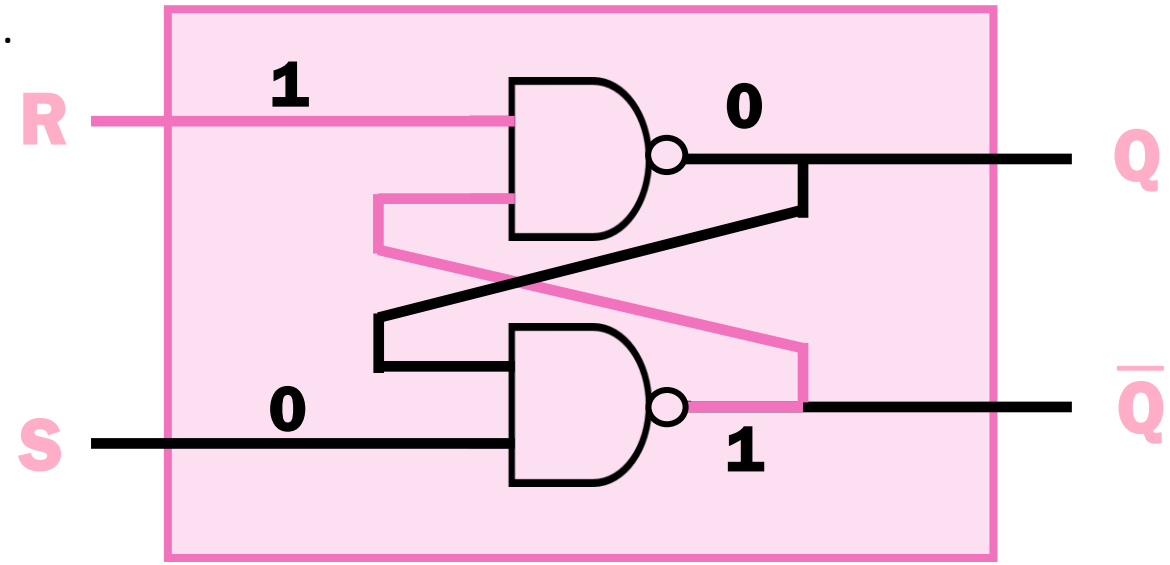


R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch RS

- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0. Q toma el valor de 0.
- Ahora, nuevamente si R=1 y S=1.

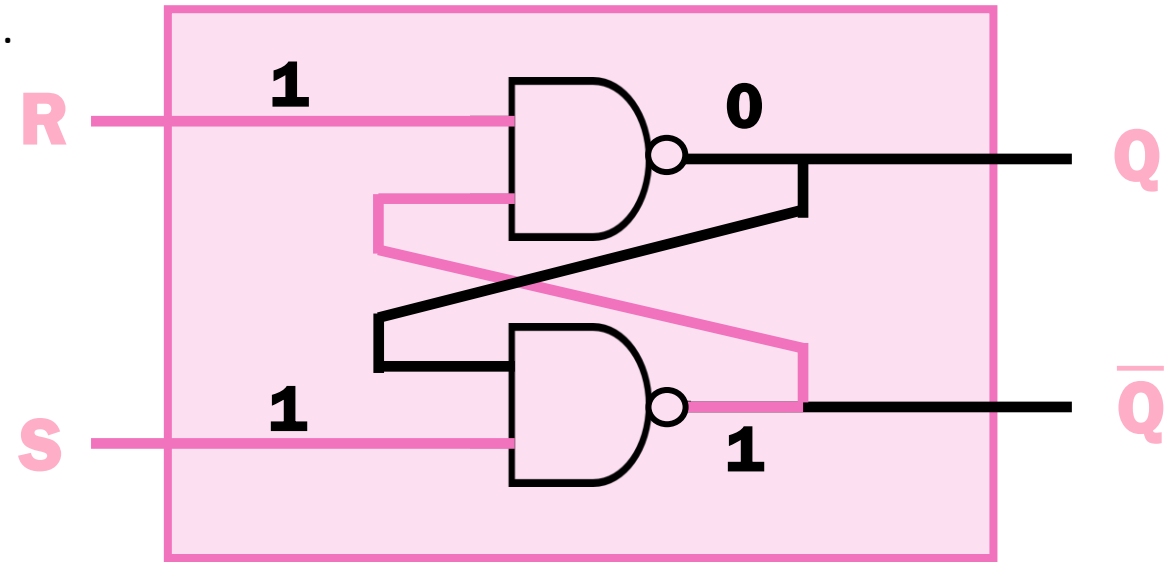
R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}



Latch RS

- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0. Q toma el valor de 0.
- Ahora, nuevamente si R=1 y S=1.

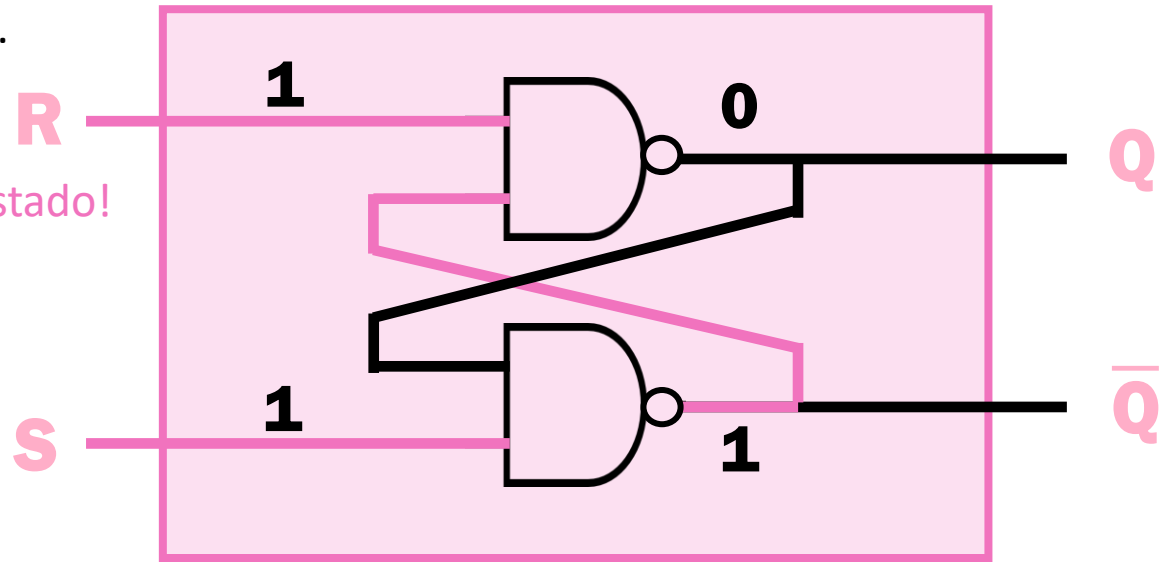
R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}



Latch RS

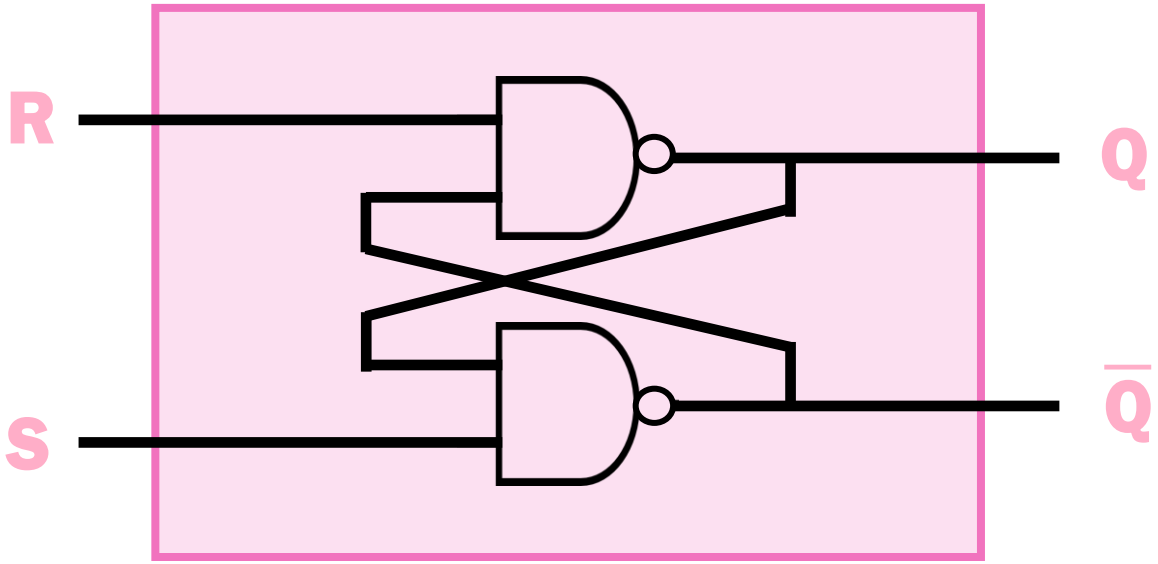
- Veamos paso a paso cómo funciona.
- Qué pasa si partimos con R=0 y S=1. Q toma el valor de 1.
- Ahora si R=1 y S=1. Se conserva el estado!
- Si R=1 y S=0. Q toma el valor de 0.
- Ahora, nuevamente si R=1 y S=1. De nuevo conserva el estado!

R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}



Latch RS

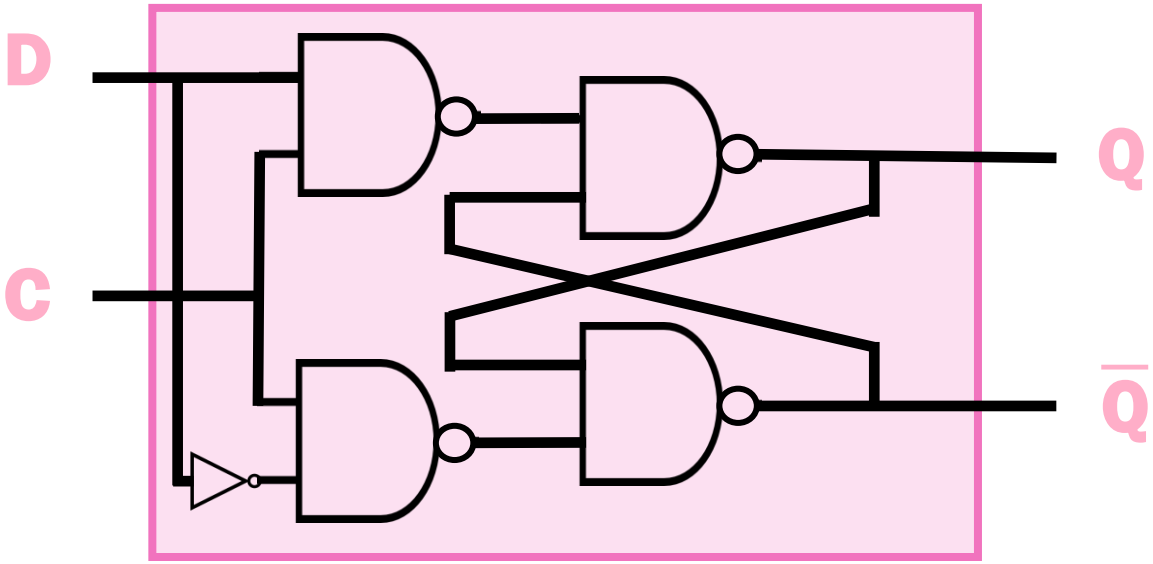
- Entonces este latch nos permite setear (Q=1) y resetear (Q=0) el valor de la memoria.
- En el caso de que R=S=1, Q se mantiene.
- Ahora si podemos guardar estados!! Tenemos memoria!



R	S	Q_t
0	0	-
0	1	1
1	0	0
1	1	Q_{t-1}

Latch D

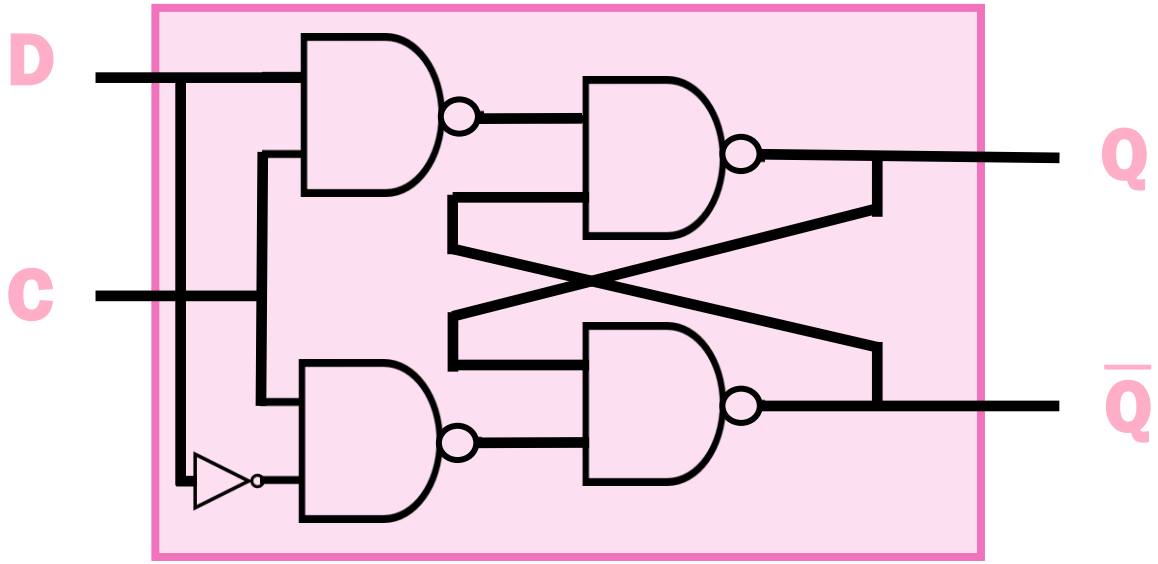
- Este latch se construye a partir de un latch RS, pero nos quita el estado invalido.
- C es una señal de control que habilita (o deshabilita) la carga de datos
- Ahora, cada vez que C=0, el estado anterior se conserva (independiente del valor de D).
- Cuando C=1, el valor que toma Q es el valor de D (Data).



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

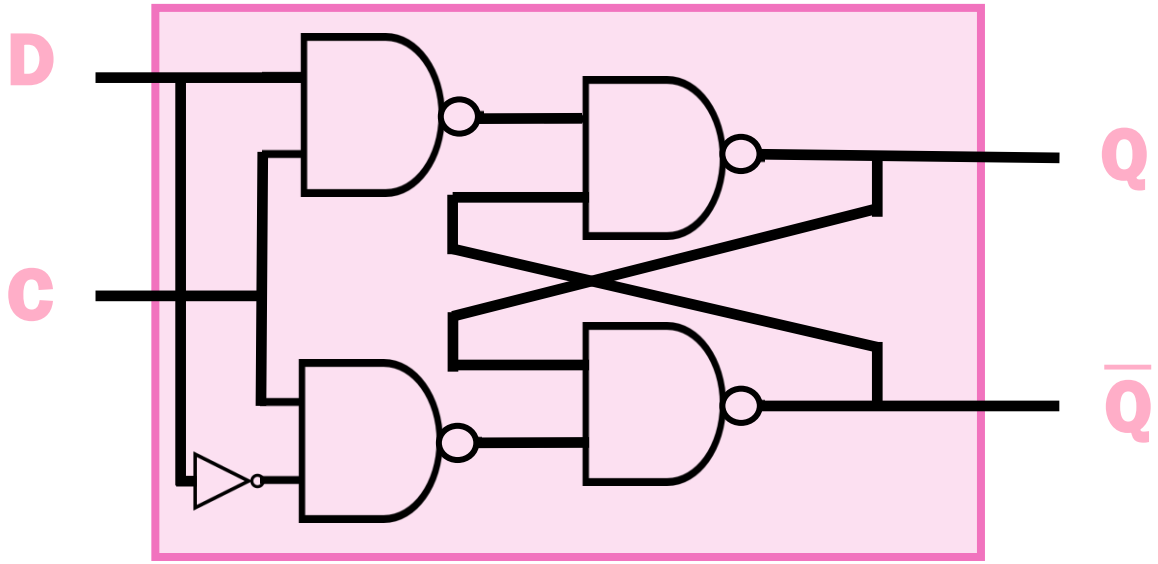
- Buenísimo… ahora tenemos la manera de almacenar un bit de información, podemos extender esto para guardar datos de más bits.



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

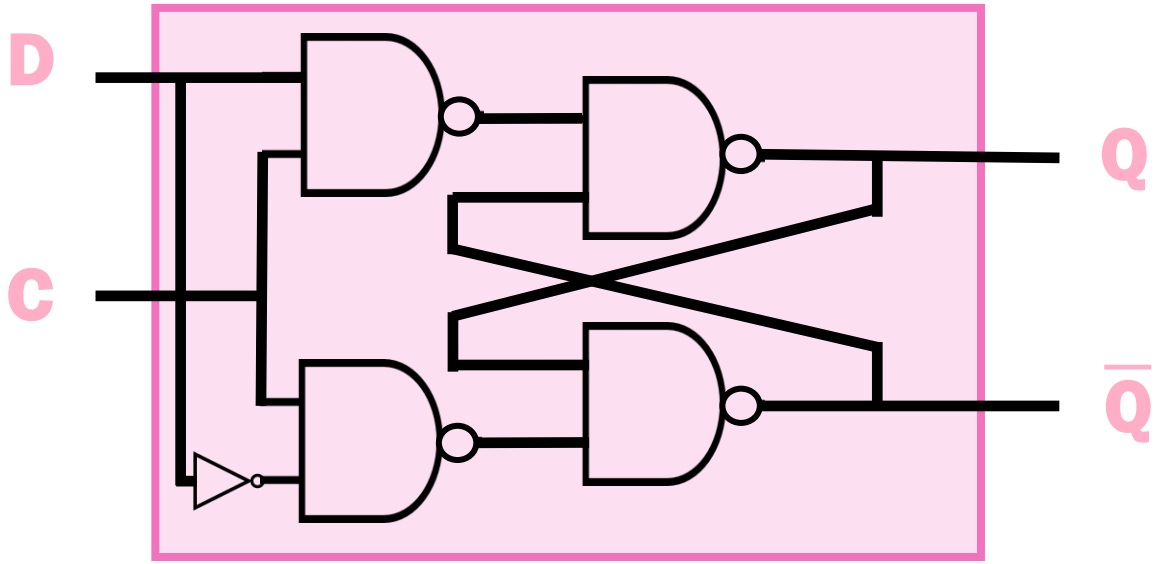
- Buenísimo… ahora tenemos la manera de almacenar un bit de información, podemos extender esto para guardar datos de más bits. **Waittt!!!**



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

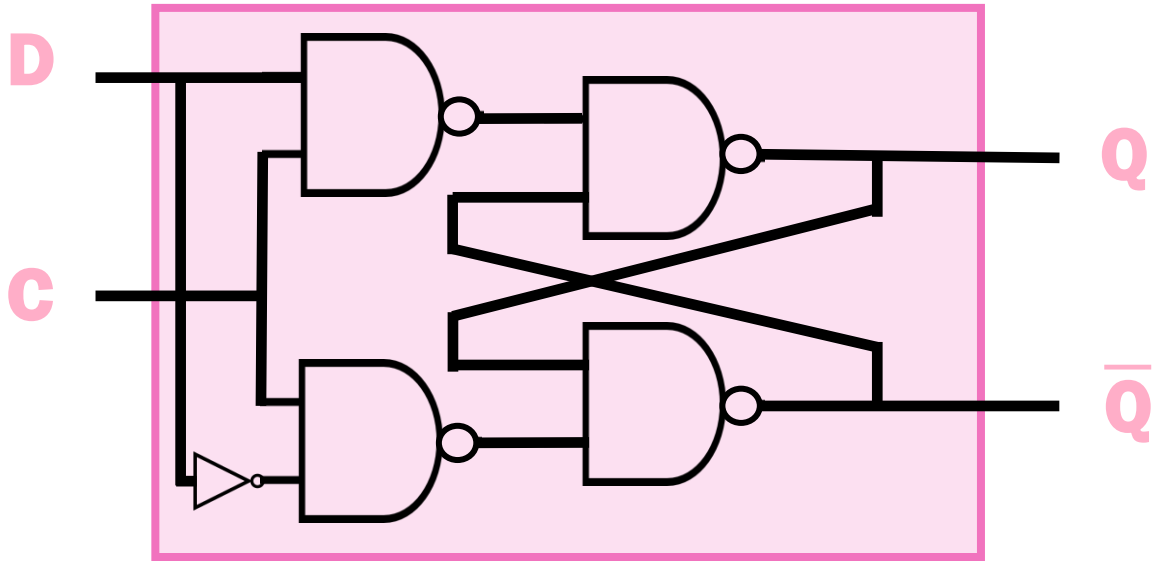
- Buenísimo… ahora tenemos la manera de almacenar un bit de información, podemos extender esto para guardar datos de más bits. **Waittt!!!**
- Aunque si sirve para guardar datos, tiene problemas.



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

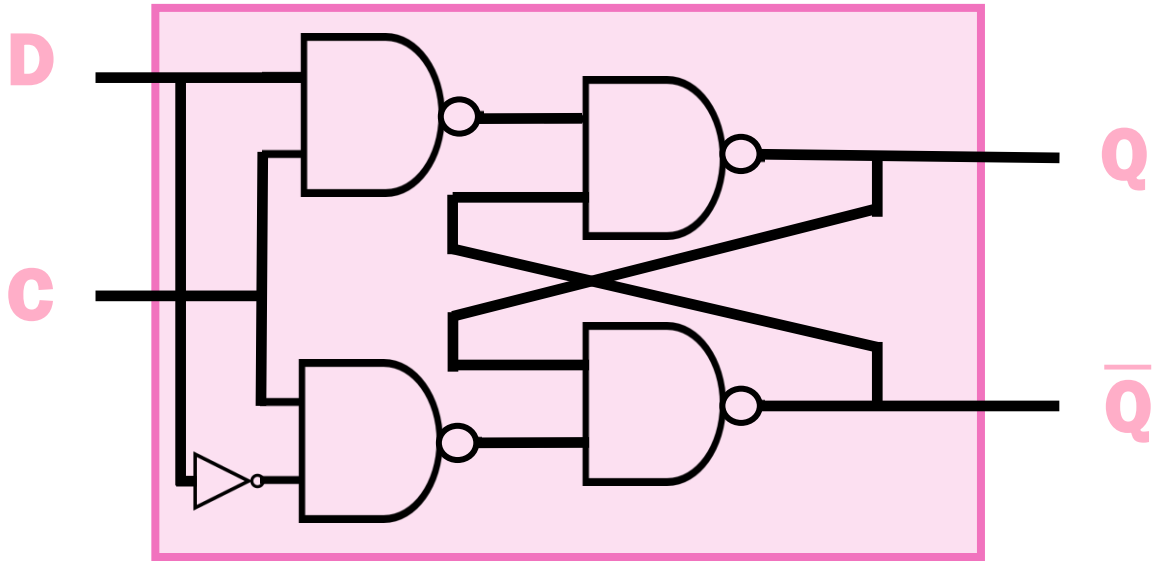
- Aunque si sirve para guardar datos, tiene problemas.
- Para que los datos se actualicen, la señal C debe ser igual a 1. Pero por qué esto es un problema?



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

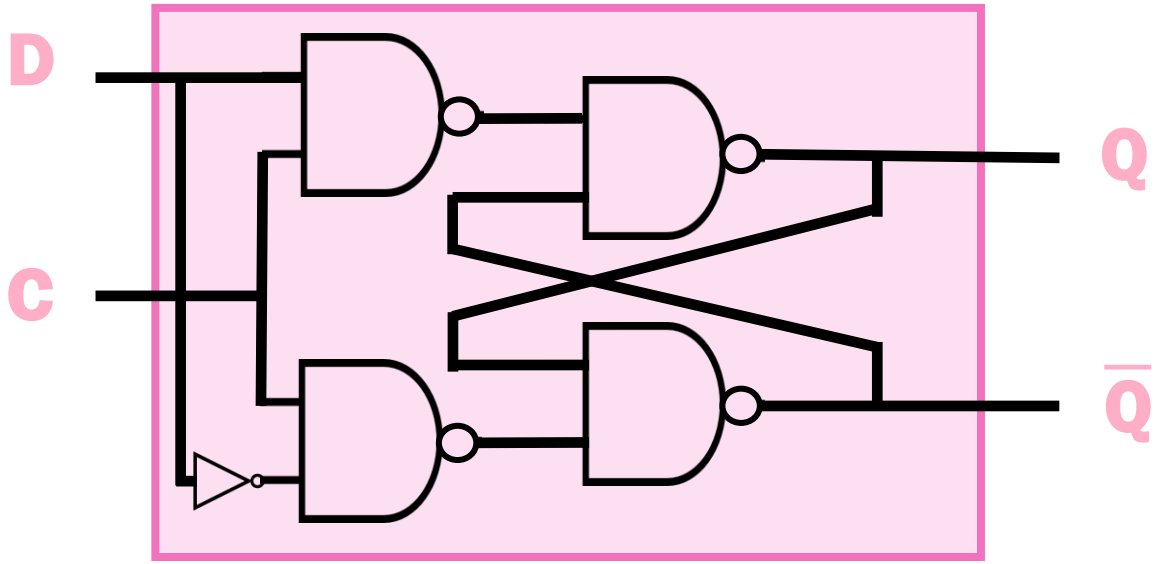
- Aunque si sirve para guardar datos, tiene problemas.
- Para que los datos se actualicen, la señal C debe ser igual a 1. Pero por qué esto es un problema?
- Qué pasa si el latch está conectado al circuito sumador (la salida del sumador es D y Q está conectado a la entrada A del sumador) y se desea guardar el valor de la salida. Se enciende C, guarda el valor, cambia el valor de A del sumador, suma, guarda, cambia A, suma, guarda, cambia A….. Todo esto en un milisegundo.



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

Latch D

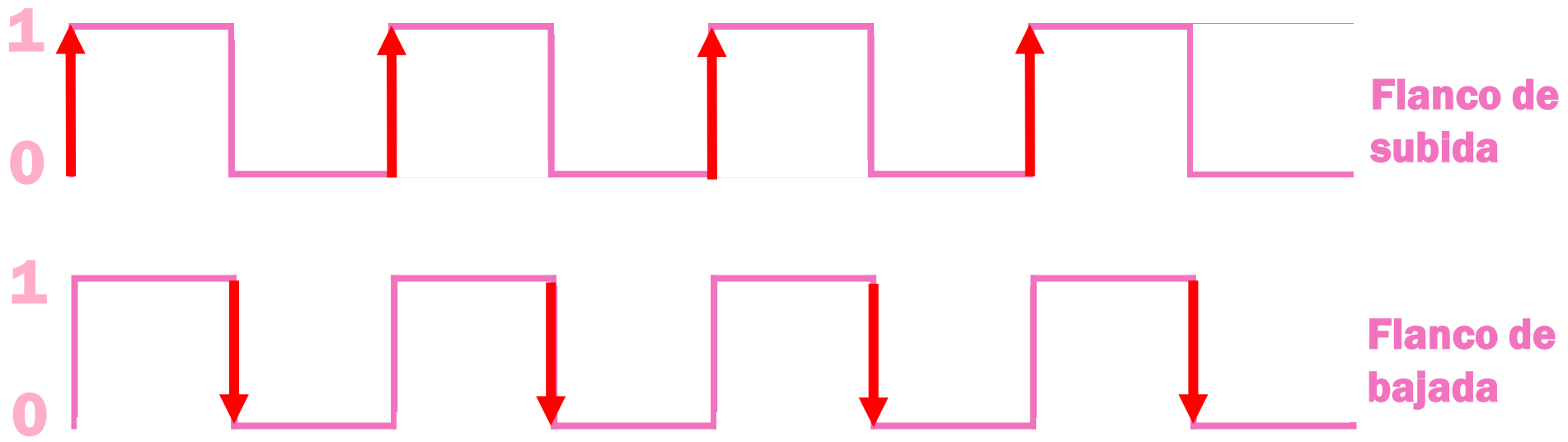
- No nos interesa guardar el valor cada vez que cambie D, necesitamos actualizar el valor solo una vez al encender el C.
- Entonces veremos como implementar esta memoria, pero que guarde cuando C este encendido, si no que cuando cambie de 0 a 1 (solamente en ese instante).



C	D	Q_t
0	0	Q_{t-1}
0	1	Q_{t-1}
1	0	0
1	1	1

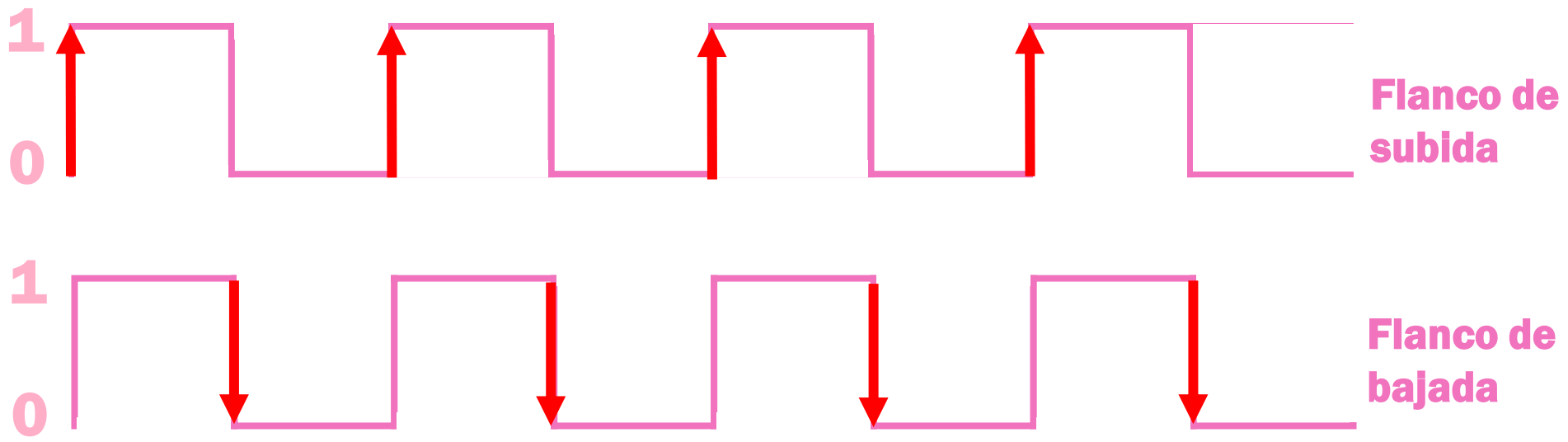
Clock

- El ***clock*** corresponde a una señal que oscila entre 0 a 1 en tiempo constante.
- Nos permitirá sincronizar los circuitos para que todos se encuentren en el estado esperado, en el mismo momento.
- Cada vez que el clock cambie de 0 a 1, se le llamará **flanco de subida** (el que utilizaremos).
- Cada vez que el clock cambie de 1 a 0, se le llamará **flanco de bajada**.



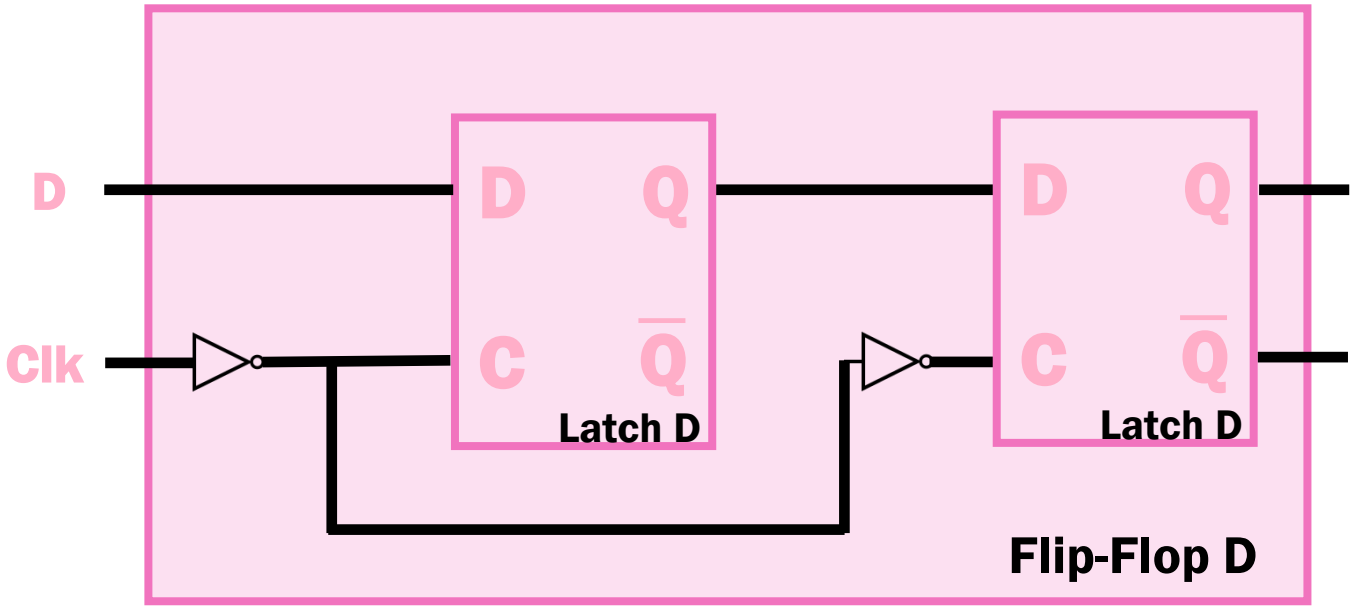
Clock

- Debemos crear una componente que solo se actualice al haber un flanco de subida.
- Esto permitirá que solamente se cambien los valores en un instante especifico, luego tienen un periodo de clock para realizar todas las operaciones que se deban hacer y estabilizar los valores antes del próximo flanco de subida (actualización de datos).



Flip-Flop D

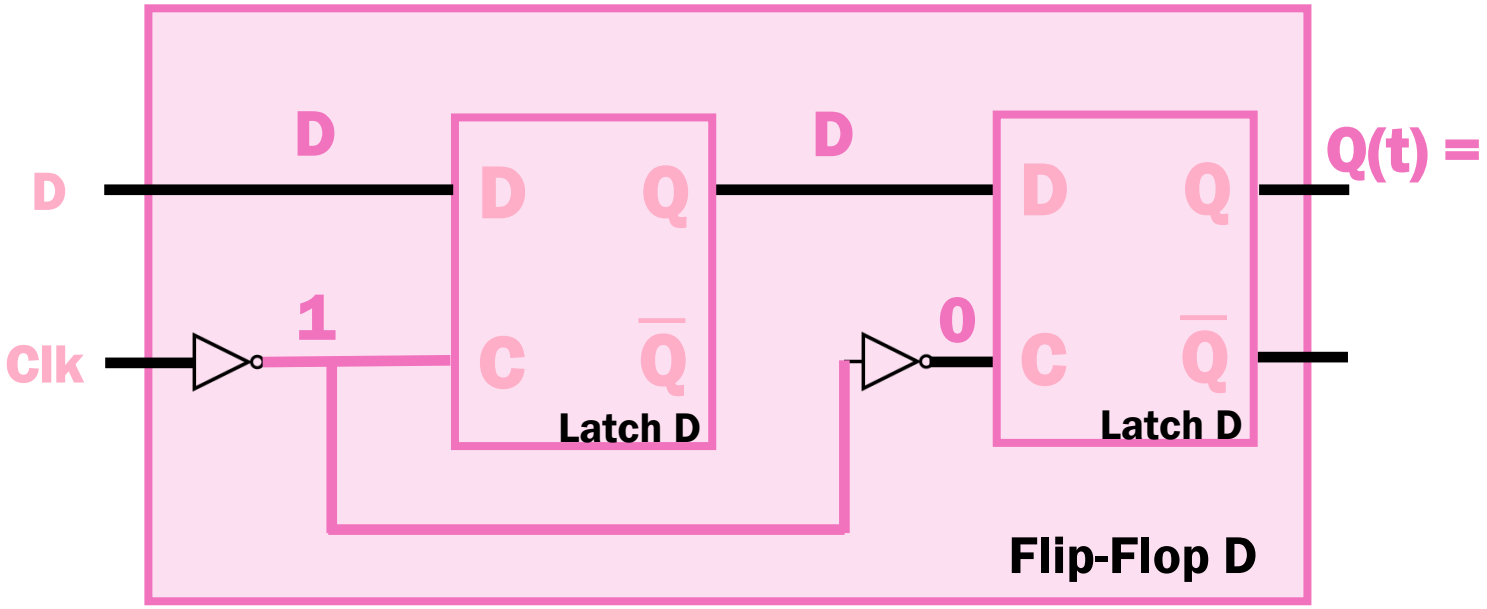
- Los ***flip-flops*** a diferencia de los latches, actualizan su valor solamente en flanco de subida de su señal de control (generalmente un clock).
- Ahora veremos como logra almacenar los datos con el flanco de subida de un clock.



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

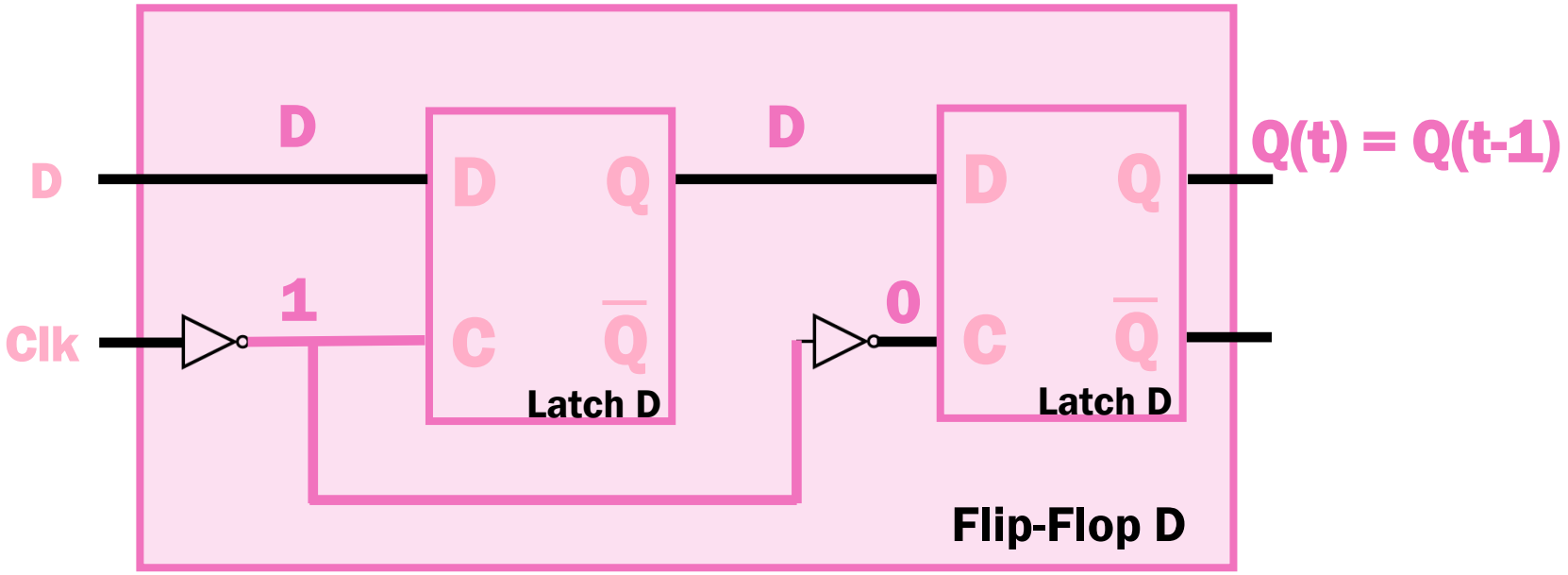
- Cuando Clk = 0.



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

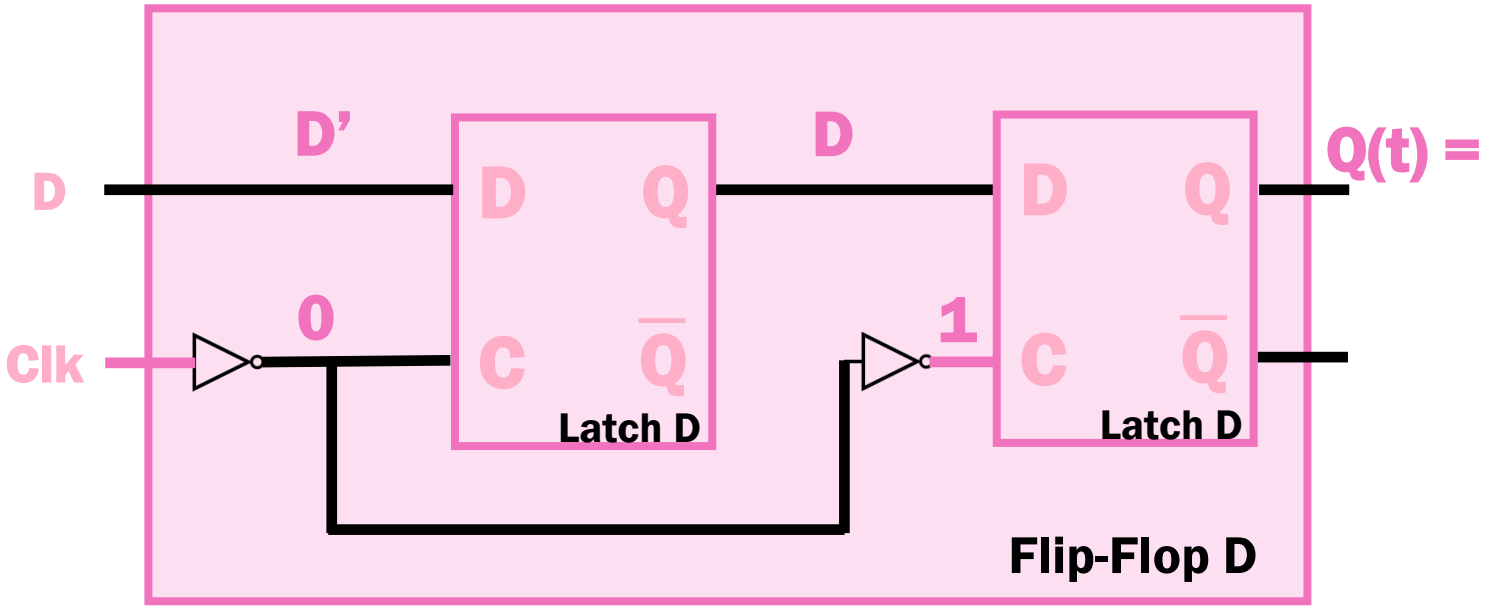
- Cuando Clk = 0. Se guarda el valor de D en el primer latch, pero no en el segundo.
- El valor final corresponderá al valor almacenado en el segundo latch, es decir que el estado previo.



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

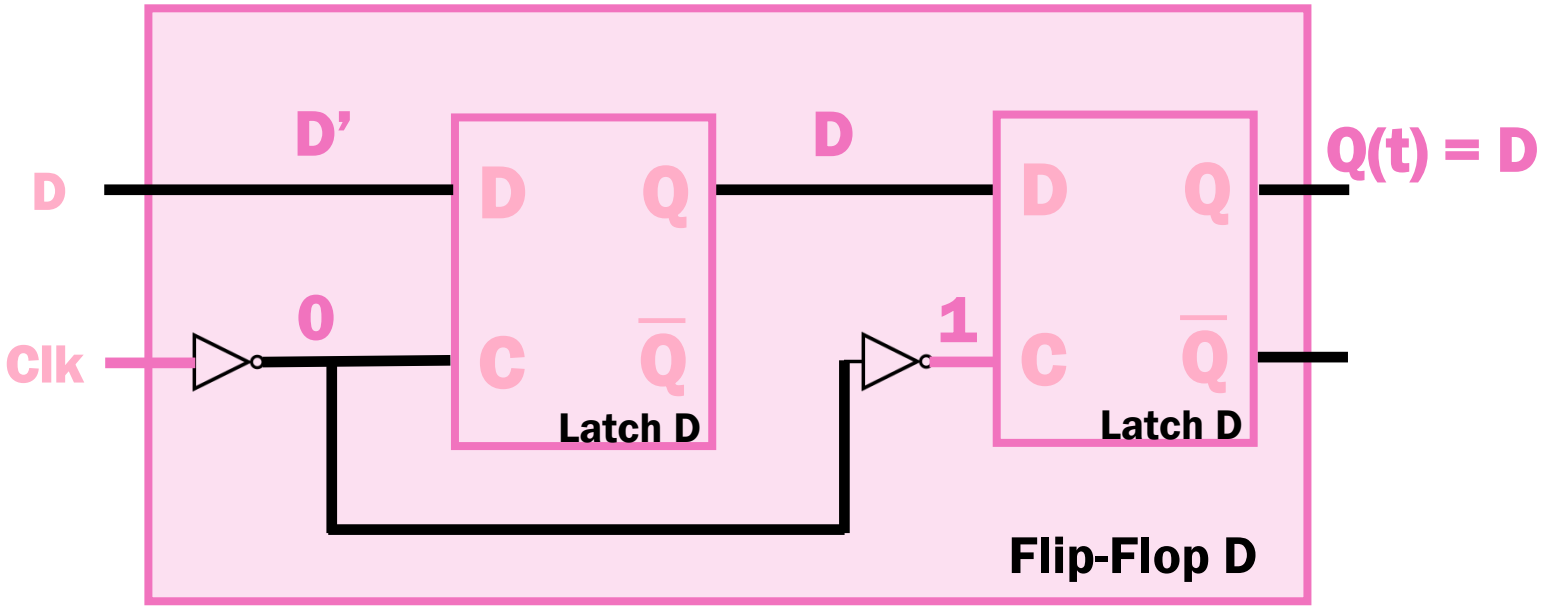
- Cuando Clk = 0 pasa a Clk=1 (Flanco de subida).



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

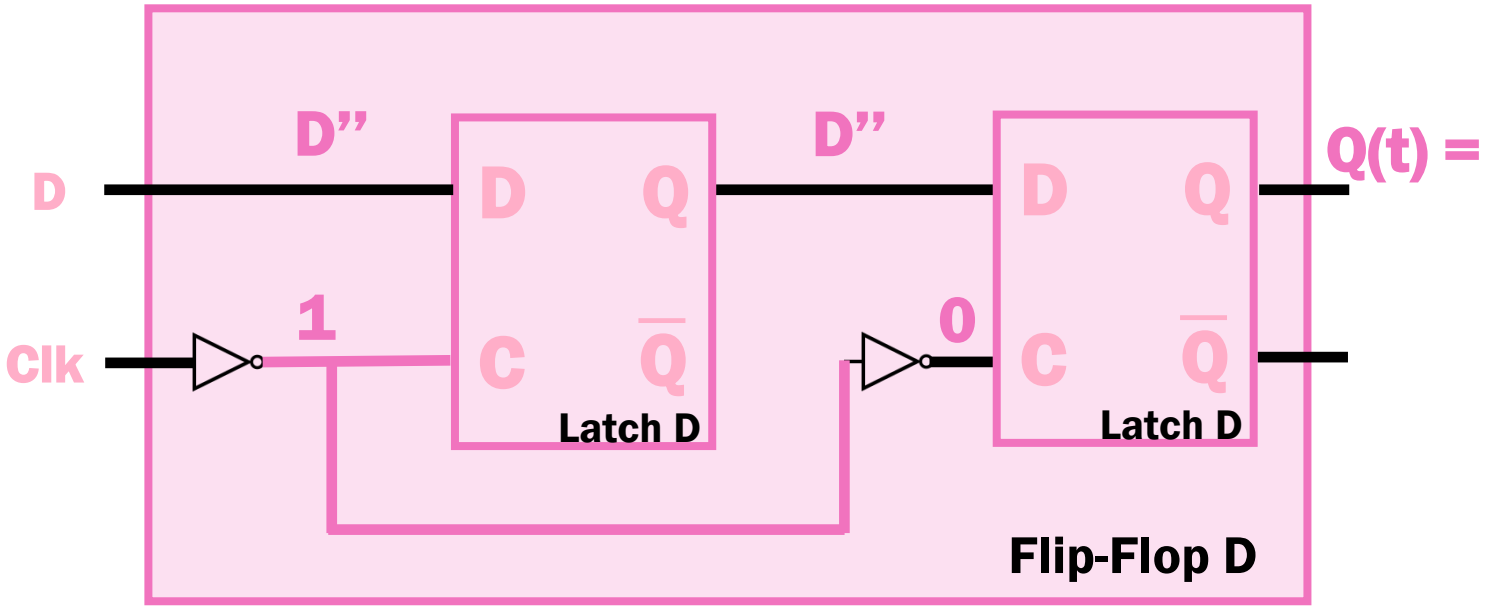
- Cuando Clk = 0 pasa a Clk=1 (Flanco de subida). Se deshabilita el almacenamiento del primer latch, por lo que el nuevo valor D’ no es guardado. El valor que contiene el latch de la izquierda es el valor D que tenía previamente.
- Ahora se habilita la carga del segundo latch, el valor que va a almacenar es D, el valor que estaba guardado el primer latch. El valor final corresponde a D.



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

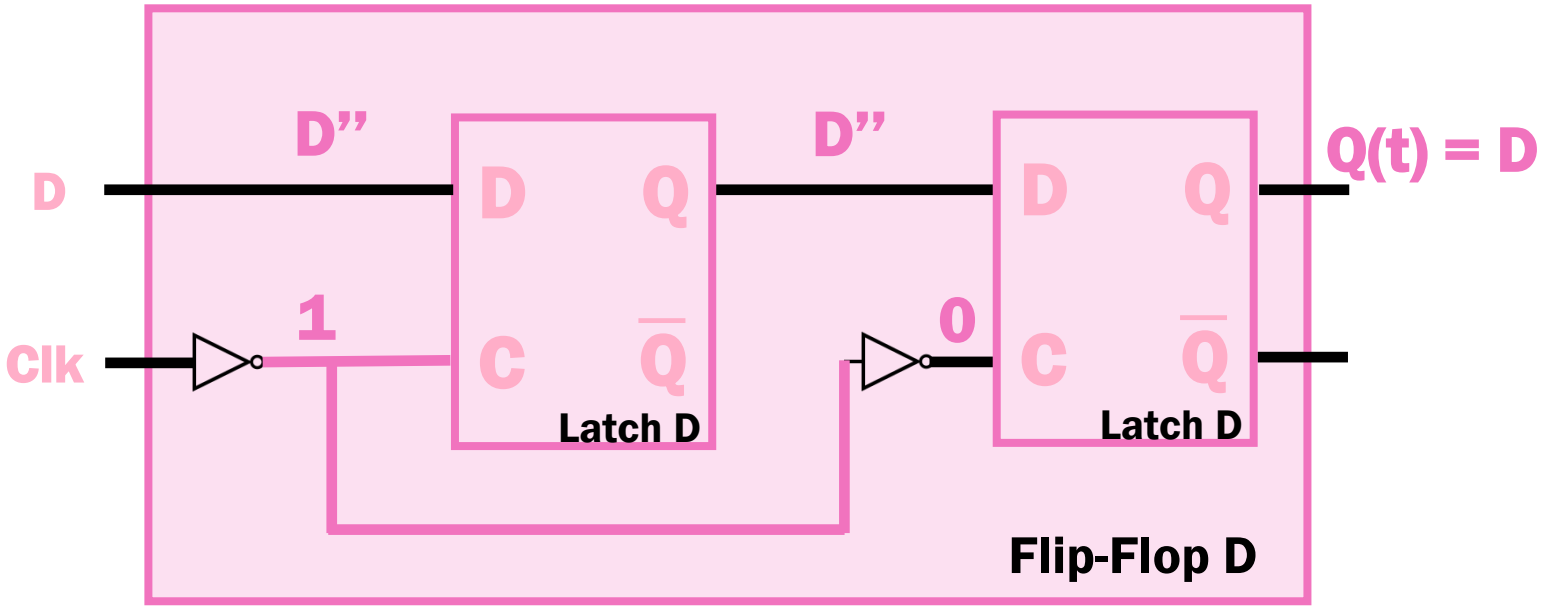
- Cuando Clk = 1 pasa a Clk=0 (Flanco de bajada).



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

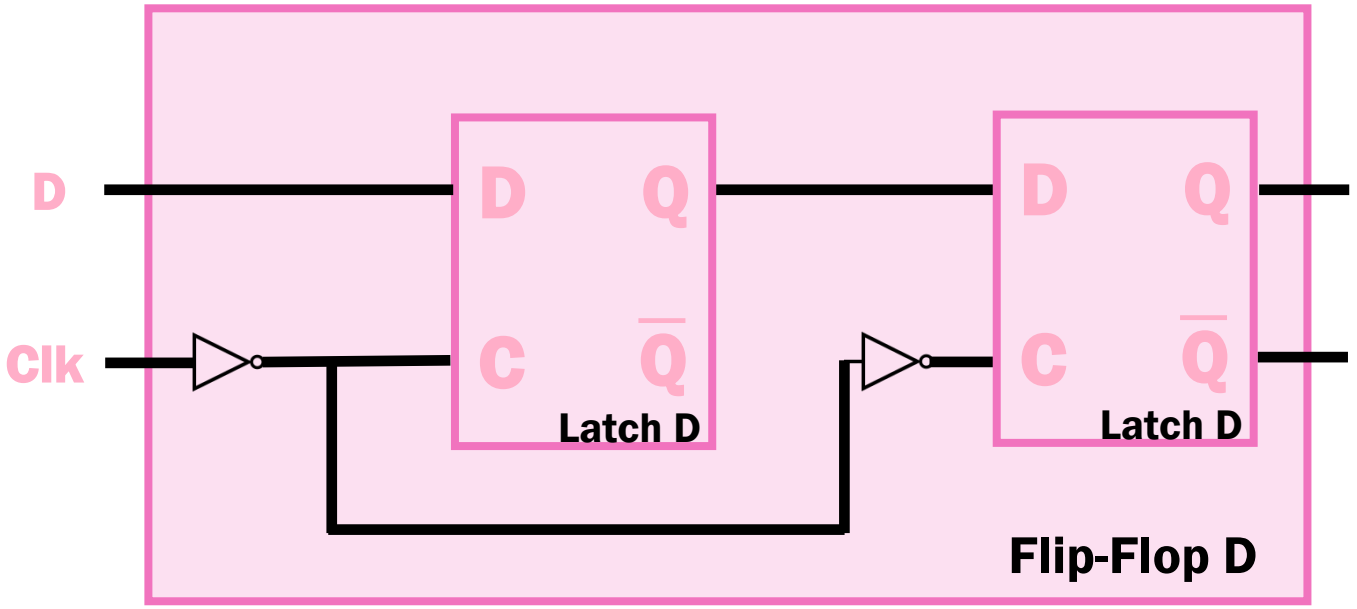
- Cuando Clk = 1 pasa a Clk=0 (Flanco de bajada). Se guarda D'' en el primer latch, pero no afecta al segundo latch. El valor final del flip-flop corresponderá al estado anterior.



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

- Ahora con el flip-flop D podemos almacenar un bit de información. Cómo podemos almacenar más de un solo bit?

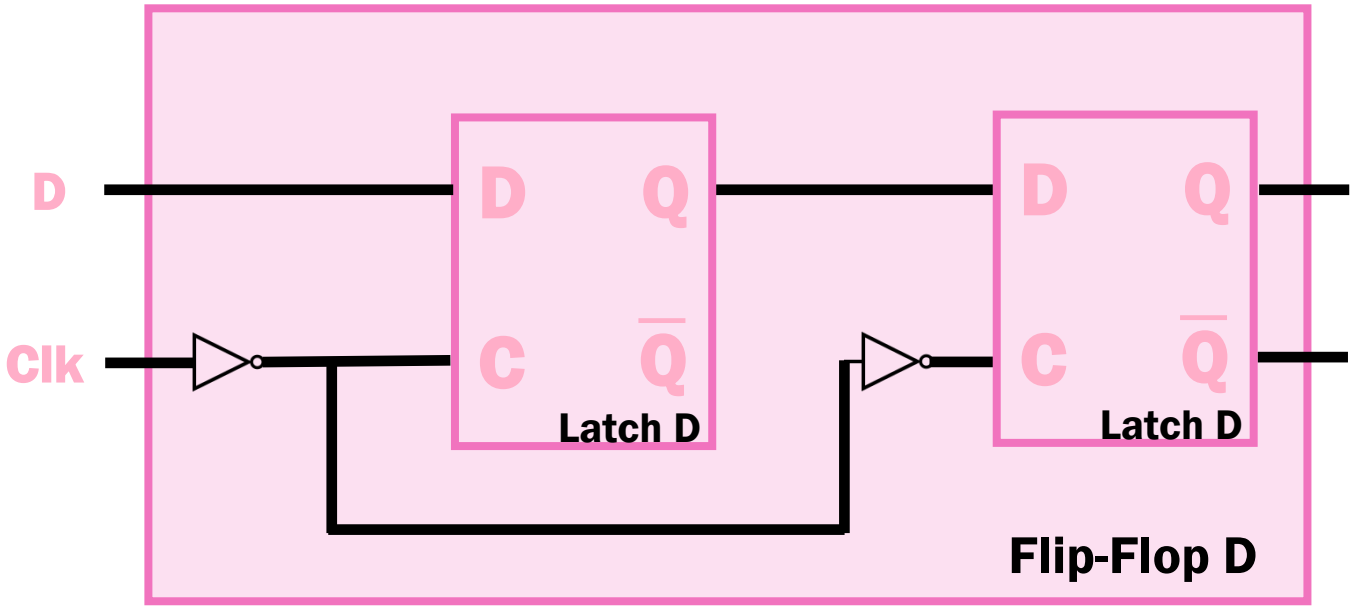


Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

Flip-Flop D

- Ahora con el flip-flop D podemos almacenar un bit de información. Cómo podemos almacenar más de un solo bit?

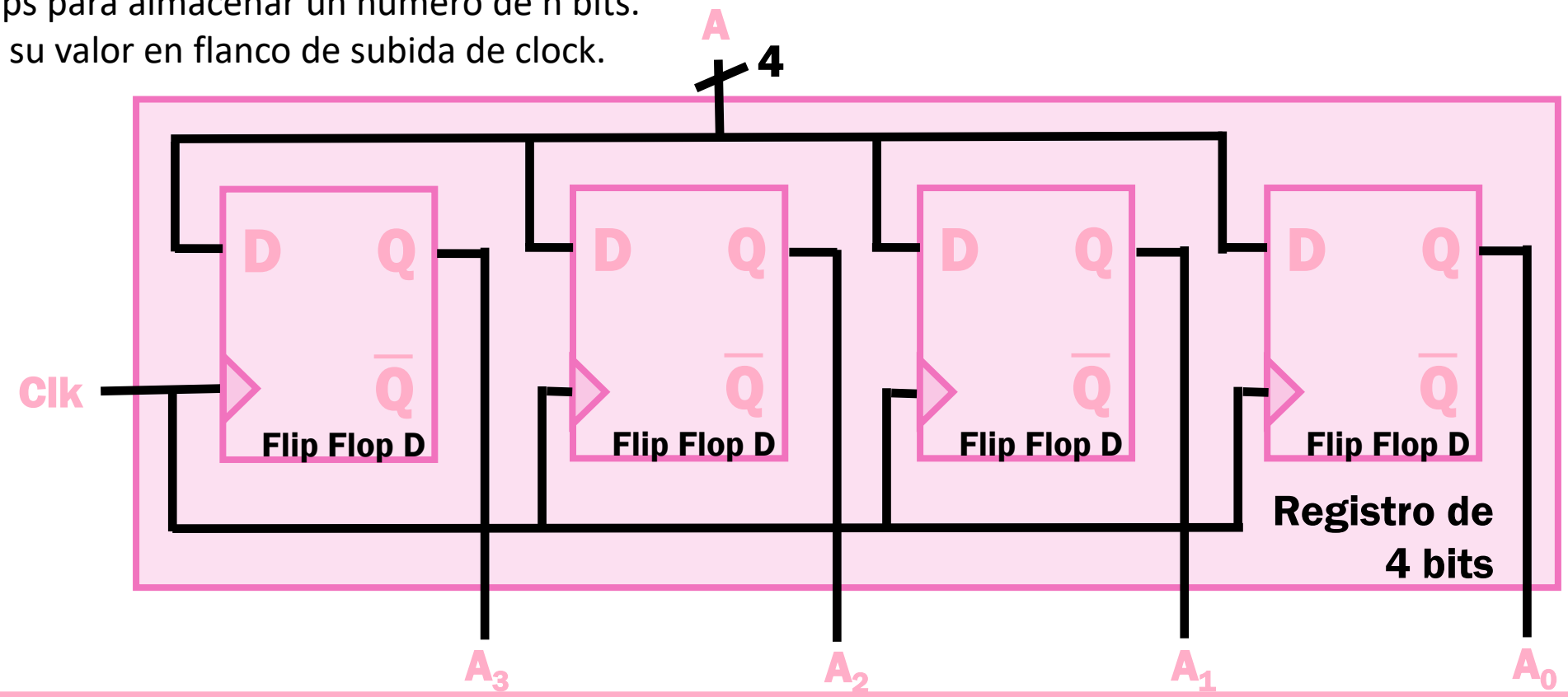
Solamente necesitamos usar más flip-flops!



Clk	D	Q_t
0/1/↓	X	Q_{t-1}
↑	0	0
↑	1	1

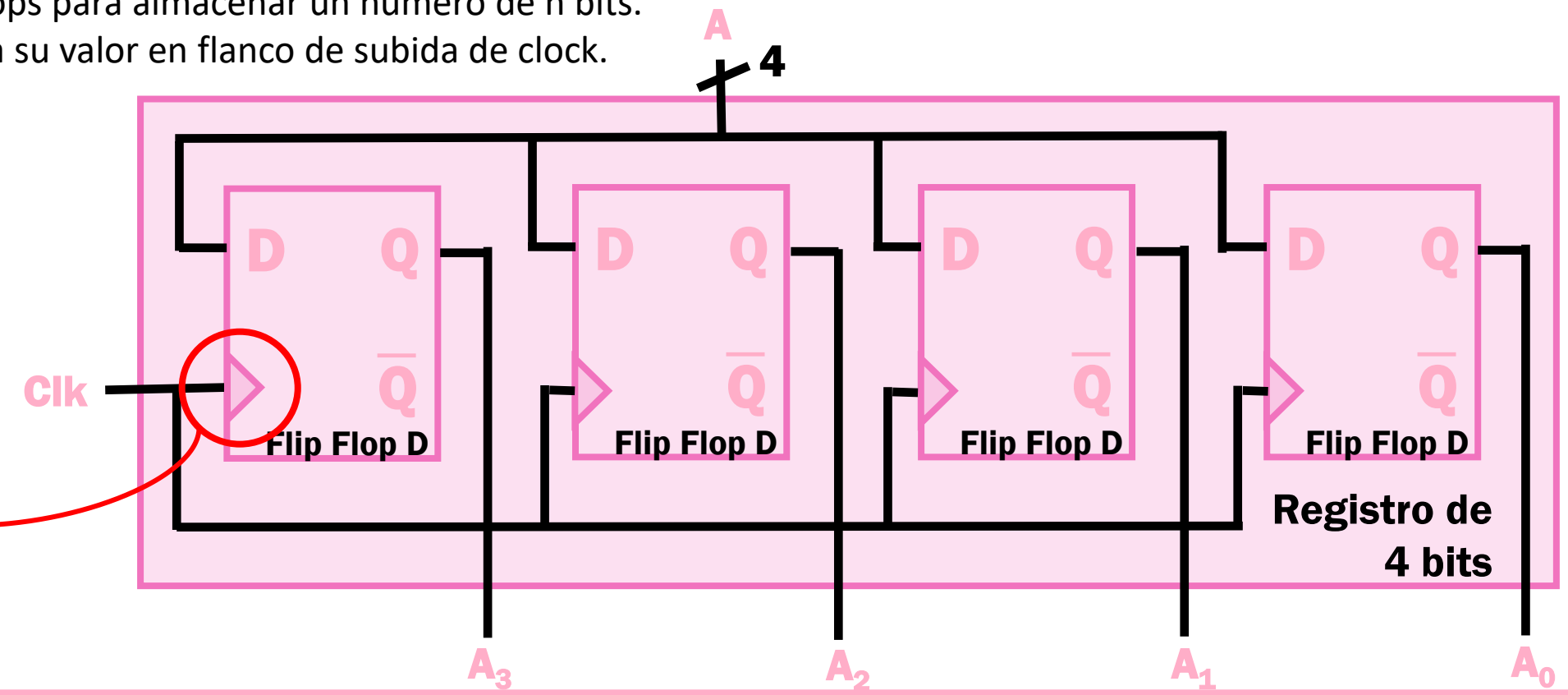
Registros

- El registro va a ser la componente que nos permita guardar números de n bits.
- Necesitamos n flip-flops para almacenar un número de n bits.
- Este registro actualiza su valor en flanco de subida de clock.



Registros

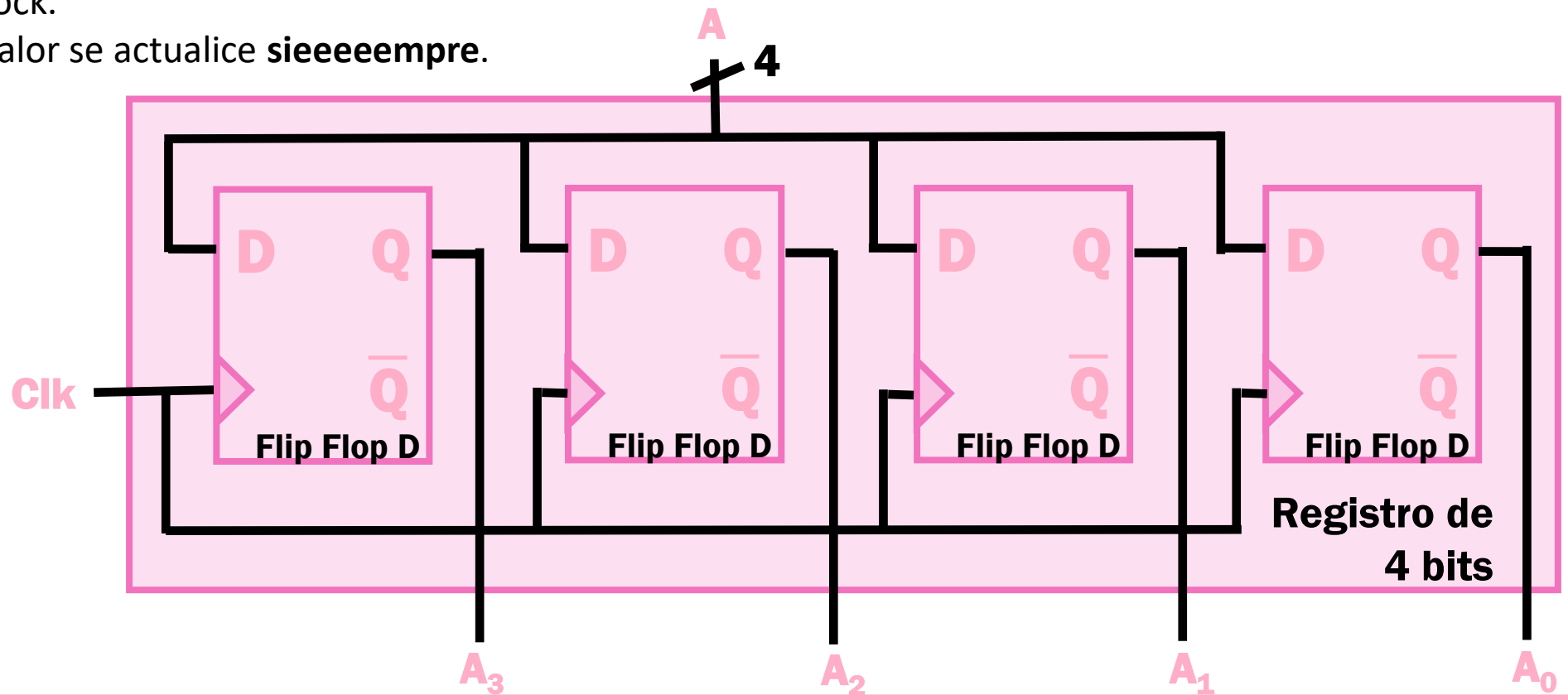
- El registro va a ser la componente que nos permita guardar números de n bits.
- Necesitamos n flip-flops para almacenar un número de n bits.
- Este registro actualiza su valor en flanco de subida de clock.



Cuando una componente tenga un input con símbolo un triangulo, significa que ese input es un clock.

Registros

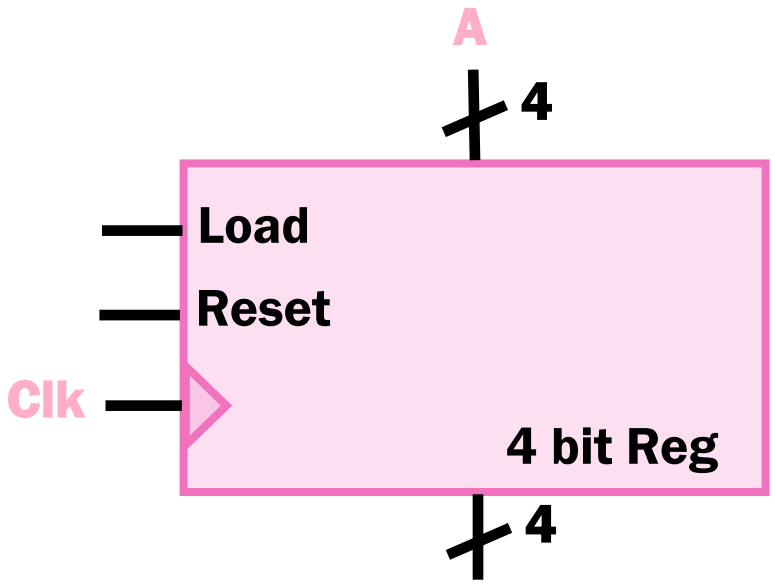
- Aunque cada vez estamos más cerca, este registro no es perfecto.. funciona.. pero se actualiza siempre que haya un flanco de subida de clock.
- No queremos que el valor se actualice **sieeeeempre**.



Registros

- Vamos a agregar dos señales de control:
  - Si **Load=1** y hay un **flanco de subida** de clock. Entonces el registro guarda el valor A. Para cualquier otro caso, el registro mantiene el valor anterior.
  - Si **Reset=1**. En algunas implementaciones también debe haber flanco de subida de clock. El valor del registro vuelve a su valor por defecto, 0.

Clk	Load	Reset	A'
0/1/↓	X	0	A'_{t-1}
↑	0	0	A'_{t-1}
↑	1	0	A
X	X	1	0

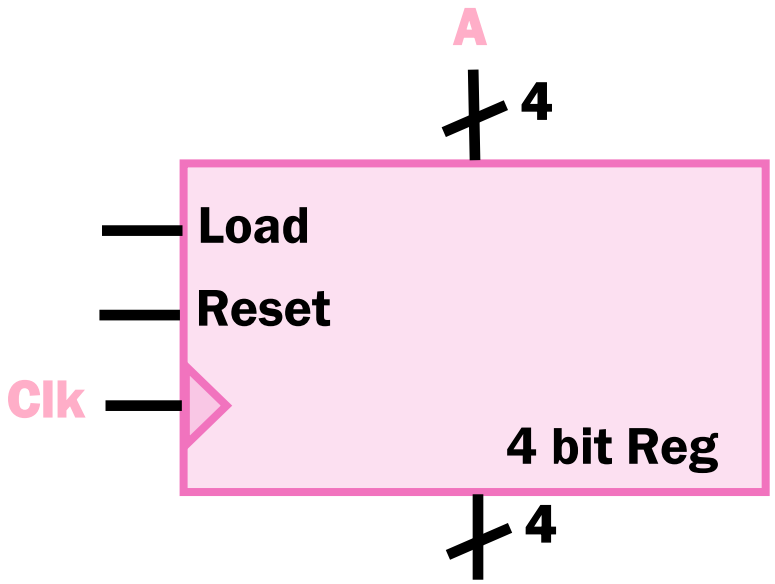


Registros

- Ahora si con este registro vamos a poder:
  - Almacenar un valor de 4 bits y que perdure en el tiempo.
  - Habilitar la carga del valor, ya no varia con cada ciclo de clock (Load=1).
  - Eliminar el contenido almacenado.

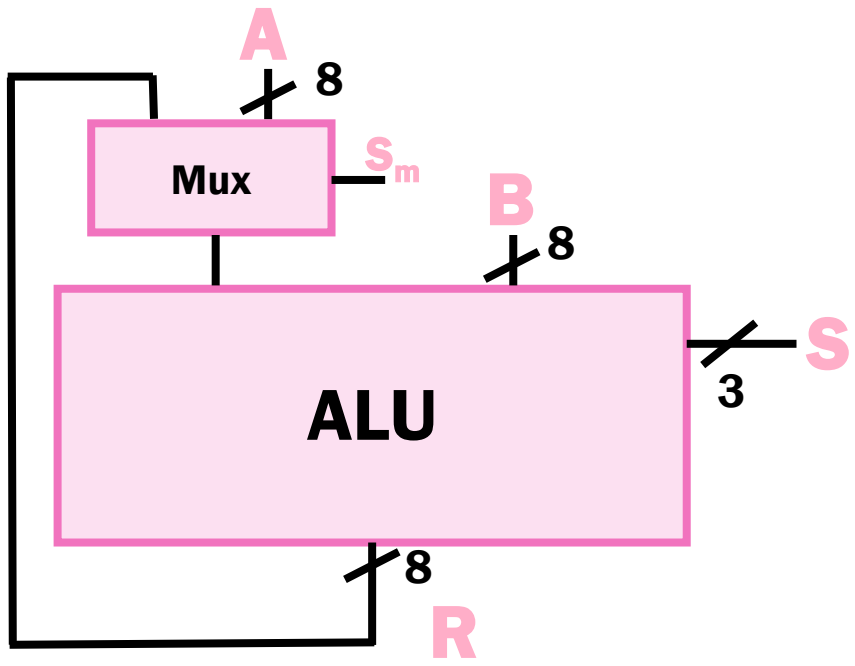
- Volvamos a la ALU.

Clk	Load	Reset	A’
0/1/↓	X	0	A'_{t-1}
↑	0	0	A'_{t-1}
↑	1	0	A
X	X	1	0



ALU

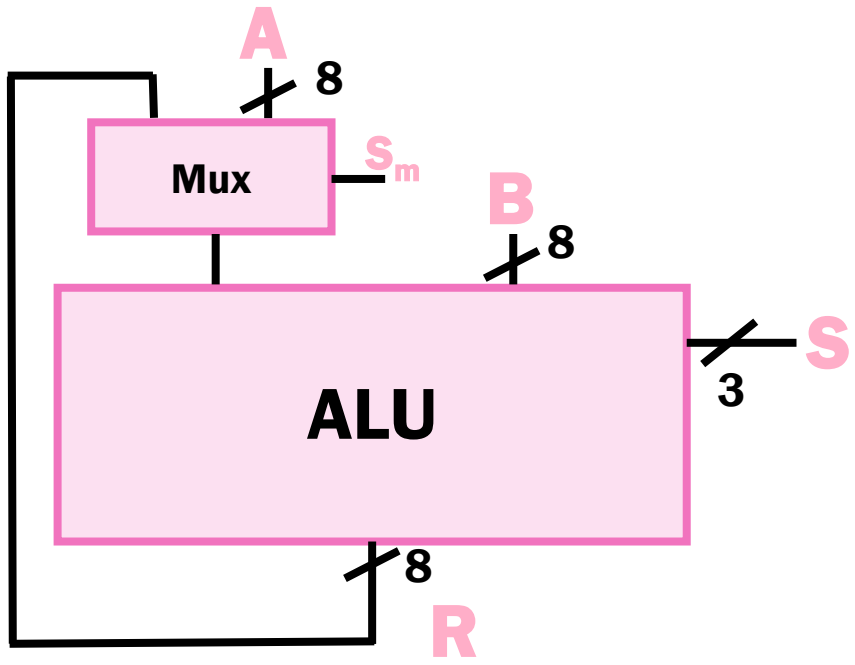
- Quedamos en que si queríamos tener una ALU, en donde se puede elegir el resultado anterior como operando podríamos pensar en construir dicho circuito de la siguiente forma.



ALU

- Quedamos en que si queríamos tener una ALU, en donde se puede elegir el resultado anterior como operando podríamos pensar en construir dicho circuito de la siguiente forma.

- Uno de los problemas que encontramos es que no hay forma de controlar que la operación se realice tan solo una vez.
- También vamos a tener el problema de que el valor no se esta guardando realmente en ninguna parte.

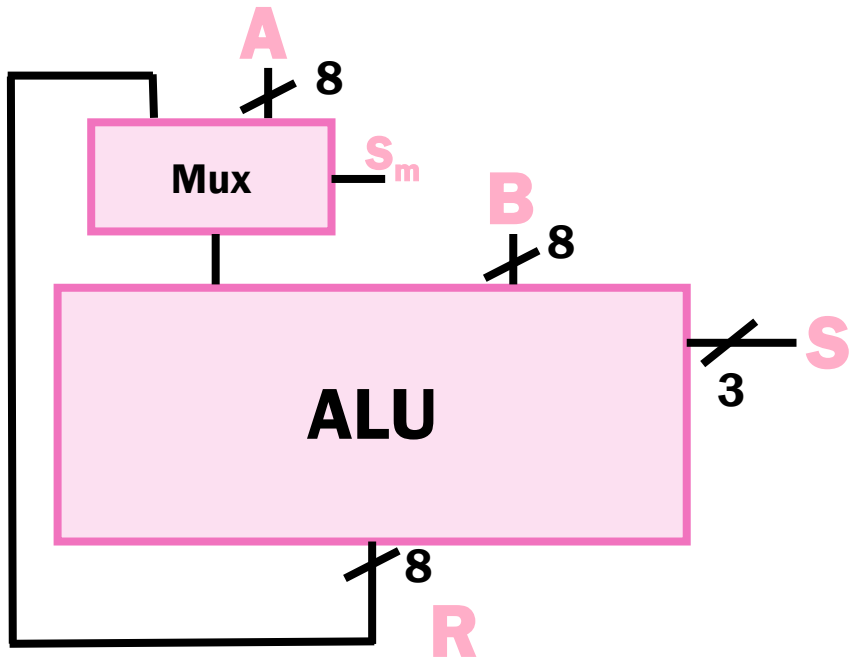


ALU

- Quedamos en que si queríamos tener una ALU, en donde se puede elegir el resultado anterior como operando podríamos pensar en construir dicho circuito de la siguiente forma.

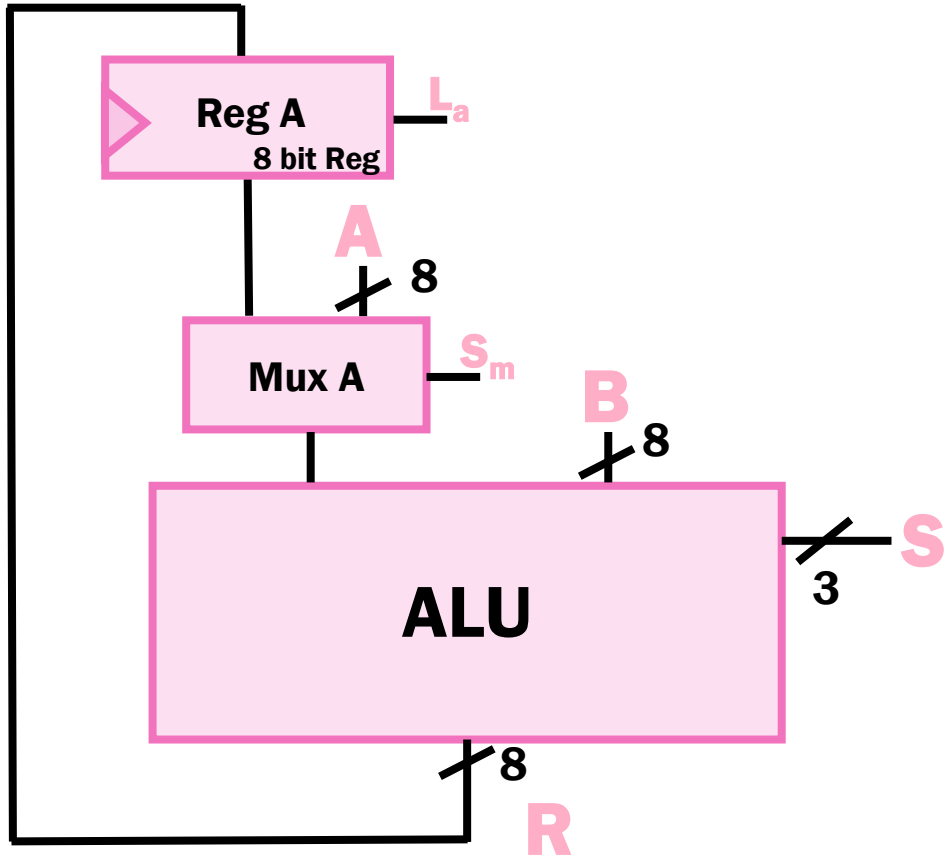
- Uno de los problemas que encontramos es que no hay forma de controlar que la operación se realice tan solo una vez.
- También vamos a tener el problema de que el valor no se esta guardando realmente en ninguna parte.

Ahora que tenemos registros podemos solucionar ambos problemas!



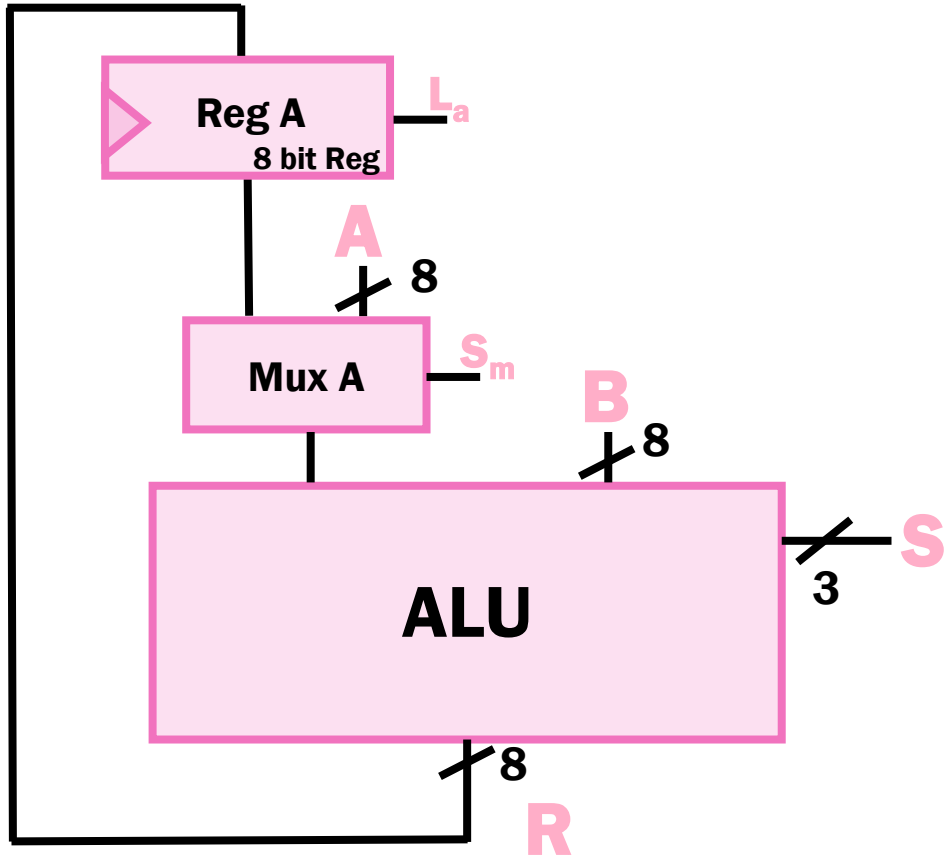
ALU

- Agregamos un registro A de 8 bits.



ALU

- Agregamos un registro A de 8 bits.
  - Su entrada va a ser el resultado de la ALU (**R**).
  - Su salida va conectada a un multiplexor (**Mux A**), que decide si entra a la ALU el valor almacenado en el registro (**Reg A**) o un input **A**.
  - Y va a tener una señal de control de carga $\mathbf{L_a}$. Cuando esta señal valga 1, al haber un flanco de subida de clock, el registro almacenará el valor que se encuentre en **R** en ese instante. Cuando esta señal es 0, el valor anterior se preserva.

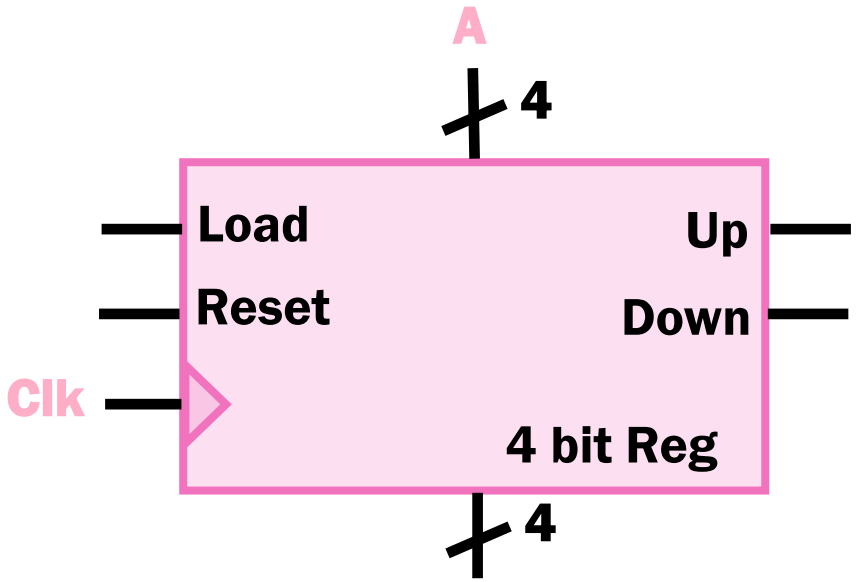


Otros tipos de memoria...

Contador

- Modificando el registro se puede convertir en un contador.
- Se agregan dos señales extra: Up y Down.
  - Up: indica si hay que incrementar en uno el valor almacenado.
  - Down: indica si hay que decrementar en uno el valor almacenado.

Ya sabemos que todas estas componentes funcionan en flanco de subida de clock, desde ahora lo vamos a “obviar”.

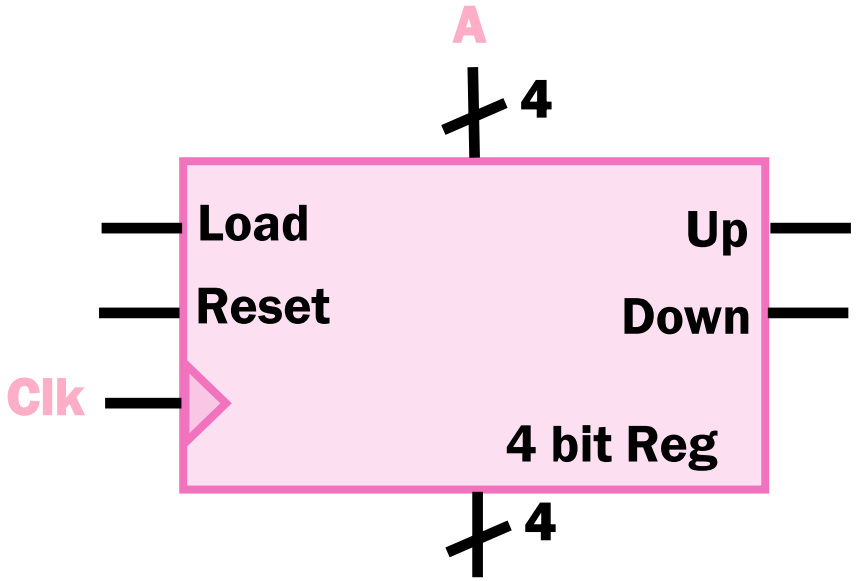


Contador

- Modificando el registro se puede convertir en un contador.
- Se agregan dos señales extra: Up y Down.
  - Up: indica si hay que incrementar en uno el valor almacenado.
  - Down: indica si hay que decrementar en uno el valor almacenado.

Ya sabemos que todas estas componentes funcionan en flanco de subida de clock, desde ahora lo vamos a “obviar”.

Esto les va a servir en su proyecto próximamente… 👀



DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 4 – Almacenamiento de Datos

Susana Figueroa