DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 3 – ALU

Susana Figueroa

Sumador-Restador

- Hace varias diapos atrás dijimos que queríamos tener un circuito que sea capaz de sumar y restar.
- Sabemos que nuestro circuito sumador lo puede lograr, pero necesitamos una forma de seleccionar el input B sea su complemento de 2 para la resta. Cómo podemos lograr esto?

Sumador-Restador

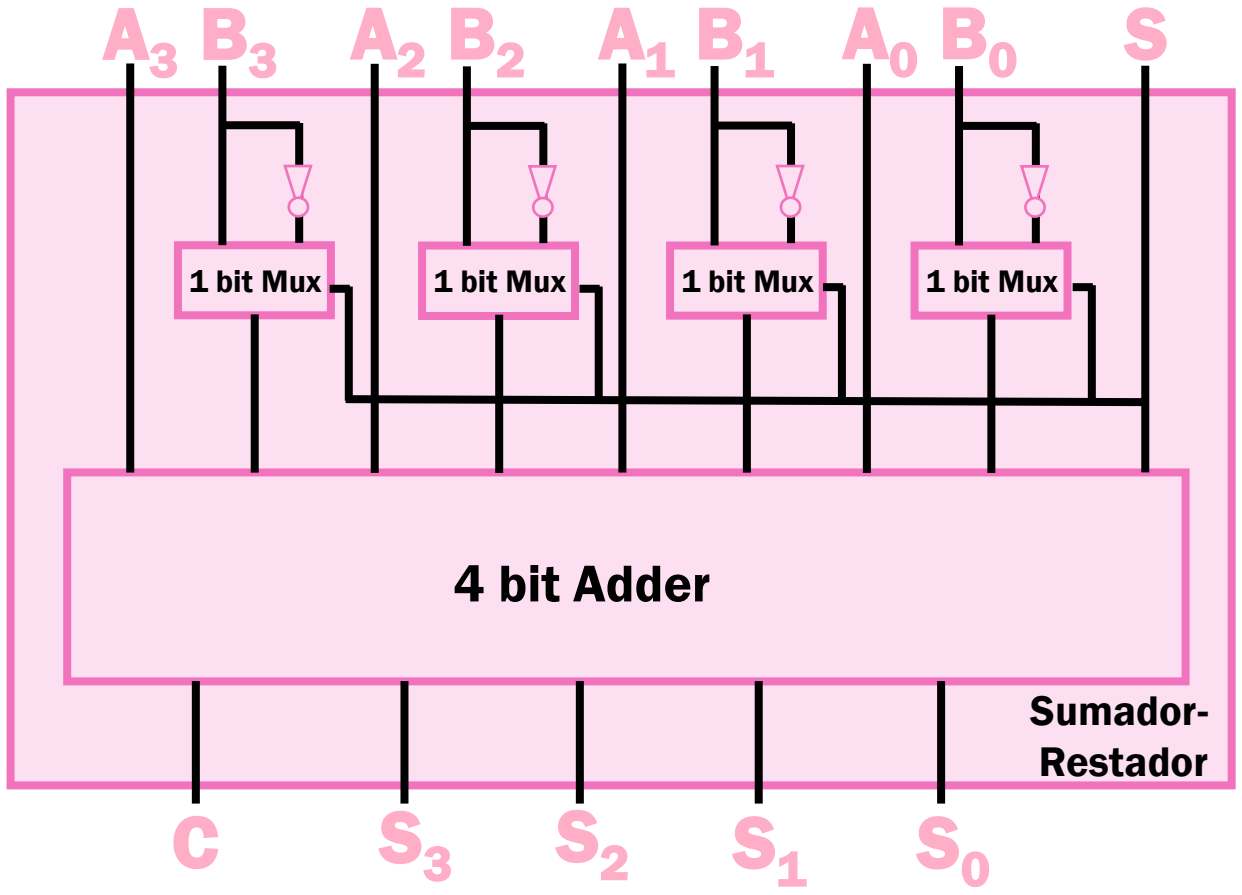
- Hace varias diapos atrás dijimos que queríamos tener un circuito que sea capaz de sumar y restar.
- Sabemos que nuestro circuito sumador lo puede lograr, pero necesitamos una forma de seleccionar el input B sea su complemento de 2 para la resta. Cómo podemos lograr esto? **Con multiplexores!**

Sumador-Restador

- Hace varias diapos atrás dijimos que queríamos tener un circuito que sea capaz de sumar y restar.
- Sabemos que nuestro circuito sumador lo puede lograr, pero necesitamos una forma de seleccionar el input B sea su complemento de 2 para la resta. Cómo podemos lograr esto? **Con multiplexores!**

Veamos cómo queda un sumador-restador de 4 bits…

Sumador-Restador



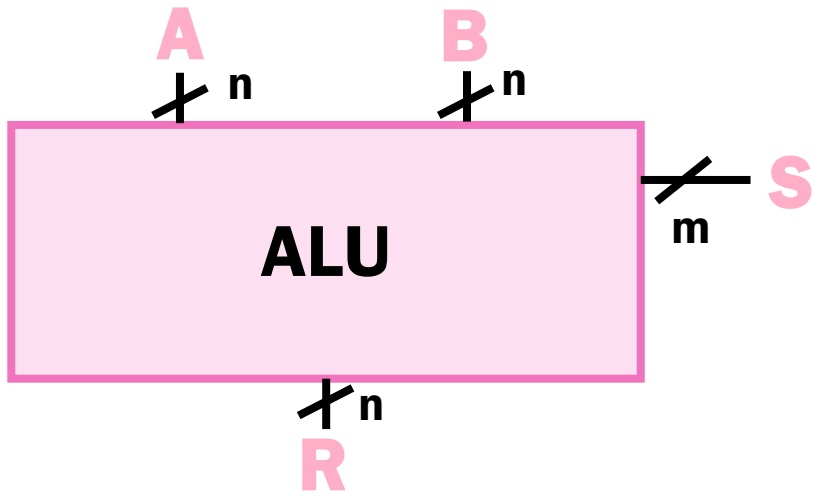
- Usaremos S para indicar si la operación que se quiere realizar es una suma o resta.
  - S = 0 indica suma
  - S = 1 indica resta

- 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.

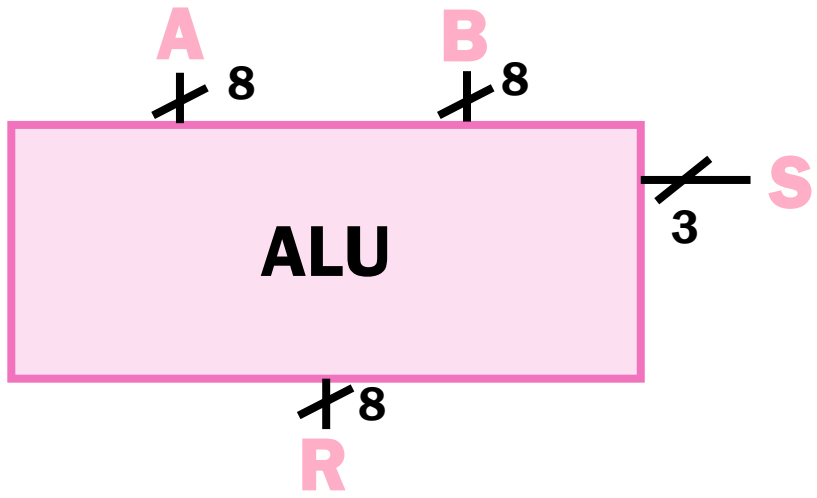


ALU

- 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.

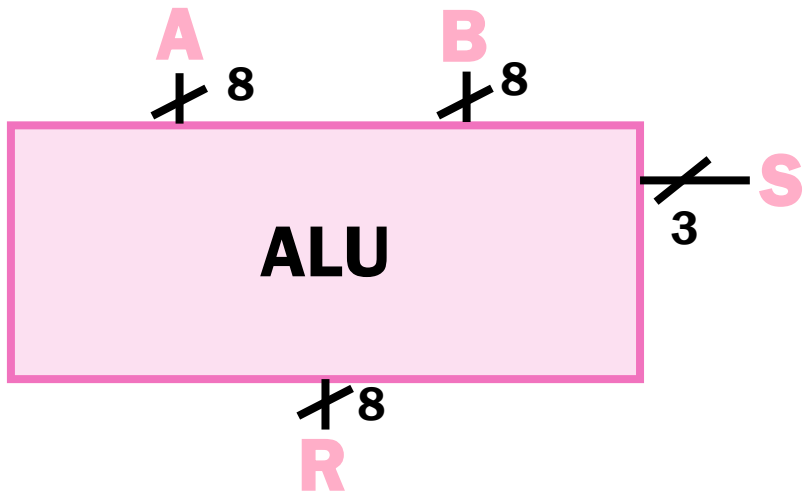
- Los operandos que utiliza: A y B, van a ser de n bits cada uno.
- El resultado R de la operación es también de n bits.
- Tiene una señal de control S de m bits que permite seleccionar entre máximo 2^m operaciones.

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

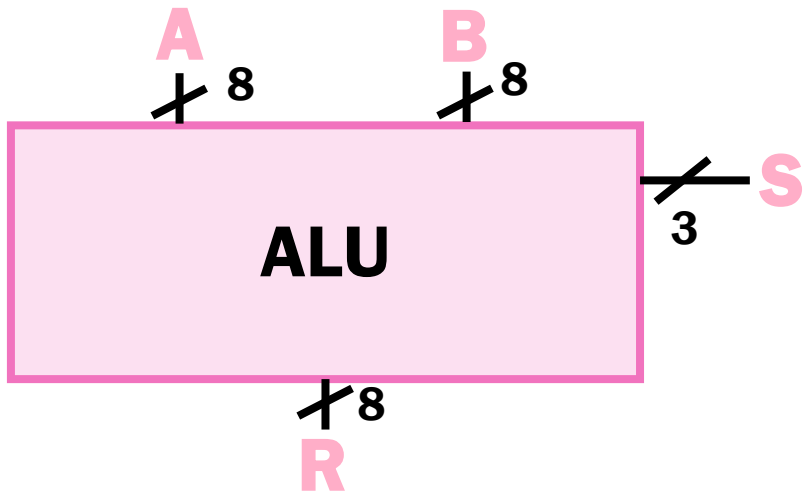
- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?



ALU

- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?

- Vamos a tener:
  - A + B
  - A - B
  - A AND B
  - A OR B
  - NOT A
  - A XOR B

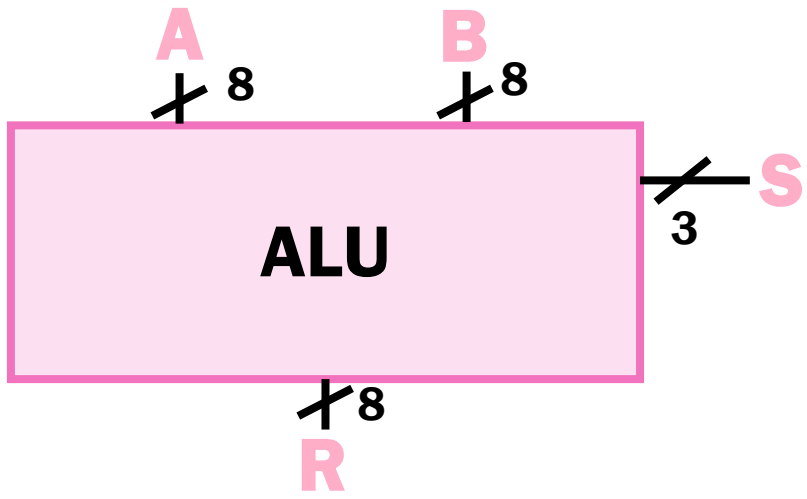


ALU

- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?

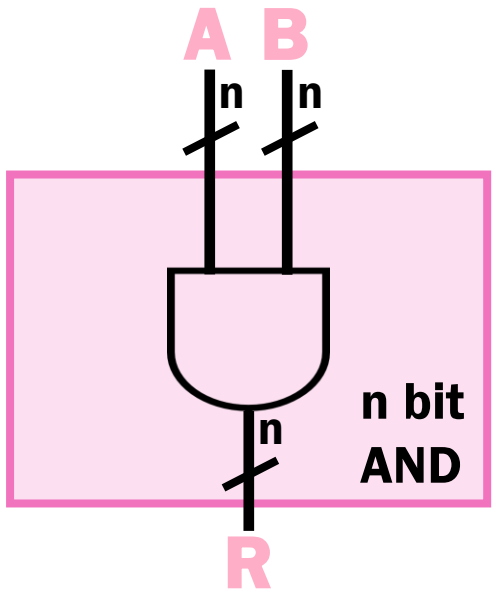
- Vamos a tener:
  - A + B
  - A - B
  - A AND B
  - A OR B
  - NOT A
  - A XOR B

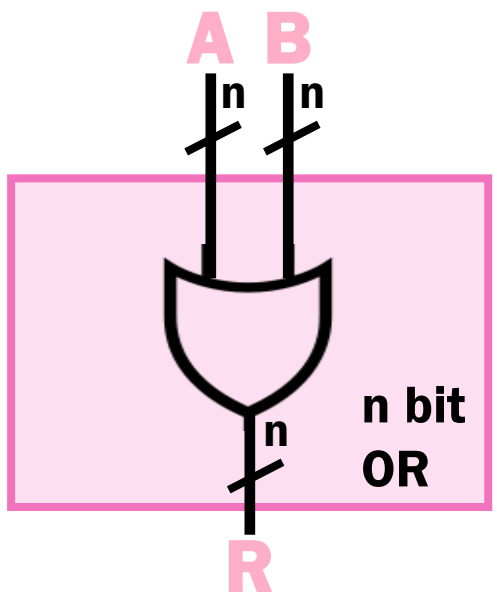
Operaciones lógicas super básicas pero muy útiles para aplicaciones más complejas

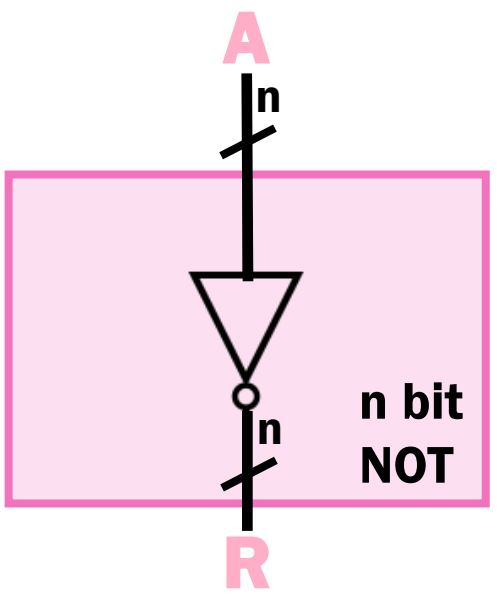


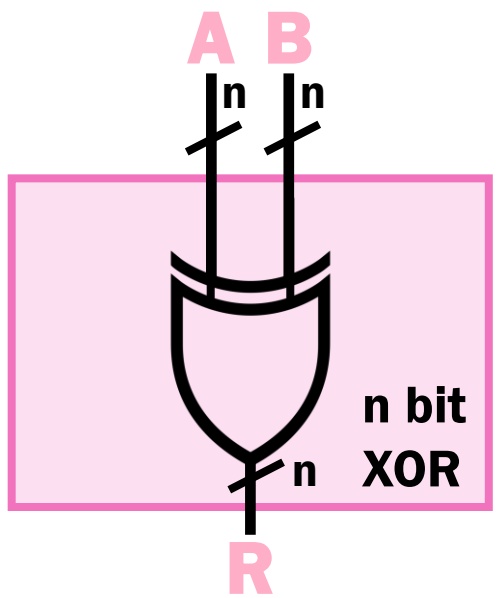
ALU

- Así se ven los circuitos de las operaciones lógicas.
- Recordar que cada operación se realiza bit a bit, esta es solo una forma de representarlo para que quede más compacto.







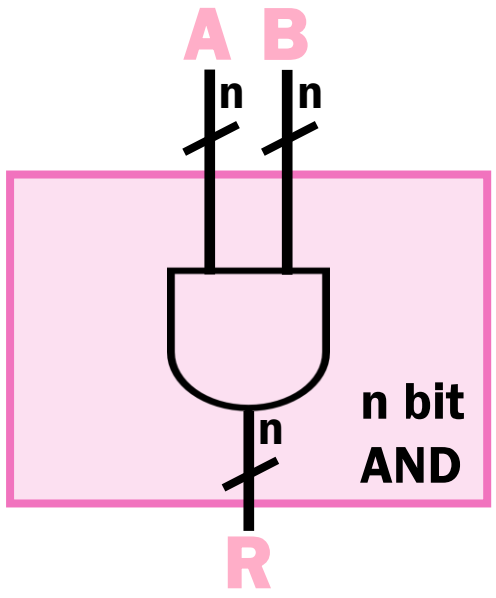


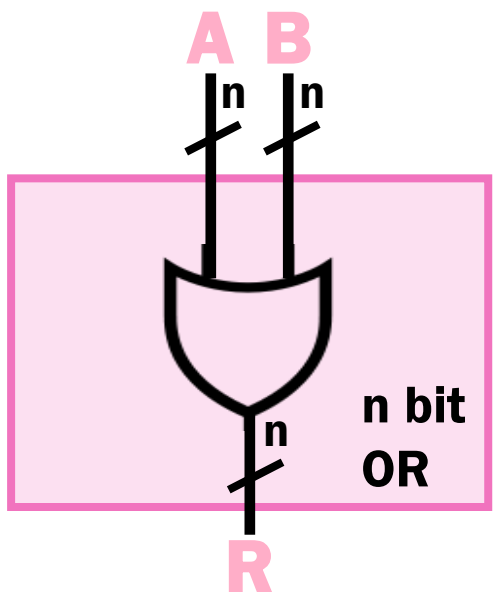
ALU

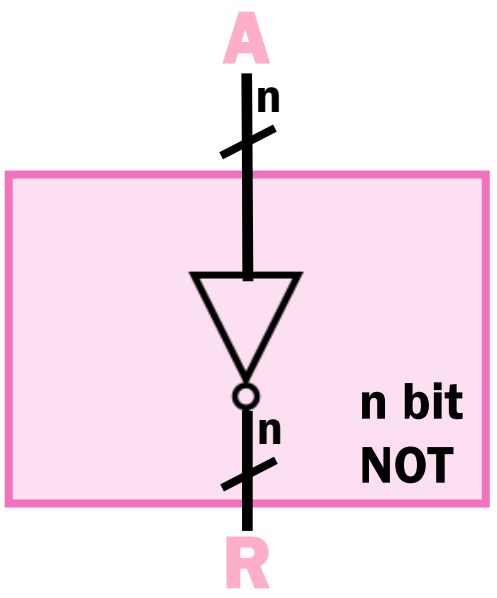
Coincidiremos...

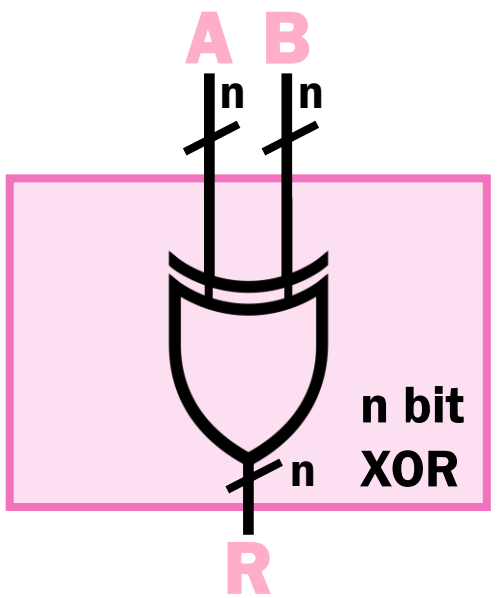
A = 1101
B = 1010

- Así se ven los circuitos de las operaciones lógicas.
- Recordar que cada operación se realiza bit a bit, esta es solo una forma de representarlo para que quede más compacto.







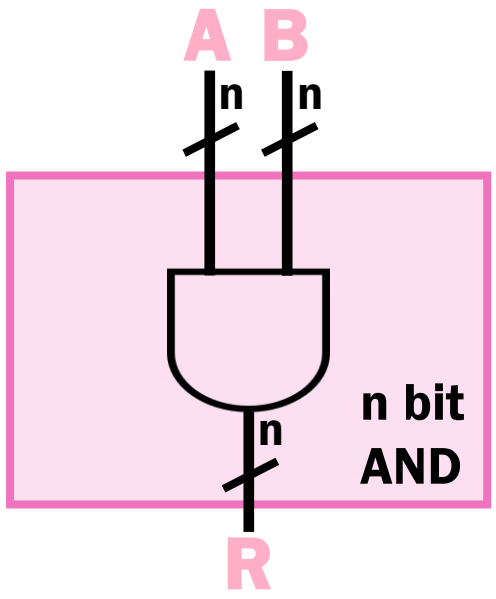


ALU

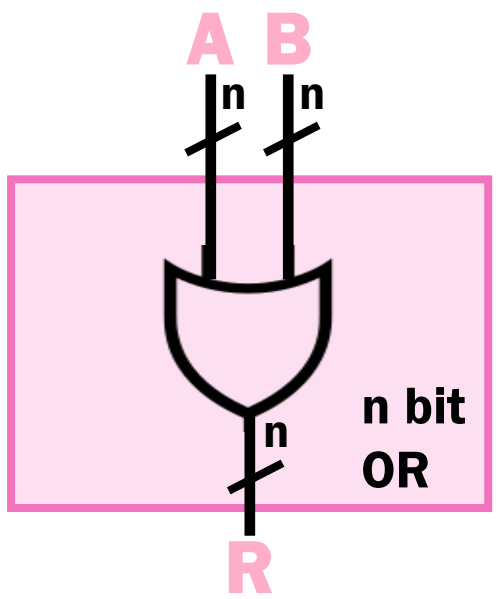
Coincidiremos...

A = 1101
B = 1010

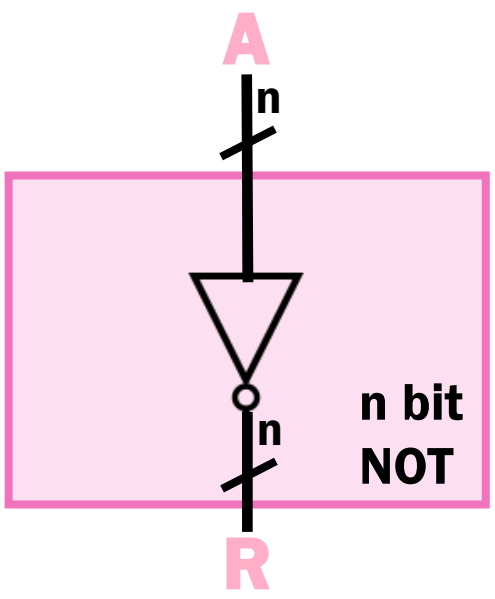
- Así se ven los circuitos de las operaciones lógicas.
- Recordar que cada operación se realiza bit a bit, esta es solo una forma de representarlo para que quede más compacto.



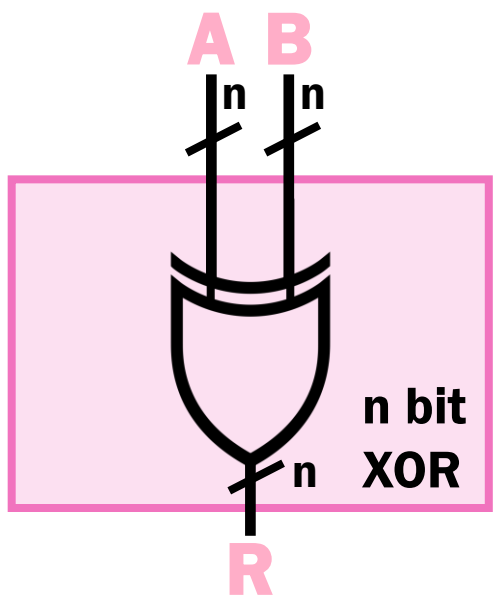
1000



1111



0010

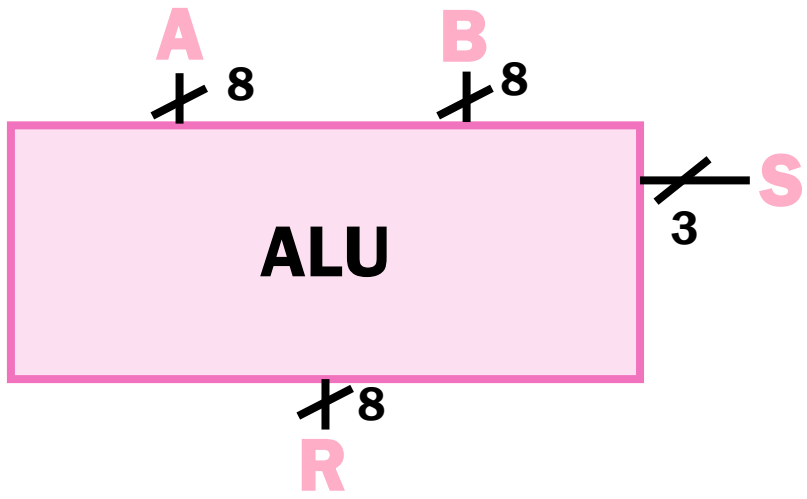


0111

ALU

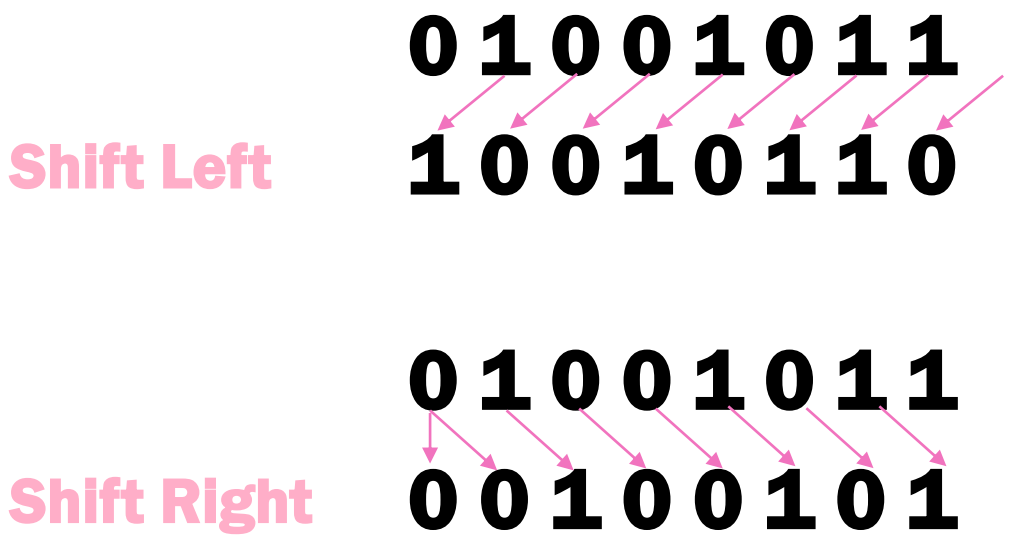
- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?

- Vamos a tener:
  - A + B
  - A - B
  - A AND B
  - A OR B
  - NOT A
  - A XOR B
  - Shift Left A
  - Shift Right A



ALU: Shifts

- Los shifts corresponden a desplazar el numero una posición a la izquierda o derecha.
  - El bit que sale se descarta.
  - Quedará una posición vacía:
    - SHL: se rellena con 0
    - SHR: se mantiene el bit



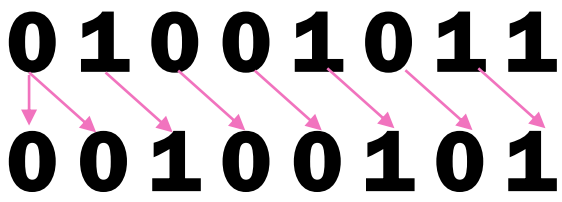
ALU: Shifts

- Los shifts corresponden a desplazar el numero una posición a la izquierda o derecha.
  - El bit que sale se descarta.
  - Quedará una posición vacía:
    - SHL: se rellena con 0
    - SHR: se mantiene el bit

por qué??

Shift Left

Shift Right



ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.
- SHL: Veamos que representa.
    - SHL(0010) = 0100

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
    - SHL(0010) = 0100
    - SHL(0101) = 1010

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
  - SHL(0010) = 0100
  - SHL(0101) = 1010

Multiplicación por 2!

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
  - SHL(0010) = 0100
  - SHL(0101) = 1010

Multiplicación por 2!

- SHR: Veamos que representa.
  - SHR(0010) = 0001

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
  - SHL(0010) = 0100
  - SHL(0101) = 1010

Multiplicación por 2!

- SHR: Veamos que representa.
  - SHR(0010) = 0001
  - SHR(0101) = 0010

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
  - SHL(0010) = 0100
  - SHL(0101) = 1010

Multiplicación por 2!

- SHR: Veamos que representa.
  - SHR(0010) = 0001
  - SHR(0101) = 0010

División entera por 2!

ALU: Shifts

- Los shifts parecen no representar nada, pero en realidad son muy útiles.

- SHL: Veamos que representa.
  - SHL(0010) = 0100
  - SHL(0101) = 1010

Multiplicación por 2!

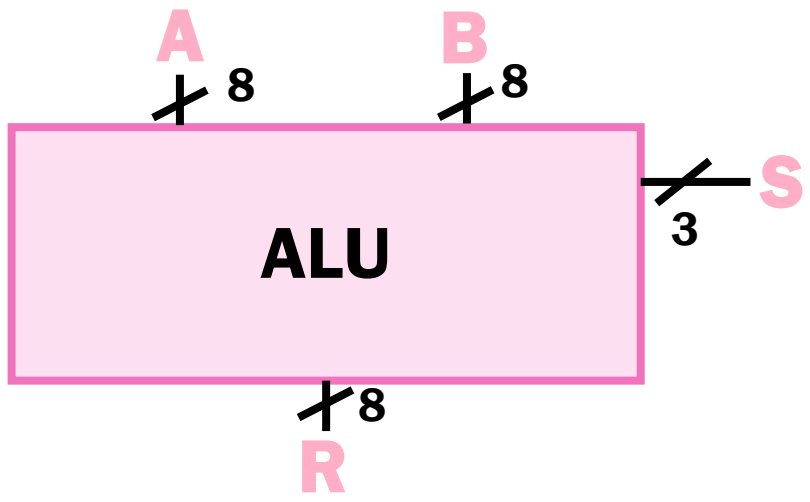
- SHR: Veamos que representa.
  - SHR(0010) = 0001
  - SHR(0101) = 0010

División entera por 2!

Estas operaciones nos permiten implementar algoritmos más eficientes.
Por ejemplo, si queremos multiplicar no necesitamos repetir la suma n veces, sino que podemos primero multiplicar por la mayor potencia de 2 (usando SHL) y luego sumar la cantidad necesaria para llegar a n.

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
----------
 100010101
```

- 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.

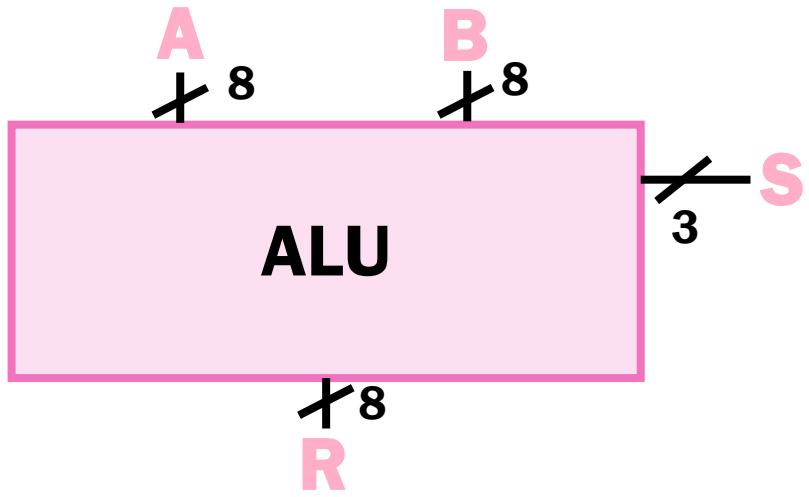
```
  01101101   (109)
+ 00010011   (19)
  ----------
  10000000   (-128)
```

- 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.

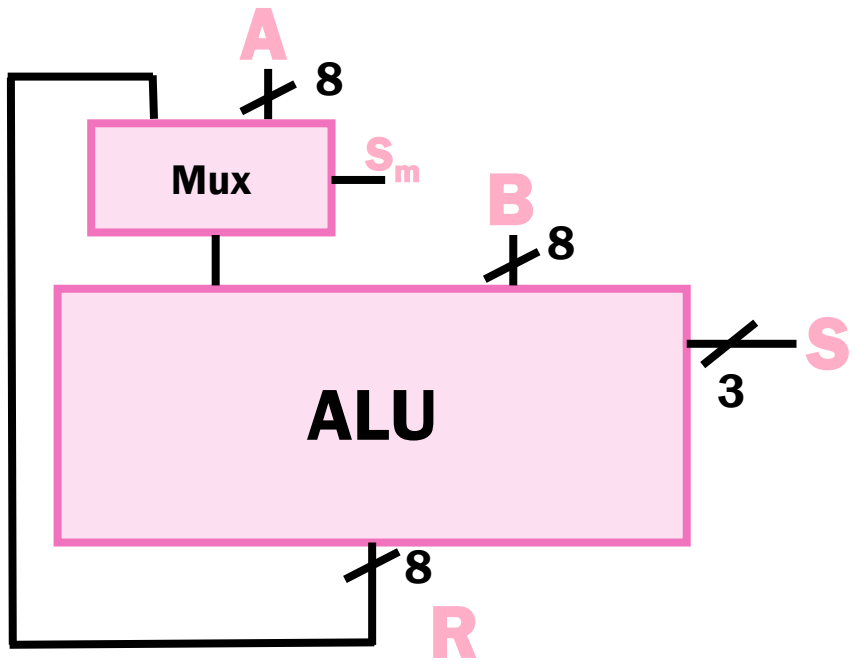


ALU

- 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.

DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 3 – ALU

Susana Figueroa