DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 2 – Lógica Digital (continuación...)

Susana Figueroa

Ahora... que pasa si queremos restar dos números binarios? Nos sirve el sumador que acabamos de diseñar?

Representación de Números Negativos

- Si queremos usar el circuito sumador para restar, necesitamos primero tener una forma de representar números negativos.

- Para poder encontrar la mejor manera de representarlos, primero hay que darnos cuenta de que propiedad queremos que cumpla nuestra representación.
- Pensemos en números decimales, tenemos que la resta cumple con lo siguiente.

$$A - B = A + (-B)$$

Es decir, que tenemos un inverso aditivo para B.

- Entonces vamos a buscar una representación para números negativos en binario tal que

$$A + (-A) = 0$$

Representación de Números Negativos: Signo / Magnitud

- Nuestra primera intuición de como representar negativos puede ser tener un bit extra que indique el signo del número.
- Es decir, que si el número es negativo el bit más significativo (más a la izquierda) debe valer 1, y si es positivo debe ser 0.

- Ejemplos de cómo quedaría esta representación:

$$3_{10} \xrightarrow{Base\ 2} 11_2 \xrightarrow{S/M} 011_2$$
$$-3_{10} \xrightarrow{Base\ 2} -11_2 \xrightarrow{S/M} 111_2$$

Es importante siempre agregar un bit extra para el signo. Si la magnitud del número se representa en n bits, entonces el número con signo quedará de n+1 bits.

Representación de Números Negativos: Signo / Magnitud

- Nuestra primera intuición de como representar negativos puede ser tener un bit extra que indique el signo del número.
- Es decir, que si el número es negativo el bit más significativo (más a la izquierda) debe valer 1, y si es positivo debe ser 0.

- Ejemplos de cómo quedaría esta representación:

$$3_{10} \xrightarrow{Base\ 2} 11_2 \xrightarrow{S/M} 011_2$$
$$-3_{10} \xrightarrow{Base\ 2} -11_2 \xrightarrow{S/M} 111_2$$

Es importante siempre agregar un bit extra para el signo. Si la magnitud del número se representa en n bits, entonces el número con signo quedará de n+1 bits. Pero cumple con tener inverso aditivo?

Representación de Números Negativos: Signo / Magnitud

- Probemos si cumple con la propiedad con un ejemplo:

$$5 + (-5)$$

Pasamos ambos números a su representación signo-magnitud

$$0101 + (1101)$$
$$= 10010$$

Entonces, la representación cumple que el inverso aditivo de un número sea el mismo, pero con signo opuesto?

Representación de Números Negativos: Signo / Magnitud

- Probemos si cumple con la propiedad con un ejemplo:

$$5 + (-5)$$

Pasamos ambos números a su representación signo-magnitud

$$0101 + (1101)$$
$$= 10010$$

Entonces, la representación cumple que el inverso aditivo de un número sea el mismo, pero con signo opuesto? **Noo!**

Representación de Números Negativos: Signo / Magnitud

- Probemos si cumple con la propiedad con un ejemplo:

$$5 + (-5)$$

Pasamos ambos números a su representación signo-magnitud

$$0101 + (1101)$$
$$= 10010$$

Entonces, la representación cumple que el inverso aditivo de un número sea el mismo, pero con signo opuesto? **Noo!**

Debemos seguir con la búsqueda de encontrar una buena representación.

Representación de Números Negativos: Complemento de 1

- Para la siguiente representación usaremos el complemento de 1 del número binario, cuál es esta? Corresponde a invertir todos los bits del número.
- Para obtener el numero en complemento de 1, es necesario que el bit más significativo sea un 0, en caso de que no lo sea se debe agregar uno.
  - Si el número a representar es negativo, todos los bits del número se invierten.
  - Si el número a representar es positivo, se mantiene igual.

- Por ejemplo,
  - $5_{10} \xrightarrow{Base\ 2} 101_2 \xrightarrow{C_1} 0101_2$
  - $-5_{10} \xrightarrow{Base\ 2} -101_2 \xrightarrow{C_1} 1010_2$

- Ahora si… Cumple esta representación con el inverso aditivo?

Representación de Números Negativos: Complemento de 1

- Probemos si cumple con la propiedad

$$5 + (-5)$$

Pasamos ambos números a su representación en complemento de 1.

$$0101 + (1010)$$
$$= 1111$$

Entonces, aunque la representación no es perfecta podemos ver que estamos llegando a algo. Qué pasa si le sumamos un 1?

Representación de Números Negativos: Complemento de 2

- Ahora veremos complemento de 2. Cual es esta representacion? Es complemento de 1 y después se suma 1.
- Para obtener el numero en complemento de 2, es necesario que el bit más significativo sea un 0, en caso de que no lo sea se debe agregar uno.
  - Si el número a representar es negativo, todos los bits del número se invierten y después se suma 1.
  - Si el número a representar es positivo, se mantiene igual.

- Por ejemplo,
  - $5_{10} \xrightarrow{Base\ 2} 101_2 \xrightarrow{C_2} 0101_2$
  - $-5_{10} \xrightarrow{Base\ 2} -101_2 \xrightarrow{C_2} 1010_2 + 1_2 = 1011_2$

- Ahora si… Cumple esta representación con el inverso aditivo?

Representación de Números Negativos: Complemento de 2

- Probemos si cumple con la propiedad

$$5 + (-5)$$

Pasamos ambos números a su representación en complemento de 2.

$$0101 + (1011)$$
$$= 10000$$

Es esto igual a 0?

Representación de Números Negativos: Complemento de 2

- Probemos si cumple con la propiedad

$$5 + (-5)$$

Pasamos ambos números a su representación en complemento de 2.

$$0101 + (1011)$$
$$= 10000$$

Es esto igual a 0? **Siii!!!**

Representación de Números Negativos: Complemento de 2

- Probemos si cumple con la propiedad

$$5 + (-5)$$

Pasamos ambos números a su representación en complemento de 2.

$$0101 + (1011)$$
$$= 10000$$

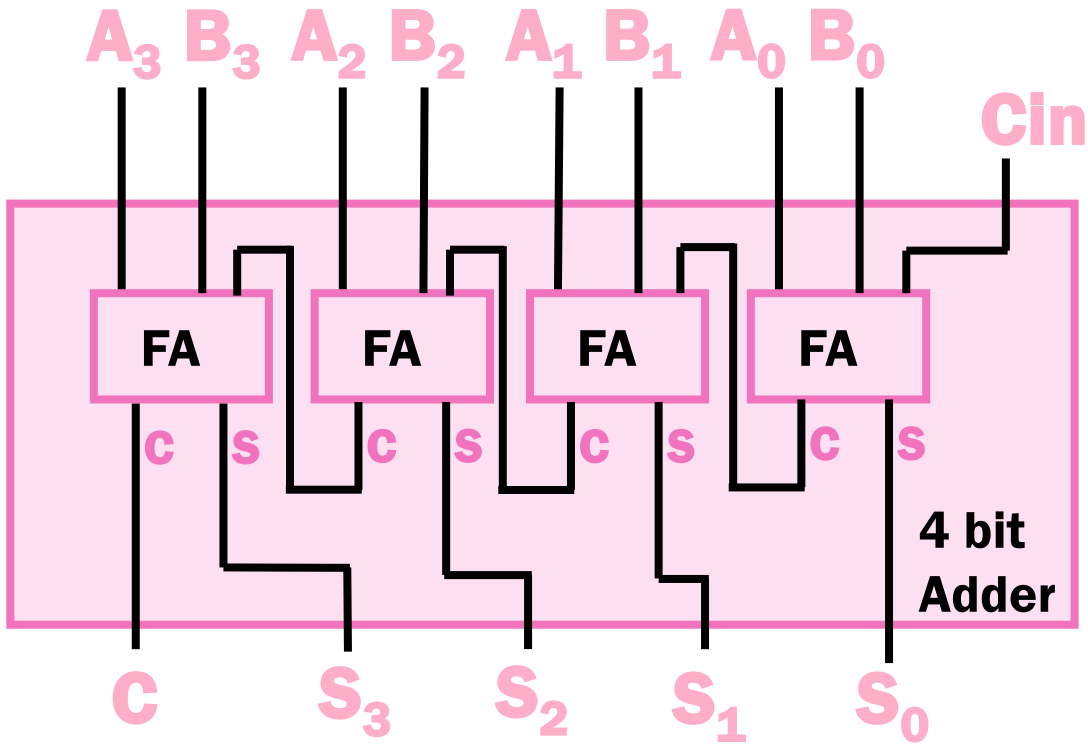
Es esto igual a 0? **Sii!!!**

Recordar que cuando sumamos números de n bits, esperamos que el resultado sea de n bits también. El resto corresponde al carry, el cual no vamos a considerar para el valor del resultado de la suma (como lo hicimos para el half adder, separamos suma y carry).

Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

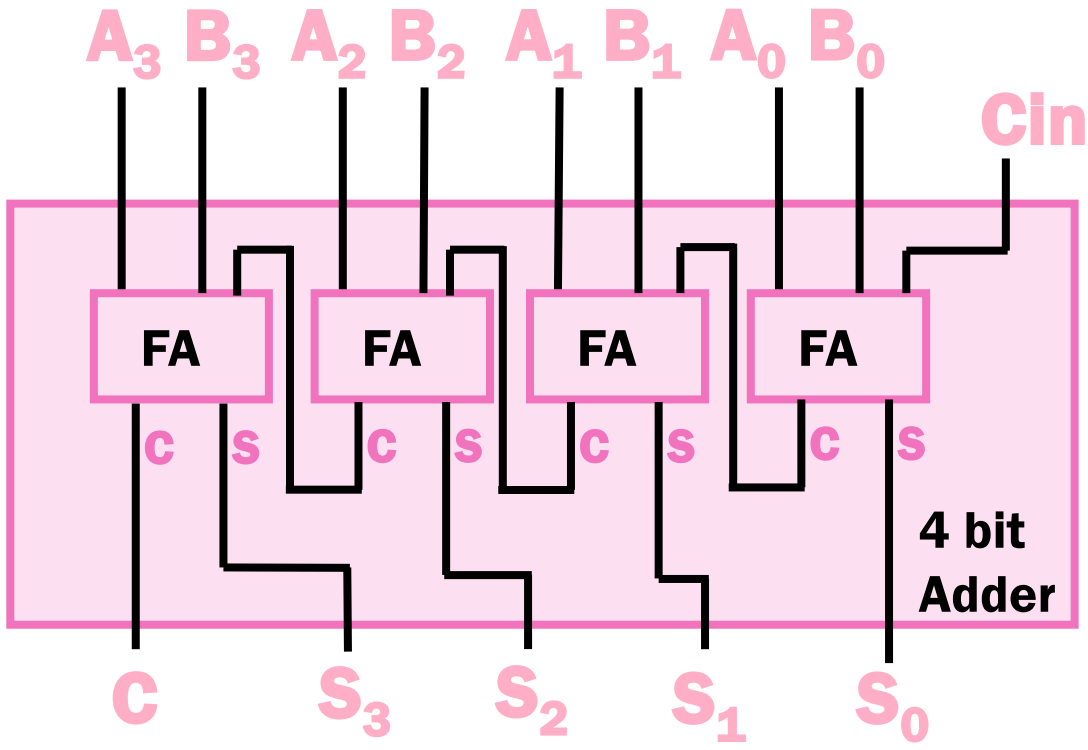


Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación?

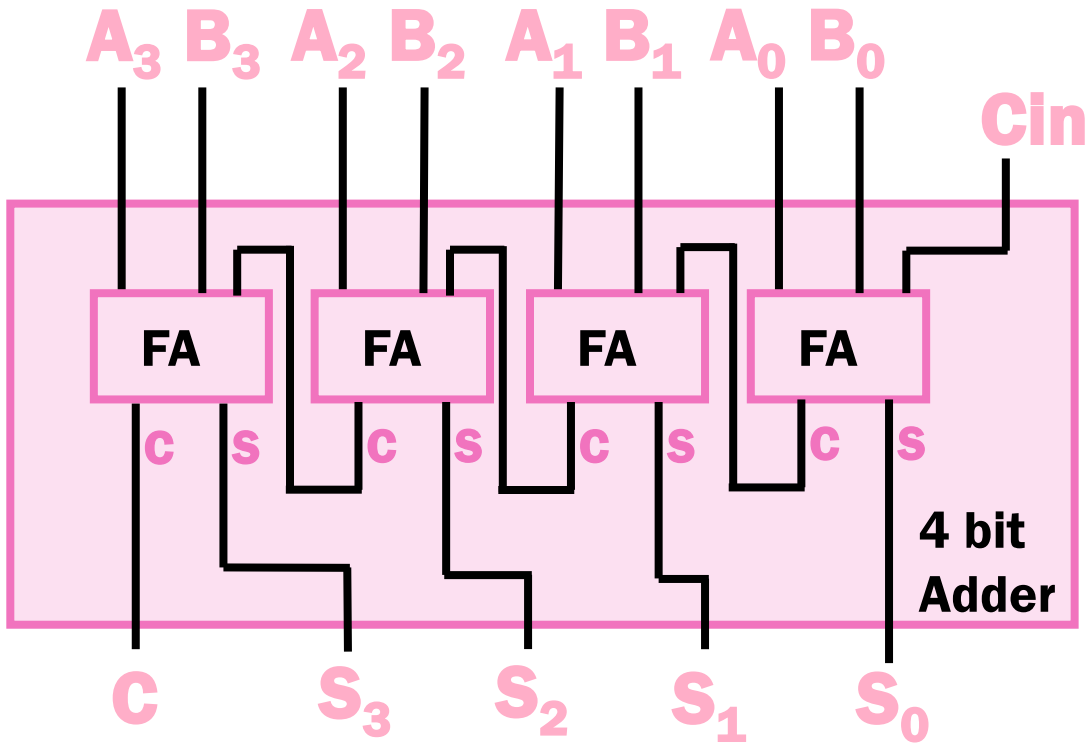


Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.



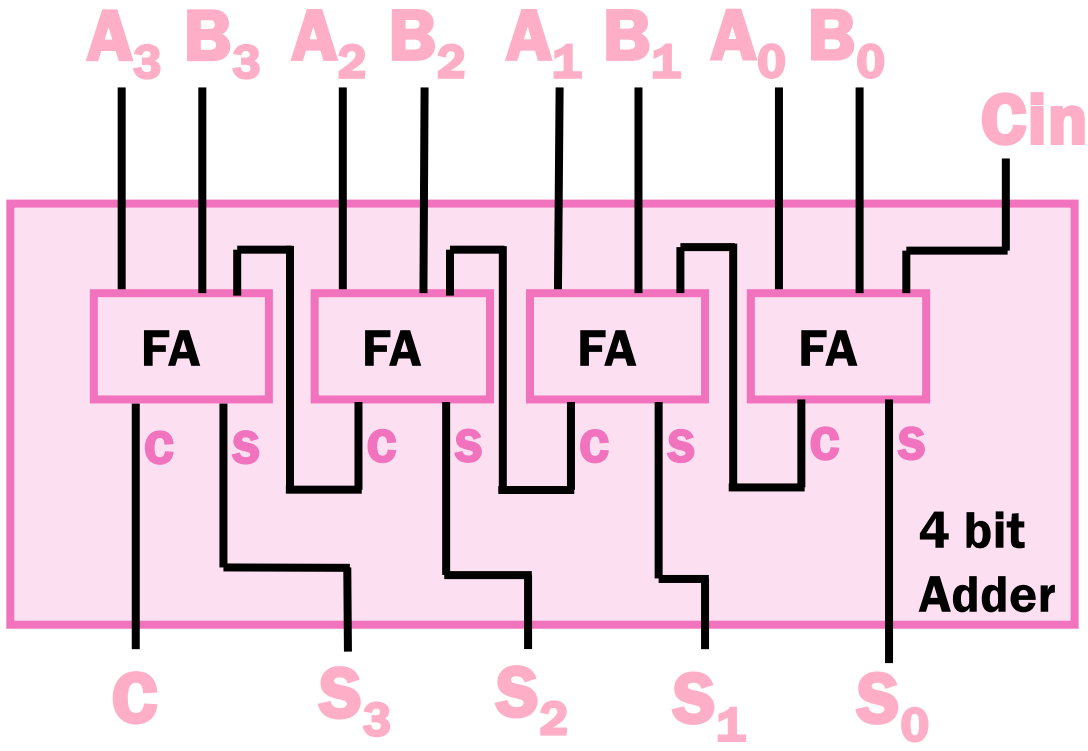
Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.

- Cómo podemos hacer que el input sea en complemento de 2?



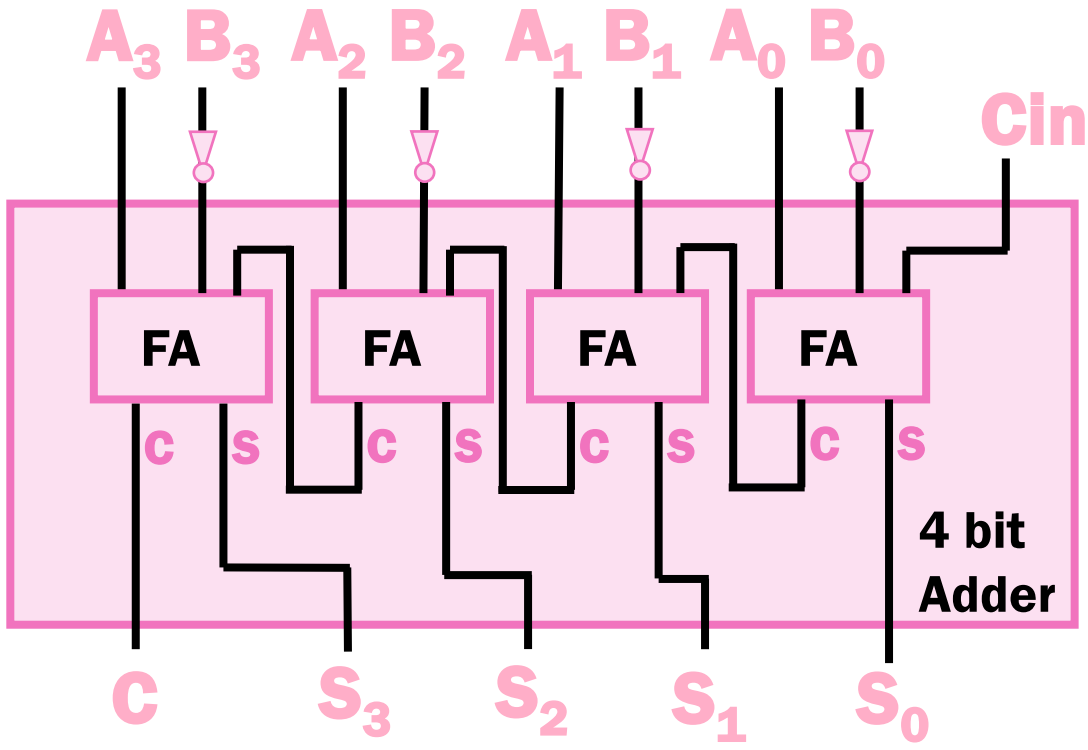
Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.

- Cómo podemos hacer que el input sea en complemento de 2? Necesitamos invertir B (negar)



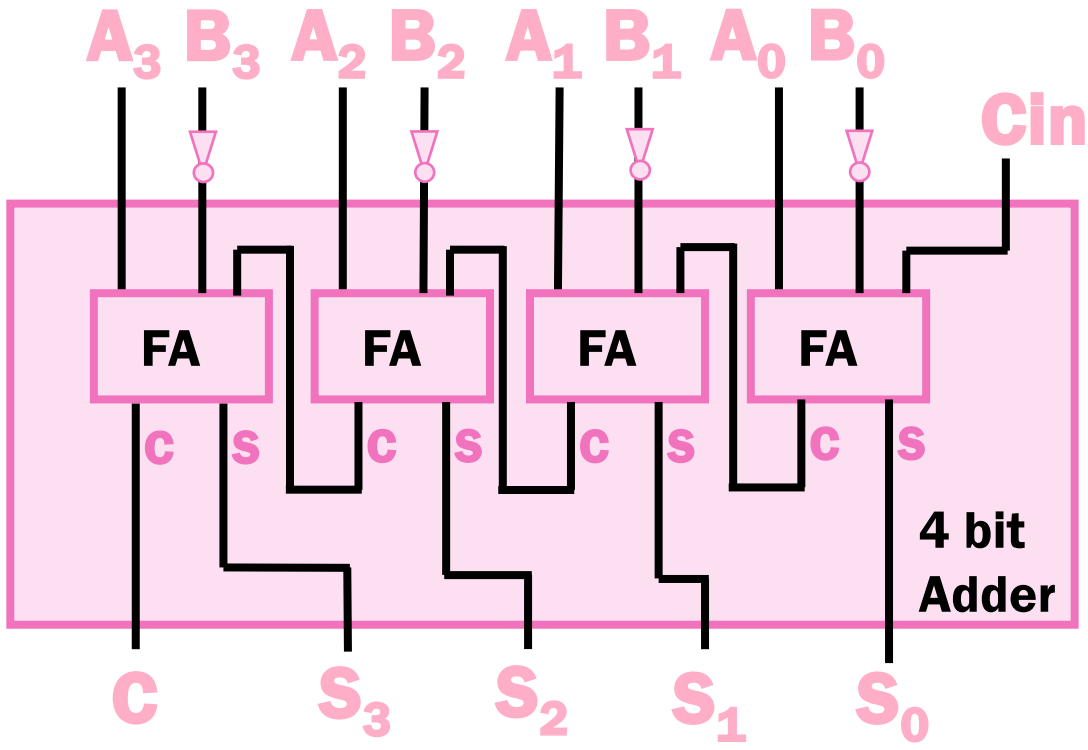
Circuito Restador

- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.

- Cómo podemos hacer que el input sea en complemento de 2? Necesitamos invertir B (negar) y sumar 1.



Circuito Restador

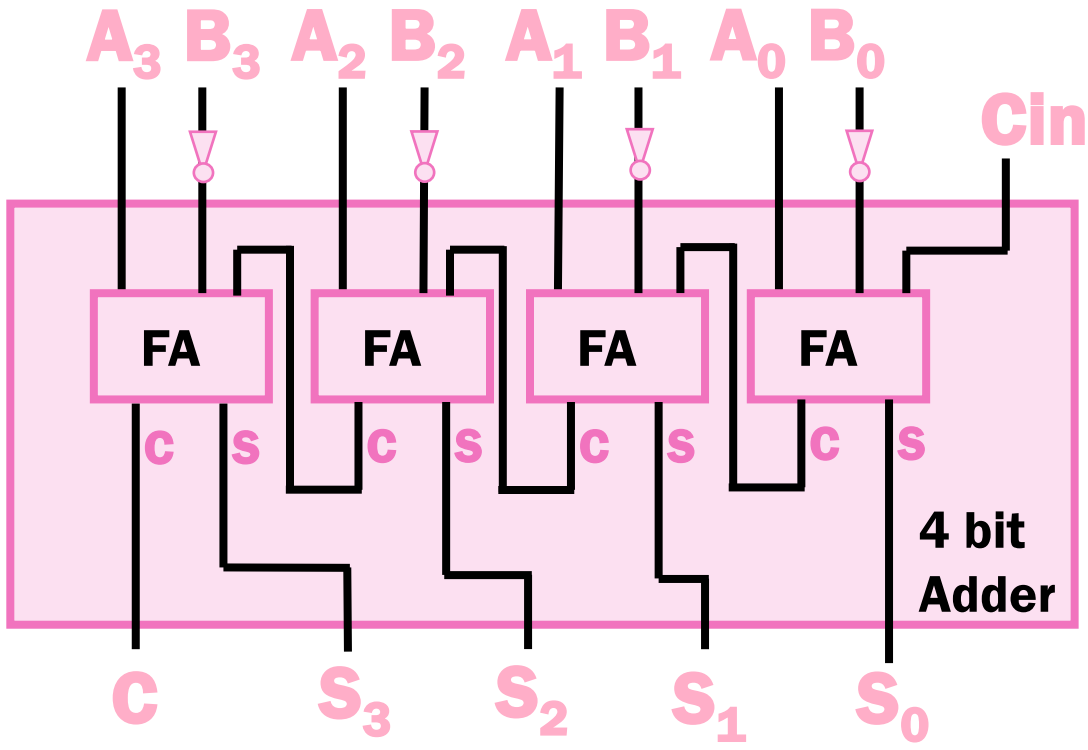
- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.

- Cómo podemos hacer que el input sea en complemento de 2? Necesitamos invertir B (negar) y sumar 1.

- Cómo podemos sumar 1?



Circuito Restador

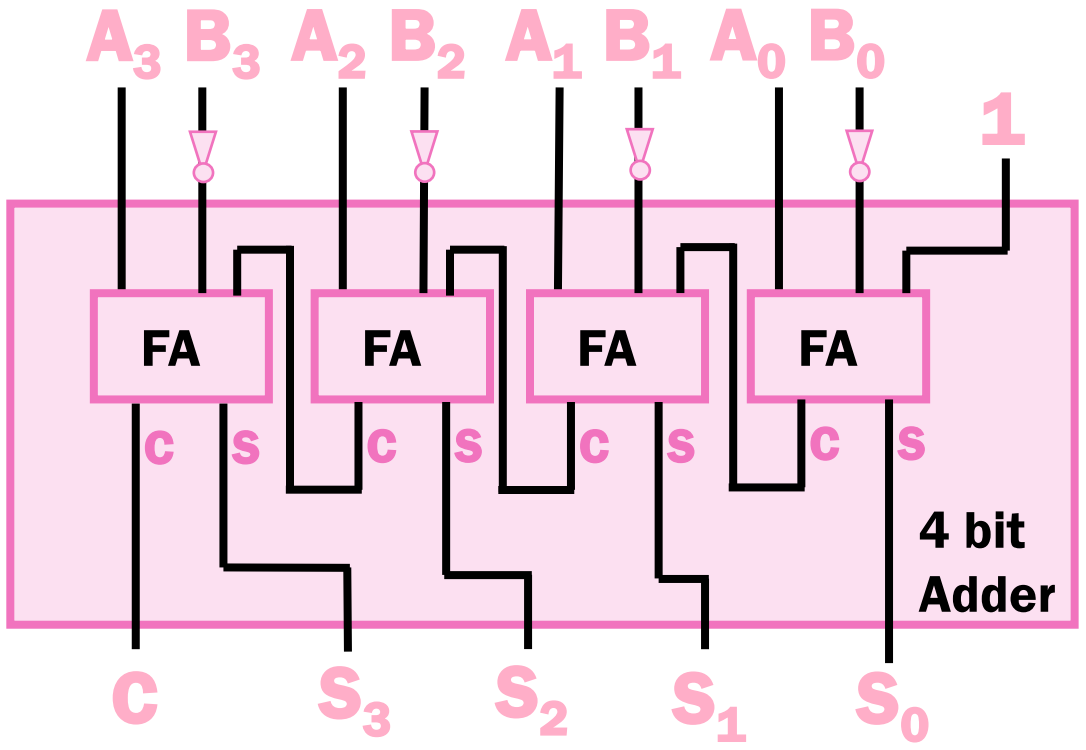
- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
  - $A - B = A + (-B)$

- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.

- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.

- Cómo podemos hacer que el input sea en complemento de 2? Necesitamos invertir B (negar) y sumar 1.

- Cómo podemos sumar 1? Podemos usar el carry inicial para sumar.



Extra: Minterms y Maxterms

- Para poder obtener un circuito combinacional a partir de una tabla de verdad es super sencillo, talvez es tedioso pero fácil.

- Primero, debes identificar cuantos 1's y 0's.
  - Si hay **menos** 1's: **Minterms**
  - Si hay **menos** 0's: **Maxterms**
  - Si hay igual cantidad: Da lo mismo cual usar

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Extra: Minterms y Maxterms

- Para poder obtener un circuito combinacional a partir de una tabla de verdad es super sencillo, talvez es tedioso pero fácil.

- Primero, debes identificar cuantos 1's y 0's hay en el output.
  - Si hay **menos** 1's: **Minterms**
  - Si hay **menos** 0's: **Maxterms**
  - Si hay igual cantidad: Da lo mismo cual usar

- Contemos primero la cantidad de 1's.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Extra: Minterms y Maxterms

- Para poder obtener un circuito combinacional a partir de una tabla de verdad es super sencillo, talvez es tedioso pero fácil.

- Primero, debes identificar cuantos 1's y 0's hay en el output.
  - Si hay **menos** 1's: **Minterms**
  - Si hay **menos** 0's: **Maxterms**
  - Si hay igual cantidad: Da lo mismo cual usar

- Contemos primero la cantidad de 1's.
- Contemos la cantidad de 0's.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Extra: Minterms y Maxterms

- Para poder obtener un circuito combinacional a partir de una tabla de verdad es super sencillo, talvez es tedioso pero fácil.

- Primero, debes identificar cuantos 1's y 0's hay en el output.
  - Si hay **menos** 1's: **Minterms**
  - Si hay **menos** 0's: **Maxterms**
  - Si hay igual cantidad: Da lo mismo cual usar

- Contemos primero la cantidad de 1's.
- Contemos la cantidad de 0's.

Como tienen la misma cantidad no importa cual usar.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Minterms

- Partimos con los minterms. La idea es crear una formula combinacional que de siempre 0, a menos que se cumpla algunos de los casos en donde el output sea 1.

- Cómo se logra esto? Forma normal disyuntiva (DNF)
  - Por cada caso en donde el output sea 1:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas AND (∧).
    - Cuando se debe negar el input? Cuando aparece como un 0.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Minterms

- Partimos con los minterms. La idea es crear una formula combinacional que de siempre 0, a menos que se cumpla algunos de los casos en donde el output sea 1.

- Cómo se logra esto? Forma normal disyuntiva (DNF)
  - Por cada caso en donde el output sea 1:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas AND (∧).
    - Cuando se debe negar el input? Cuando aparece como un 0.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x e y valen 1, aparecen tal cual en la formula. Por otro lado, z vale 0, entonces se debe usar su versión negada.

$$m_3 = x \cdot y \cdot \bar{z}$$

Minterms

- Partimos con los minterms. La idea es crear una formula combinacional que de siempre 0, a menos que se cumpla algunos de los casos en donde el output sea 1.

- Cómo se logra esto? Forma normal disyuntiva (DNF)
  - Por cada caso en donde el output sea 1:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas AND (∧).
    - Cuando se debe negar el input? Cuando aparece como un 0.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x e y valen 1, aparecen tal cual en la formula. Por otro lado, z vale 0, entonces se debe usar su versión negada.

$$m_3 = x \cdot y \cdot \bar{z}$$
$$m_5 = \bar{x} \cdot y \cdot z$$

Minterms

- Partimos con los minterms. La idea es crear una formula combinacional que de siempre 0, a menos que se cumpla algunos de los casos en donde el output sea 1.

- Cómo se logra esto? Forma normal disyuntiva (DNF)
  - Por cada caso en donde el output sea 1:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas AND (∧).
    - Cuando se debe negar el input? Cuando aparece como un 0.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x e y valen 1, aparecen tal cual en la formula. Por otro lado, z vale 0, entonces se debe usar su versión negada.

$$m_3 = x \cdot y \cdot \bar{z}$$
$$m_5 = \bar{x} \cdot y \cdot z$$
$$m_7 = x \cdot \bar{y} \cdot z$$

Minterms

- Partimos con los minterms. La idea es crear una formula combinacional que de siempre 0, a menos que se cumpla algunos de los casos en donde el output sea 1.

- Cómo se logra esto? Forma normal disyuntiva (DNF)
  - Por cada caso en donde el output sea 1:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas AND (∧).
    - Cuando se debe negar el input? Cuando aparece como un 0.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x e y valen 1, aparecen tal cual en la formula. Por otro lado, z vale 0, entonces se debe usar su versión negada.

$$m_3 = x \cdot y \cdot \bar{z}$$
$$m_5 = \bar{x} \cdot y \cdot z$$
$$m_7 = x \cdot \bar{y} \cdot z$$
$$m_8 = x \cdot y \cdot z$$

Minterms

$$m_3 = x \cdot y \cdot \bar{z}$$
$$m_5 = \bar{x} \cdot y \cdot z$$
$$m_7 = x \cdot \bar{y} \cdot z$$
$$m_8 = x \cdot y \cdot z$$

- Una vez que ya se tienen todas las fórmulas de las combinaciones, se deben juntar usando compuertas OR's (V).

- Entonces la fórmula que se obtiene para la tabla de verdad es la siguiente.

$$f(x, y, z) = (x \cdot y \cdot \bar{z}) + (\bar{x} \cdot y \cdot z) + (x \cdot \bar{y} \cdot z) + (x \cdot y \cdot z)$$

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Minterms

$$m_3 = x \cdot y \cdot \bar{z}$$
$$m_5 = \bar{x} \cdot y \cdot z$$
$$m_7 = x \cdot \bar{y} \cdot z$$
$$m_8 = x \cdot y \cdot z$$

- Una vez que ya se tienen todas las fórmulas de las combinaciones, se deben juntar usando compuertas OR's (V).

- Entonces la fórmula que se obtiene para la tabla de verdad es la siguiente.

$$f(x, y, z) = (x \cdot y \cdot \bar{z}) + (\bar{x} \cdot y \cdot z) + (x \cdot \bar{y} \cdot z) + (x \cdot y \cdot z)$$

Pueden comprobar que si reemplazan con cualquier valor de la tabla de verdad, la formula va a dar el resultado esperado.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Maxterms

- Para los maxterms es el mismo procedimiento, pero inverso. La idea es crear una formula combinacional que de siempre 1, a menos que se cumpla algunos de los casos en donde el output sea 0.

- Cómo se logra esto? Forma normal conjuntiva (CNF)
  - Por cada caso en donde el output sea 0:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas OR (V).
    - Cuando se debe negar el input? Cuando aparece como un 1.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Maxterms

- Para los maxterms es el mismo procedimiento, pero inverso. La idea es crear una formula combinacional que de siempre 1, a menos que se cumpla algunos de los casos en donde el output sea 0.

- Cómo se logra esto? Forma normal conjuntiva (CNF)
  - Por cada caso en donde el output sea 0:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas OR (V).
    - Cuando se debe negar el input? Cuando aparece como un 1.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x, y, y z aparecen como 0, ninguno se debe negar.

$$M_0 = x + y + z$$

Maxterms

- Para los maxterms es el mismo procedimiento, pero inverso. La idea es crear una formula combinacional que de siempre 1, a menos que se cumpla algunos de los casos en donde el output sea 0.

- Cómo se logra esto? Forma normal conjuntiva (CNF)
  - Por cada caso en donde el output sea 0:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas OR (V).
    - Cuando se debe negar el input? Cuando aparece como un 1.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x, y, y z aparecen como 0, ninguno se debe negar.

$$M_0 = x + y + z$$
$$M_1 = x + \bar{y} + z$$

Maxterms

- Para los maxterms es el mismo procedimiento, pero inverso. La idea es crear una formula combinacional que de siempre 1, a menos que se cumpla algunos de los casos en donde el output sea 0.

- Cómo se logra esto? Forma normal conjuntiva (CNF)
  - Por cada caso en donde el output sea 0:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas OR (V).
    - Cuando se debe negar el input? Cuando aparece como un 1.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x, y, y z aparecen como 0, ninguno se debe negar.

$$M_0 = x + y + z$$
$$M_1 = x + \bar{y} + z$$
$$M_2 = \bar{x} + y + z$$

Maxterms

- Para los maxterms es el mismo procedimiento, pero inverso. La idea es crear una formula combinacional que de siempre 1, a menos que se cumpla algunos de los casos en donde el output sea 0.

- Cómo se logra esto? Forma normal conjuntiva (CNF)
  - Por cada caso en donde el output sea 0:
    - Todos los inputs deben aparecer ya sea en su forma “normal” o negada, unidas por compuertas OR (V).
    - Cuando se debe negar el input? Cuando aparece como un 1.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

- Partamos con el primer caso. Como x, y, y z aparecen como 0, ninguno se debe negar.

$$M_0 = x + y + z$$
$$M_1 = x + \bar{y} + z$$
$$M_2 = \bar{x} + y + z$$
$$M_4 = x + y + \bar{z}$$

Maxterms

$$M_0 = x + y + z$$
$$M_1 = x + \bar{y} + z$$
$$M_2 = \bar{x} + y + z$$
$$M_4 = x + y + \bar{z}$$

- Una vez que ya se tienen todas las fórmulas de las combinaciones, se deben juntar usando compuertas AND's (∧).

- Entonces la fórmula que se obtiene para la tabla de verdad es la siguiente.

$$f(x, y, z) = (x + y + z) \cdot (x + \bar{y} + z) \cdot (\bar{x} + y + z) \cdot (x + y + \bar{z})$$

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Maxterms

$$M_0 = x + y + z$$
$$M_1 = x + \bar{y} + z$$
$$M_2 = \bar{x} + y + z$$
$$M_4 = x + y + \bar{z}$$

- Una vez que ya se tienen todas las fórmulas de las combinaciones, se deben juntar usando compuertas AND's (∧).

- Entonces la fórmula que se obtiene para la tabla de verdad es la siguiente.

$$f(x, y, z) = (x + y + z) \cdot (x + \bar{y} + z) \cdot (\bar{x} + y + z) \cdot (x + y + \bar{z})$$

Pueden comprobar que si reemplazan con cualquier valor de la tabla de verdad, la formula va a dar el resultado esperado.

x	y	z	f (x, y, z)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Enabler

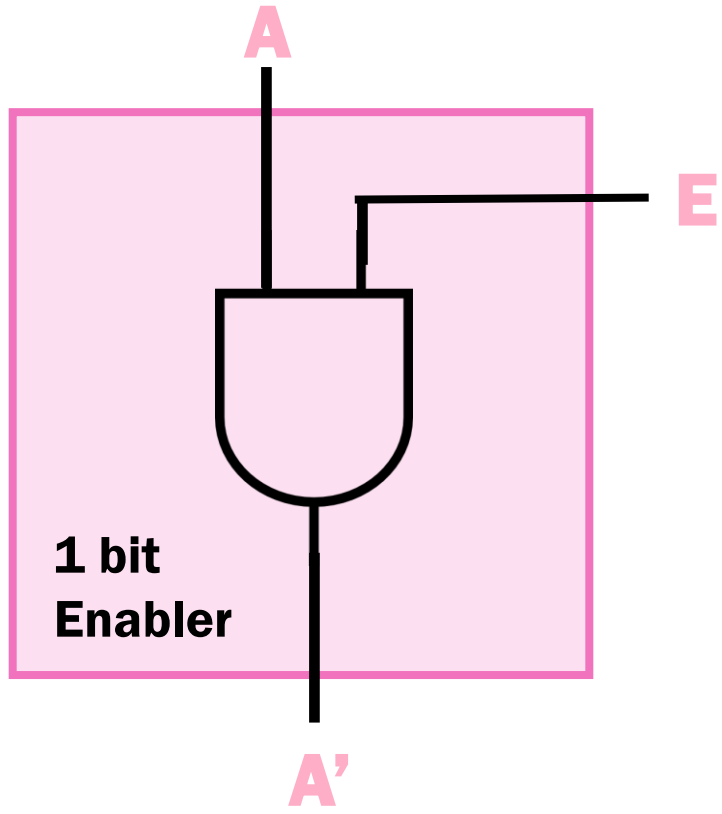
- Maravilloso.. Ya tenemos una forma de sumar y restar números binarios usando el mismo circuito, pero como hacemos para compartirlo? Tenemos que tener alguna manera de avisarle al sumador cuando se deben sumar o restar los números.
- Tenemos que tener alguna forma de controlar que input entra al circuito sumador.
- La forma de controlar que tenemos es usar una **señal de control**, es decir que mediante otro input indicaremos que operación queremos realizar.
- El primer circuito que nos permitirá seleccionar la salida es el **1 bit enabler**.

Enabler

- El 1 bit **enabler** nos permite controlar la salida de un circuito a partir de la señal de control E.

- Depende de la señal de control E, la salida del circuito.
  - Si E=0, la salida del circuito va a ser 0.
  - Si E=1, entonces la salida del circuito va a ser el valor de A (este puede ser 0 o 1).

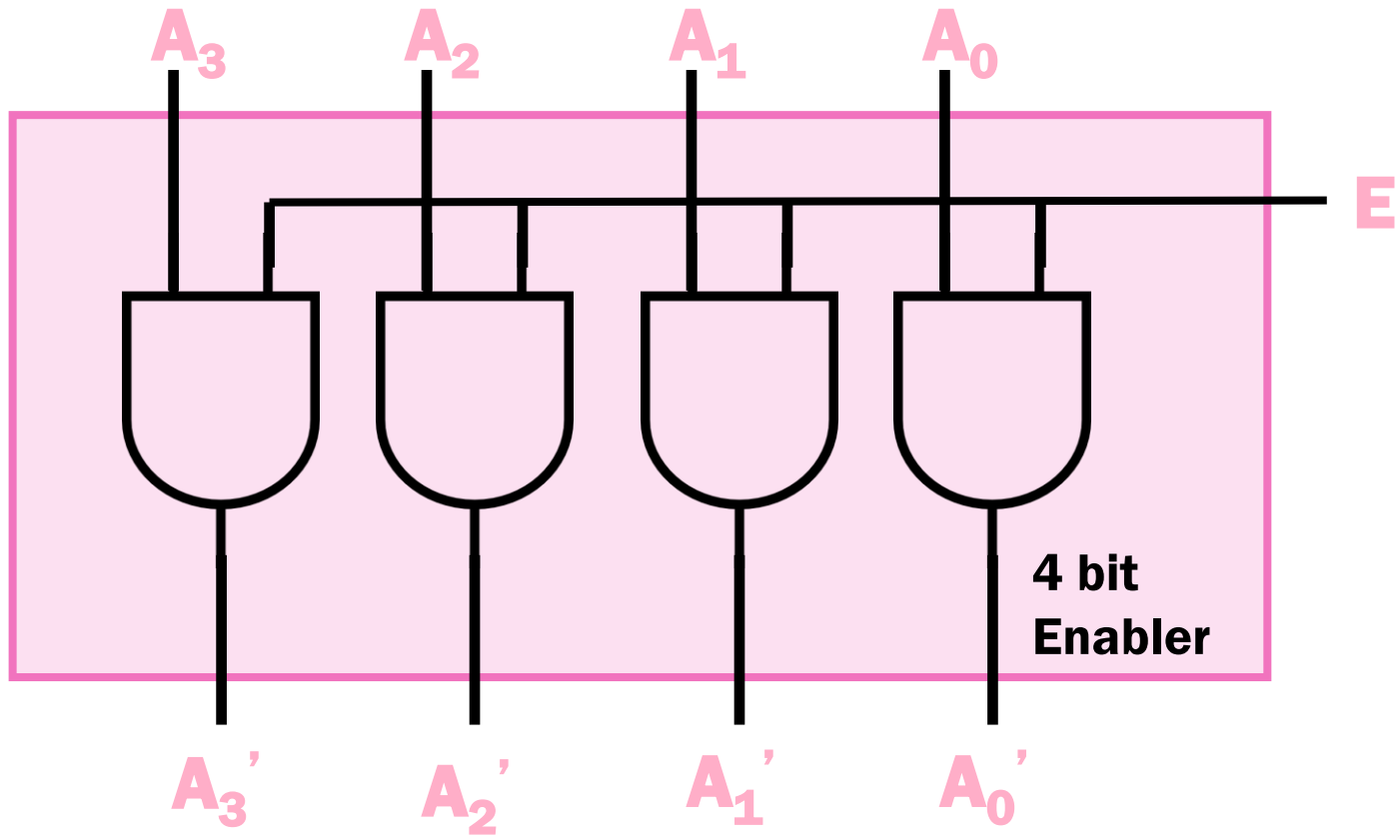
E	A’
0	0
1	A



Enabler

- Este circuito se puede extender a más bits, por ejemplo a 4.

E	A’
0	0
1	A



Multiplexor

- Otra forma que tenemos para poder controlar la salida de un circuito a partir de una señal de control son los **multiplexores**.
- Para que nos sirven? A diferencia de los enablers, podemos usar todos los casos de la señal de control para direccionar la salida.
- A la señal de control de los multiplexores también se le llama **selector**.
- Por ejemplo, si tenemos un selector de 1 bit significa que podemos hacer que el output del circuito sea A (si S = 0), o B (si S = 1).

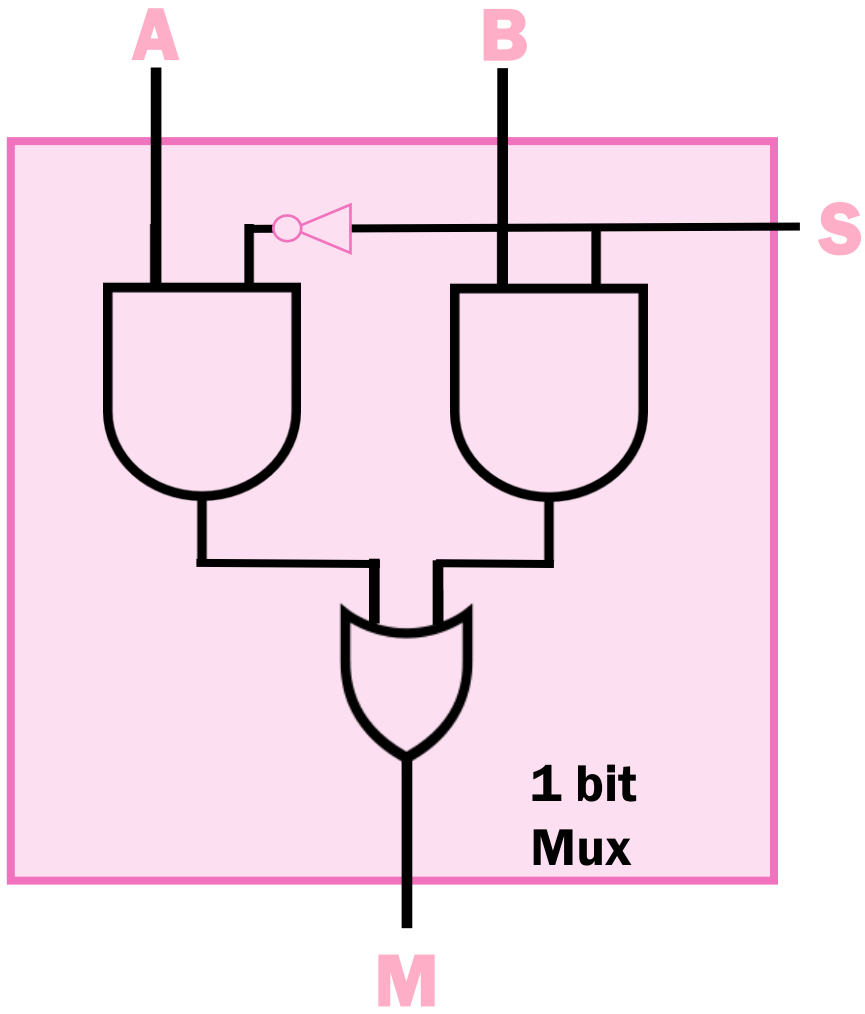
Depende de la cantidad de bits del selector, la cantidad de entradas que se pueden seleccionar.

Multiplexor

- Veamos cómo se ve un multiplexor con entradas de 1 bit y selector de 1 bit.

- El valor de M depende del selector S.
  - Si S=0, M toma el valor de A
  - Si S=1, M toma el valor de B

S	M
0	A
1	B

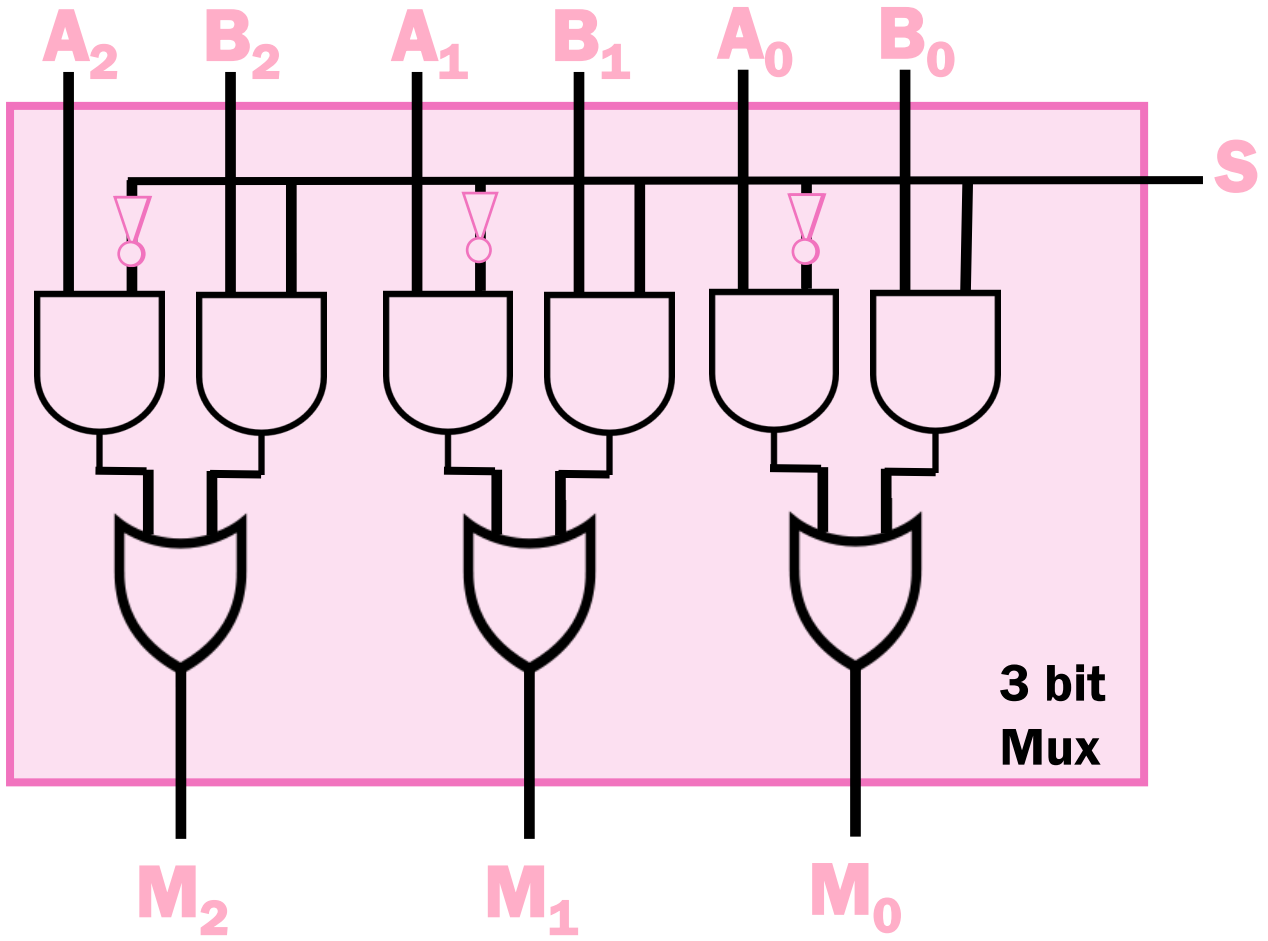


Multiplexor

- Ahora, como es un multiplexor con entradas de 3 bits y selector de 1 bit.

- El valor de M depende del selector S.
  - Si S=0, M toma el valor de A
  - Si S=1, M toma el valor de B

S	M
0	A
1	B

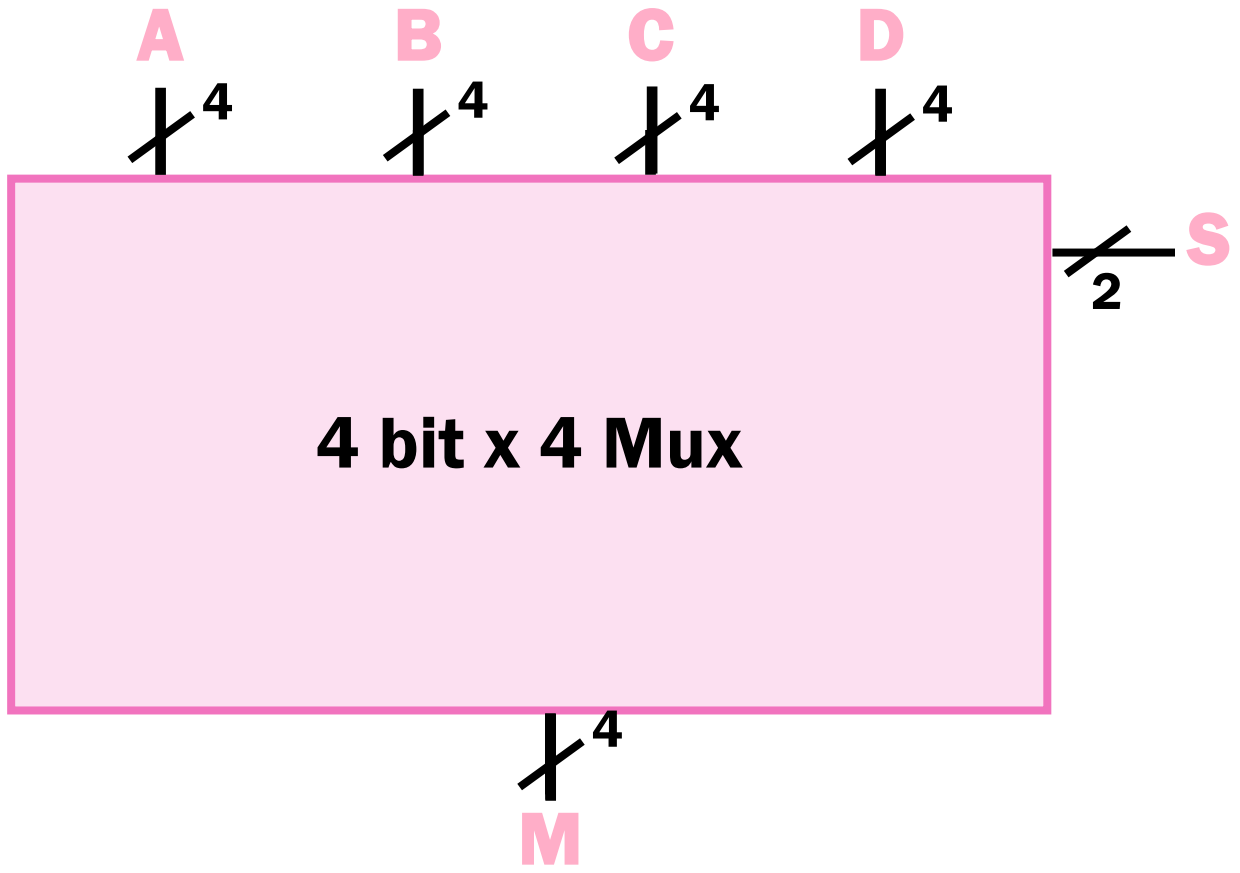


Multiplexor

- Ahora, como es un multiplexor con entradas de 4 bits, pero un selector de 2 bits, esto nos permite tener más entradas.

- El valor de M depende del selector S.
  - Si S=0, M toma el valor de A
  - Si S=1, M toma el valor de B
  - Si S=2, M toma el valor de C
  - Si S=3, M toma el valor de D

S_1	S_0	M
0	0	A
0	1	B
1	0	C
1	1	D



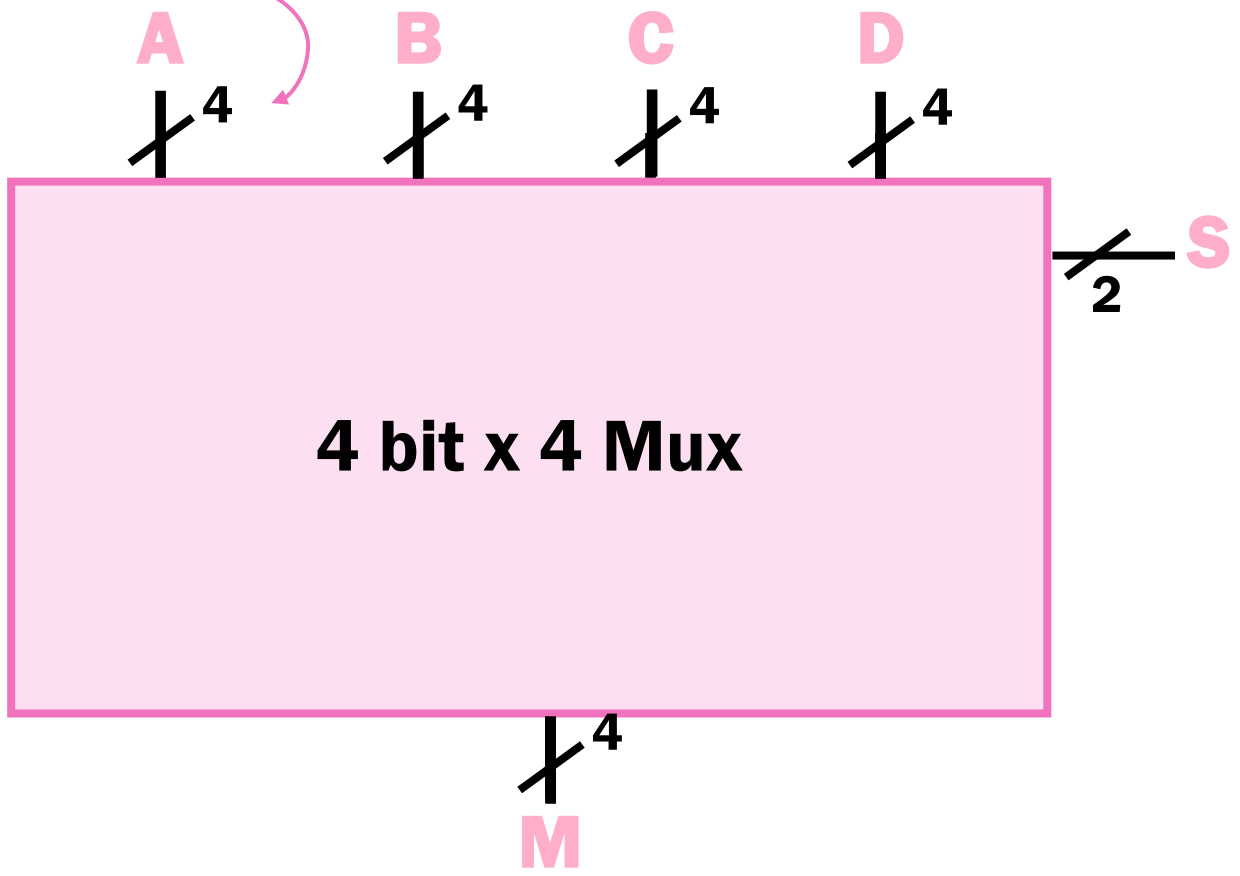
Multiplexor

- Ahora, como es un multiplexor con entradas de 4 bits, pero un selector de 2 bits, esto nos permite tener más entradas.

- El valor de M depende del selector S.
  - Si S=0, M toma el valor de A
  - Si S=1, M toma el valor de B
  - Si S=2, M toma el valor de C
  - Si S=3, M toma el valor de D

S_1	S_0	M
0	0	A
0	1	B
1	0	C
1	1	D

Podemos abreviar un poco el circuito utilizando la notación de bus de datos, esto quiere decir que, en vez de ser 1 bit, son 4 bits.



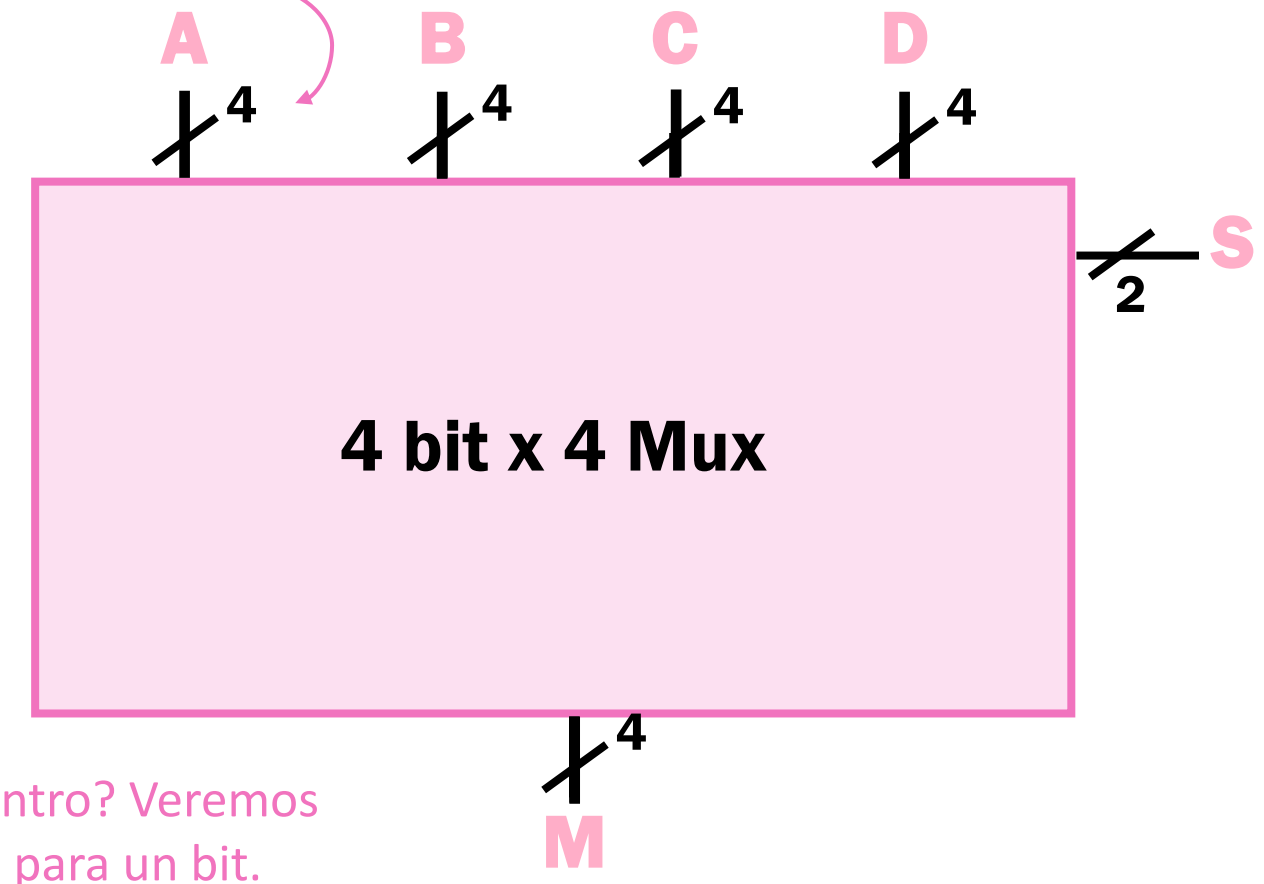
Multiplexor

- Ahora, como es un multiplexor con entradas de 4 bits, pero un selector de 2 bits, esto nos permite tener más entradas.

- El valor de M depende del selector S.
  - Si S=0, M toma el valor de A
  - Si S=1, M toma el valor de B
  - Si S=2, M toma el valor de C
  - Si S=3, M toma el valor de D

S_1	S_0	M
0	0	A
0	1	B
1	0	C
1	1	D

Podemos abreviar un poco el circuito utilizando la notación de bus de datos, esto quiere decir que, en vez de ser 1 bit, son 4 bits.

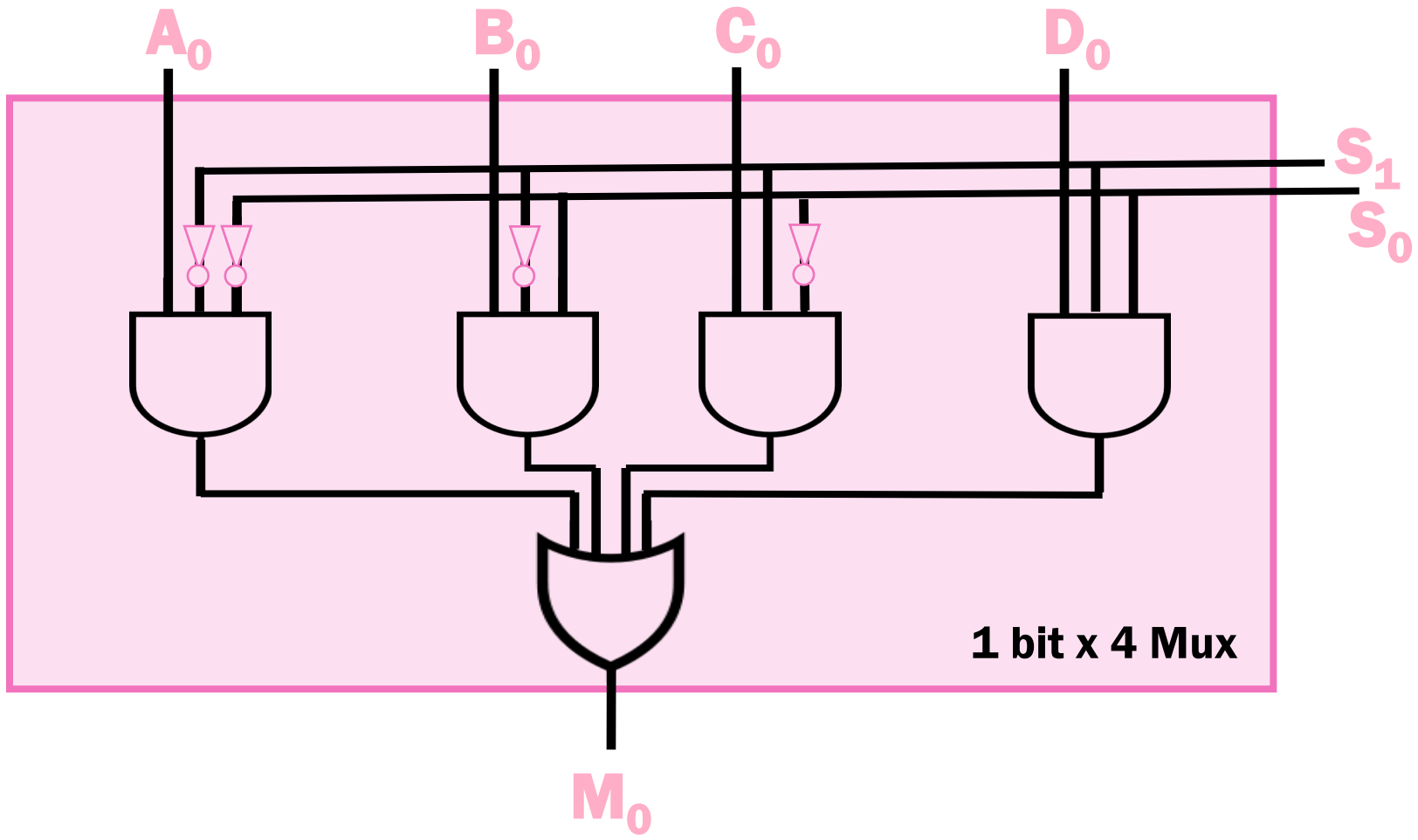


Cómo se ve por dentro? Veremos cómo se comporta para un bit.

Multiplexor

- Podemos ver cómo funciona el multiplexor de varias entradas para un bit.
- Para extenderlo a n bits solo se necesitan replicar varios de estos por cada bit de los inputs.

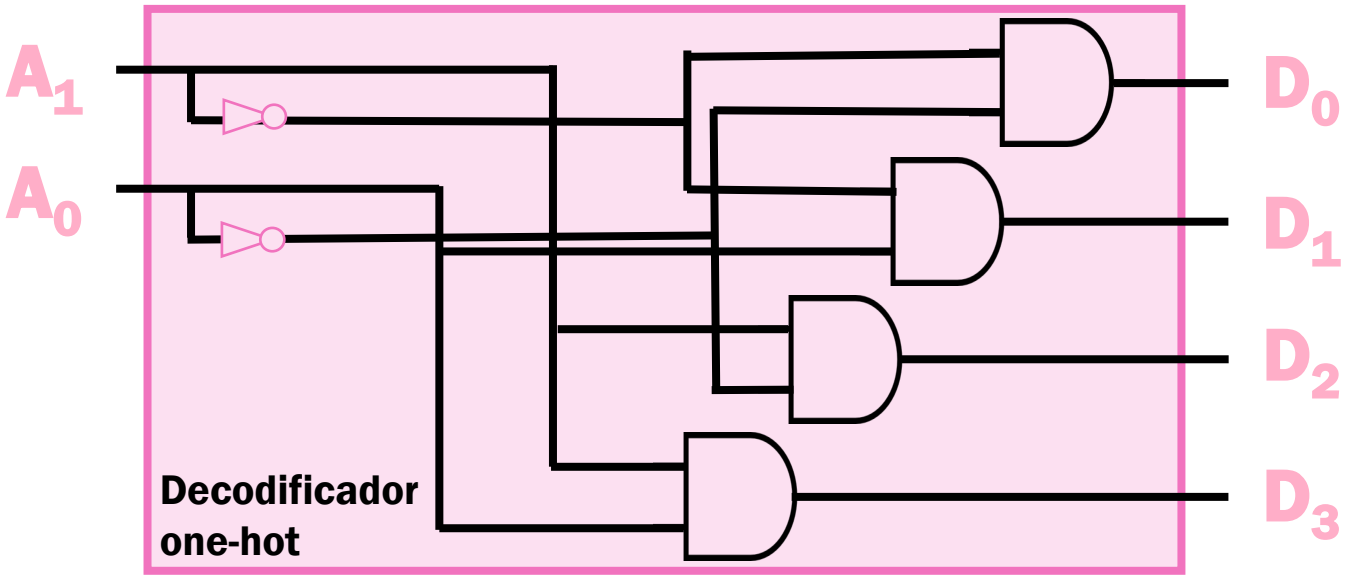
- Si tenemos un selector de n bits, podemos tener 2^n entradas.



Decodificador

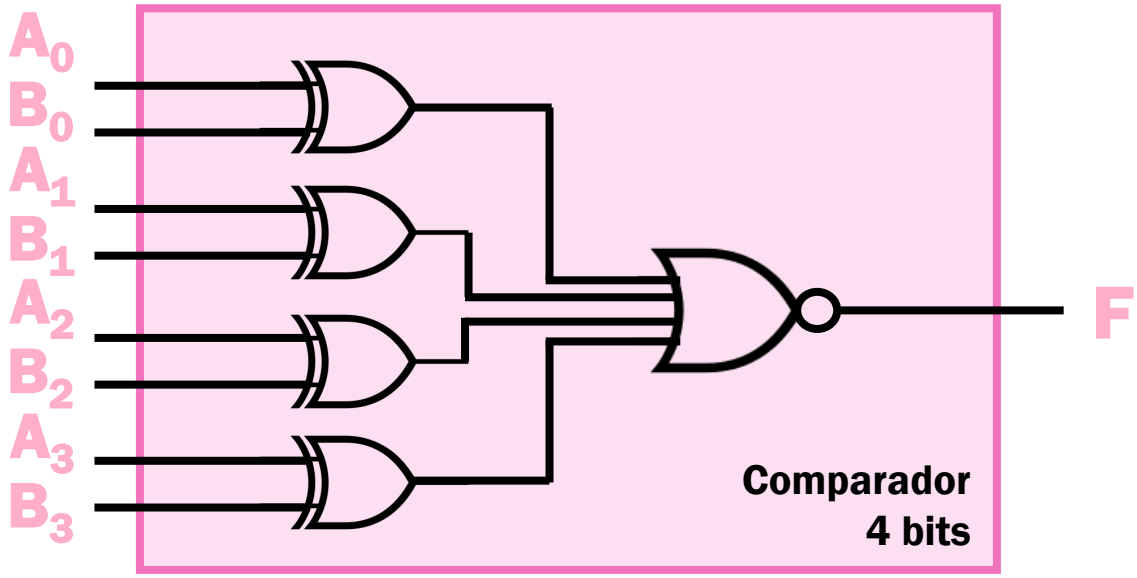
- Ya estamos casi listos para construir una de las componentes más importantes de nuestro computador básico, pero antes necesitamos ver un par de circuitos más.
- Un **decodificador** tiene n bits de inputs y 2^n de output.
- Nos permite "descomprimir" una señal más compacta.
- Por ejemplo, podemos tener un decodificador ***one-hot***. Este solo enciende una de las señales de output a la vez.

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Comparador

- Compara dos patrones de bits del **mismo** largo.
- Si ambos números son iguales bit a bit, entonces el circuito toma el valor de 1, en caso contrario es 0.
- Ejemplo de un circuito comparador de 4 bits



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

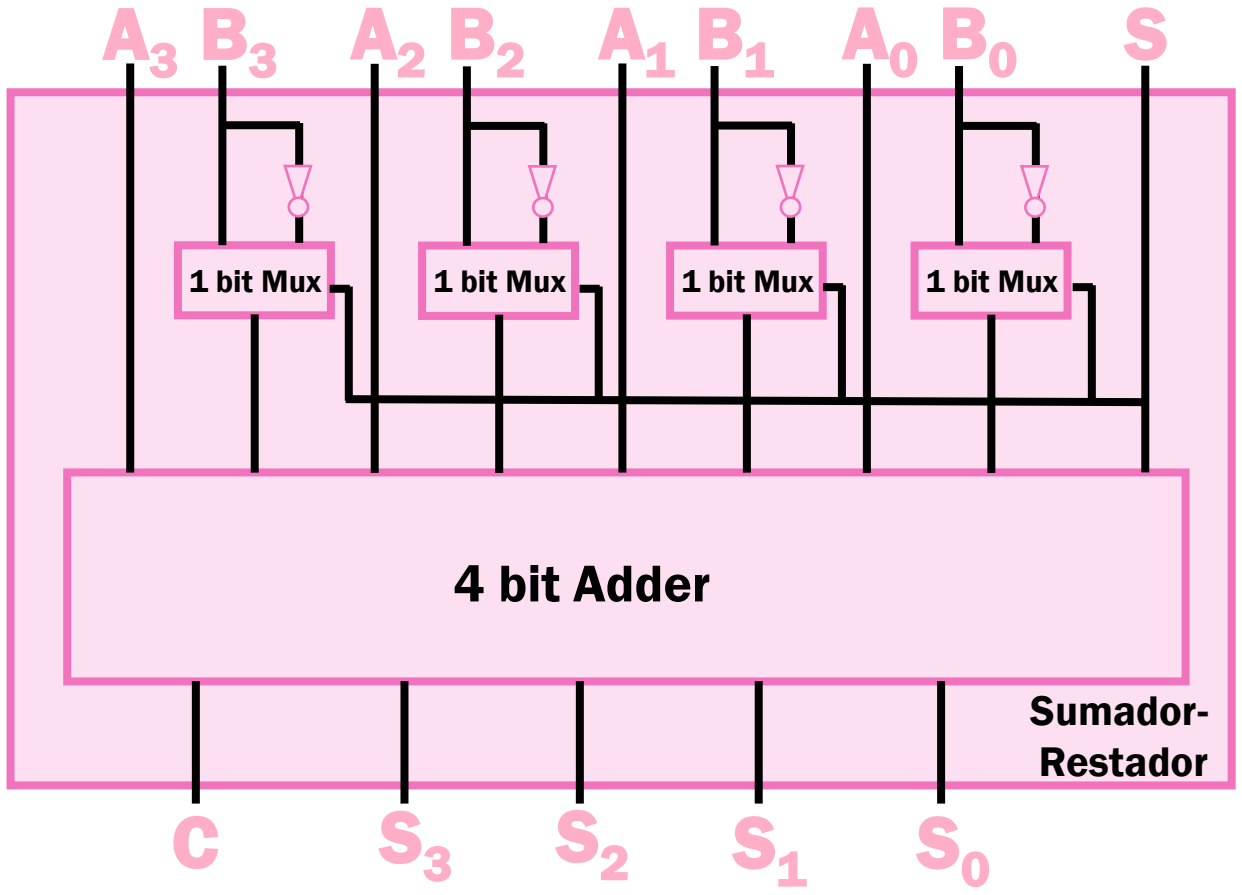
- Hace varias diapos atrás dijimos que queríamos tener un circuito que sea capaz de sumar y restar.
- Sabemos que nuestro circuito sumador lo puede lograr, pero necesitamos una forma de seleccionar el input B sea su complemento de 2 para la resta. Cómo podemos lograr esto? **Con multiplexores!**

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



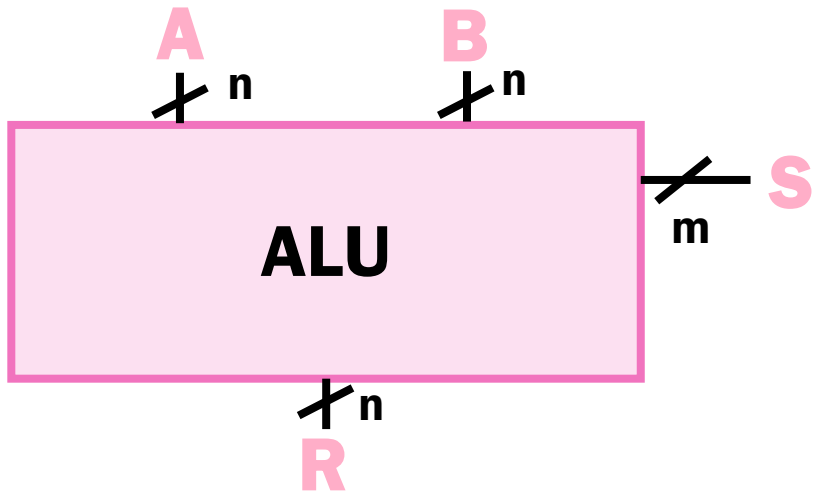
- Usaremos S para indicar si la operación que se quiere realizar es una suma o resta.
  - S = 0 indica suma
  - S = 1 indica resta

- 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.

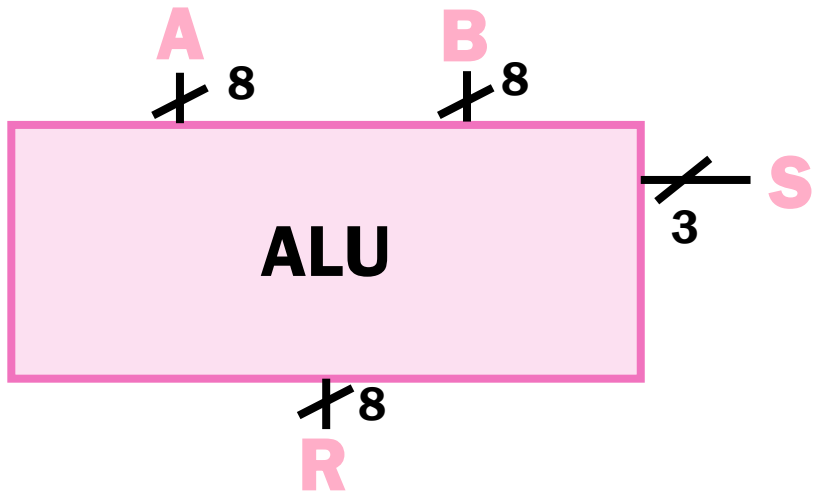


ALU

- 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.

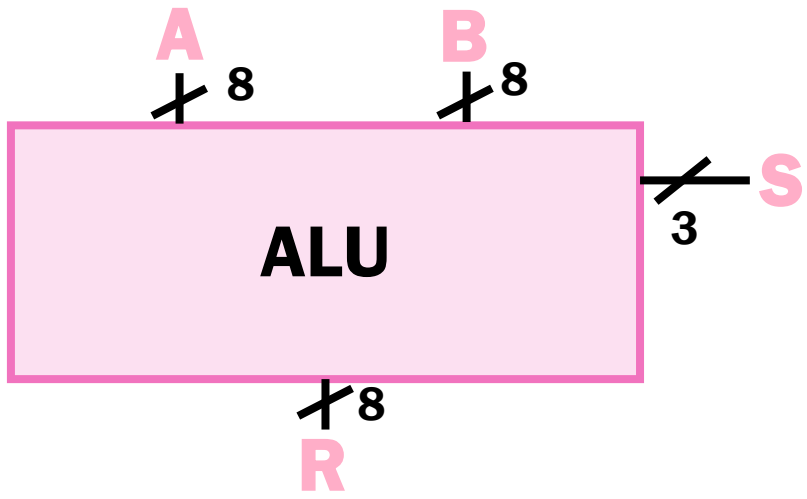
- Los operandos que utiliza: A y B, van a ser de n bits cada uno.
- El resultado R de la operación es también de n bits.
- Tiene una señal de control S de m bits que permite seleccionar entre máximo 2^m operaciones.

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

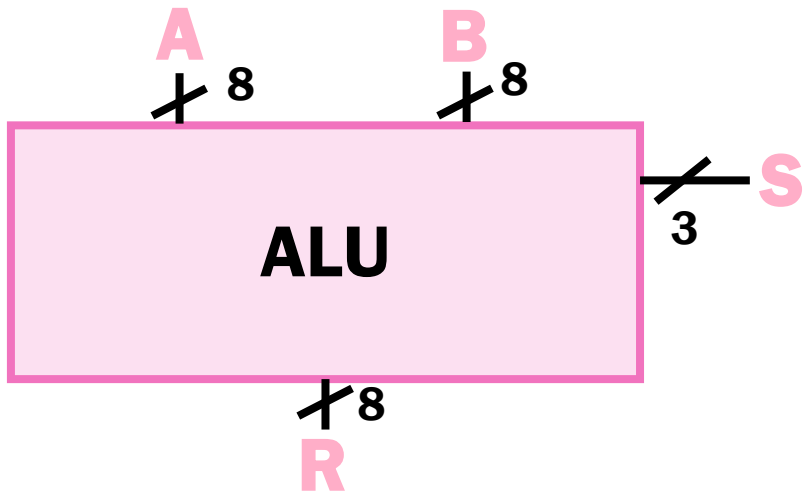
- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?



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

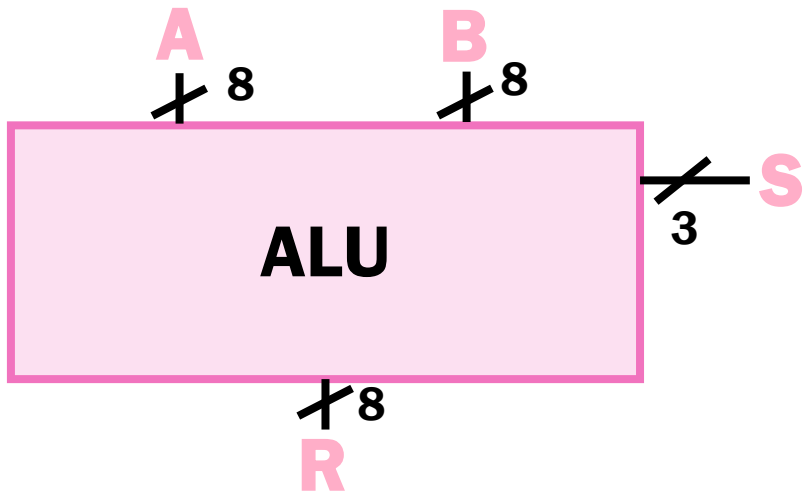


ALU

- 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?

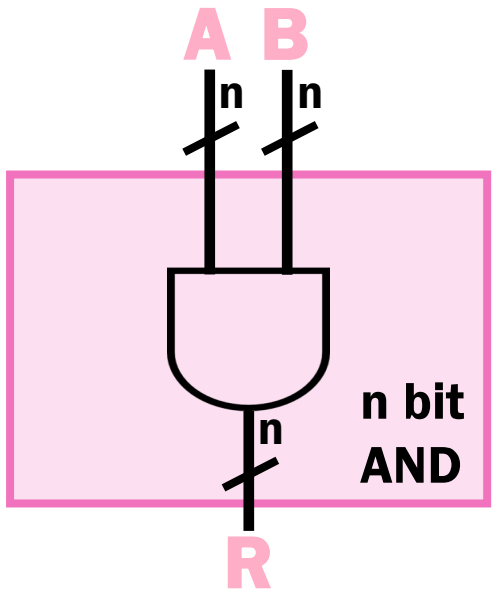
- Vamos a tener:
  - A + B
  - A - B
  - A AND B
  - A OR B
  - NOT A
  - A XOR B

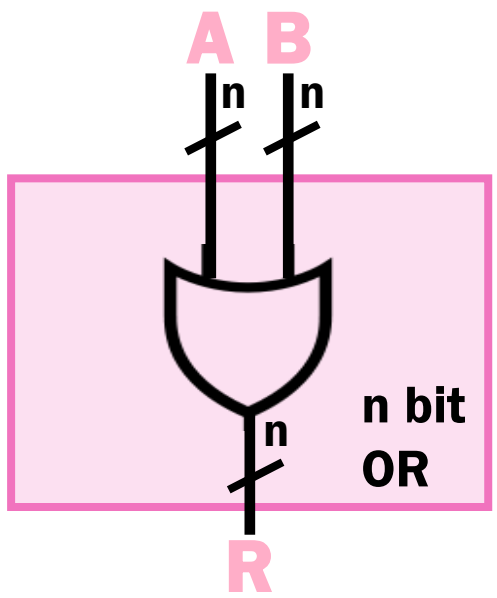
Operaciones lógicas super básicas pero muy útiles para aplicaciones más complejas

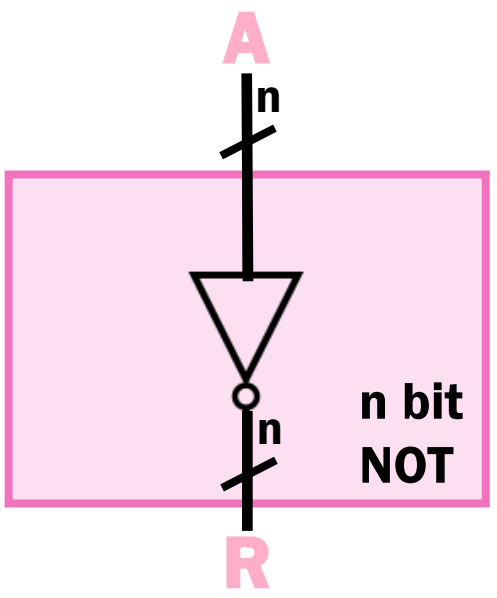


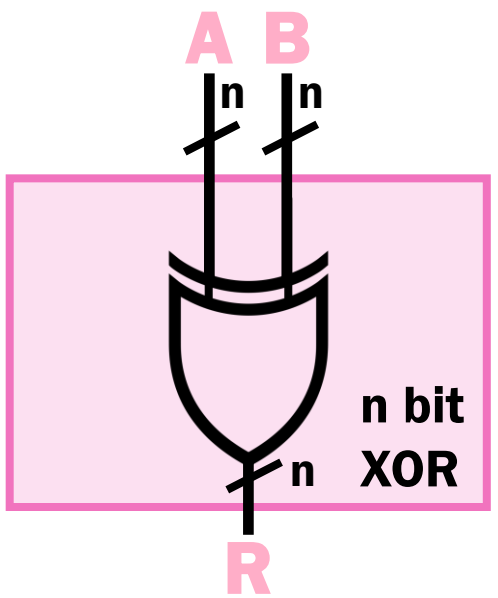
ALU

- 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.







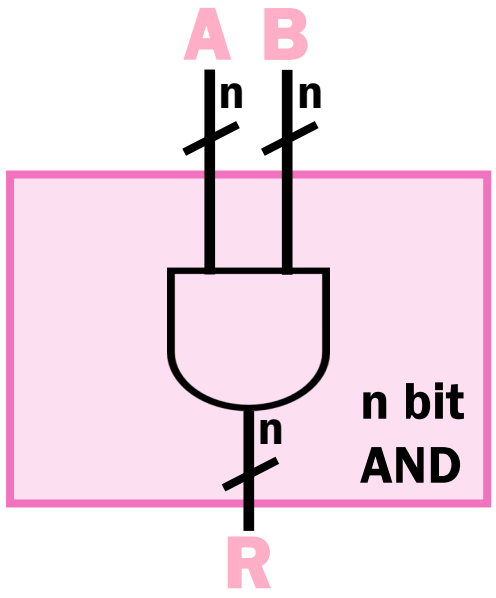


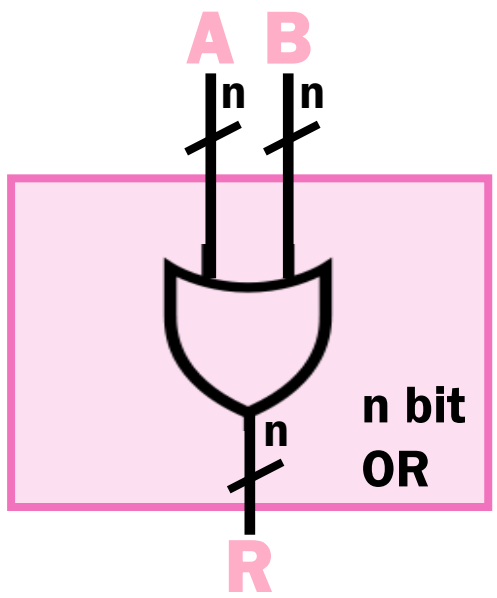
ALU

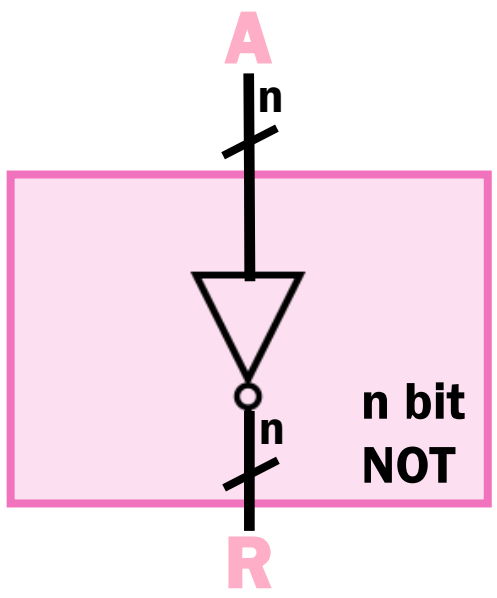
Coincidiremos...

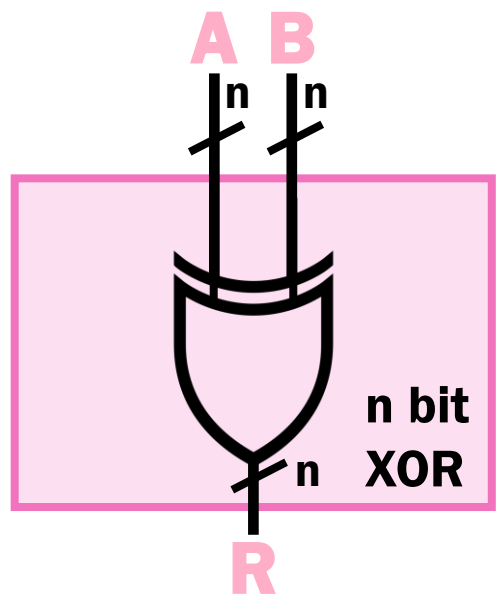
A = 1101
B = 1010

- 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.







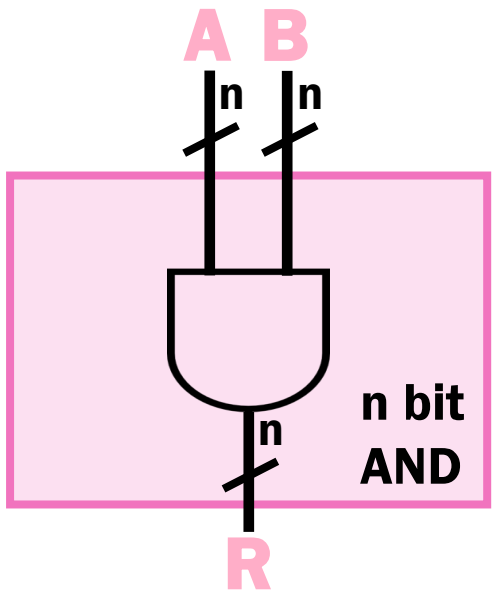


ALU

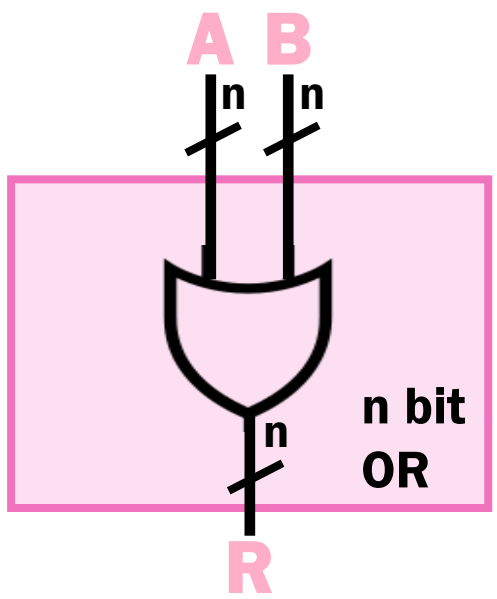
Coincidiremos...

A = 1101
B = 1010

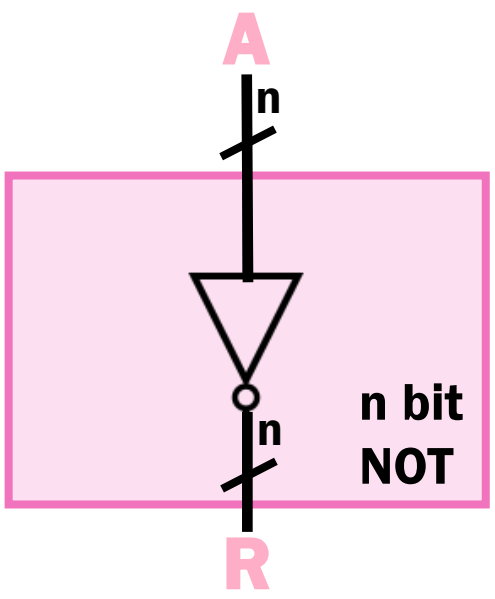
- Así se ven los circuitos de las operaciones lógicas.
- Recordar que cada operación se realiza bit a bit, esta es solo una forma de representarlo para que quede más compacto.



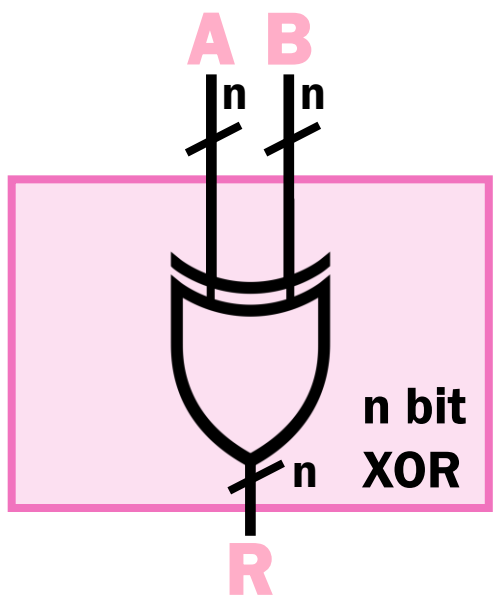
1000



1111



0010

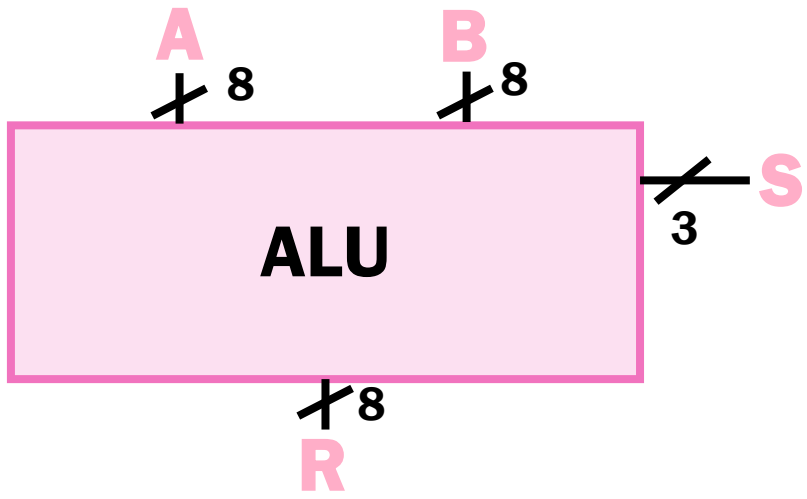


0111

ALU

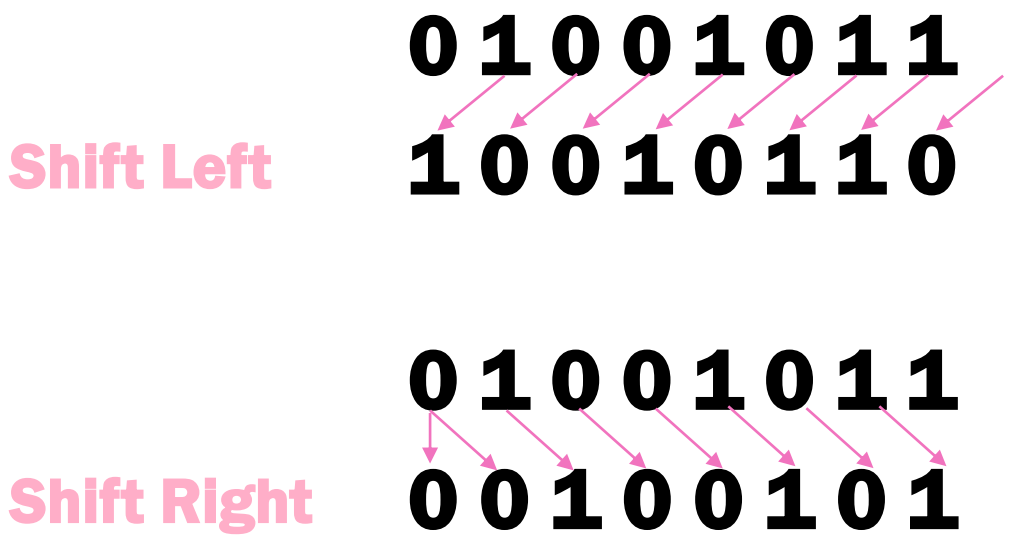
- Entonces si nuestra ALU va a ser capaz de realizar 8 operaciones y sabemos que tenemos la suma y la resta. Cuáles son las otras 6 operaciones?

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



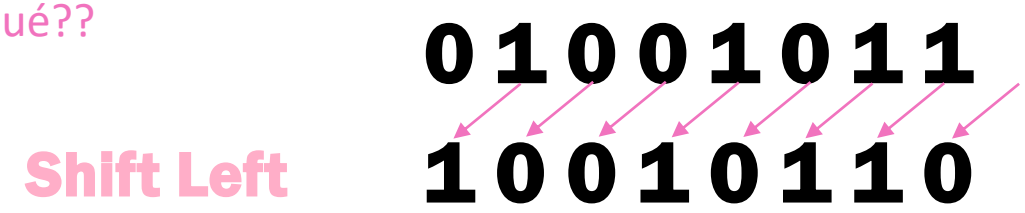
ALU: Shifts

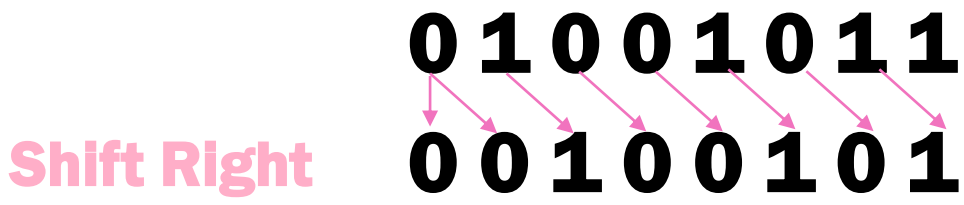
- Los shifts corresponden a desplazar el numero una posición a la izquierda o derecha.
  - El bit que sale se descarta.
  - Quedará una posición vacía:
    - SHL: se rellena con 0
    - SHR: se mantiene el bit



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??





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!

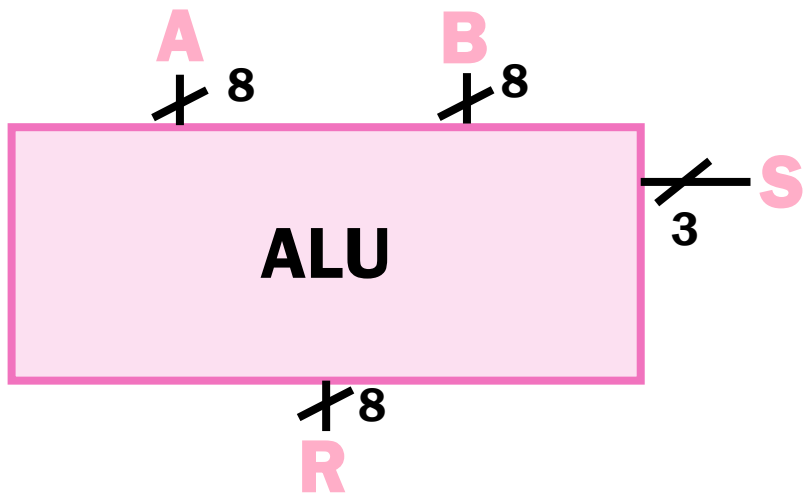
- SHR: Veamos que representa.
  - SHR(0010) = 0001
  - SHR(0101) = 0010

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

DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 2 – Lógica Digital (continuación...)

Susana Figueroa