



DCC
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 1 – Lógica Digital

Susana Figueroa

- Durante el curso vamos a aprender los componentes básicos de un computador, como interaccionan y como programarlos.
- Tenemos que entender entonces como se almacenan los datos en los computadores. Ya dijimos que los computadores solo funcionan con 0s y 1s, entonces si queremos guardar un dato (número entero) en el computador, como debemos hacerlo?

Representación de números enteros: Base 10 (Decimal)

- Creo que todos estamos super familiarizados con esta forma de representar números, pero es probable que no muchos se hayan cuestionado porque es así y que significa realmente.
- La representación en base 10 significa que para representar un numero cualquiera tenemos tan solo 10 símbolos para construirlo, en este caso son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Y también usa una notación posicional donde el valor del número no depende solo del símbolo, sino que también depende de la posición en donde se encuentra.

Por ejemplo, el numero 291 quiere decir el valor que toma la siguiente operación

$$291_{10} = 2 \times 10^2 + 9 \times 10^1 + 1 \times 10^0$$

* Cuando usamos un subíndice en los numero quiere decir que el numero está representado en esa base. No es lo mismo decir 10_{10} que 10_2 . Por eso es importante siempre especificarla, al menos que sea muy evidente por el contexto.

Representación de números enteros: Base 2 (Binario)

- Se comporta de manera similar a la base 10, solamente que ahora contamos con menos símbolos para la representación.
- La representación en base 2 significa que para representar un numero cualquiera tenemos solamente 2 símbolos para construirlo, estos son: 0 y 1.
- También utiliza la notación posicional.

Ahora cómo se representa el numero 291 en base 2?

$$291_{10} = 1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Entonces,

$$291_{10} = 100100011_2$$

Representación de números enteros: Base 2 (Binario)

$$291_{10} = 100100011_2$$

Diagram illustrating the binary representation of the decimal number 291. The binary string is 100100011, with bits labeled from left to right: bit 8 (most significant), bit 7, bit 6, bit 5, bit 4, bit 3, bit 2, bit 1, and bit 0 (least significant). The bit 8 is labeled "(más significativo)" and the bit 0 is labeled "(menos significativo)".

- El bit **menos** significativo corresponde al de más a la derecha.
- El bit **más** significativo es el bit más a la izquierda. En el caso de un número de n bits, es el número que está multiplicado por 2^{n-1} .

Representación de números enteros: Base X

- Ahora que sabemos cómo representar números en base 10 y 2 podemos extender esto a cualquier base.
- En particular, hay dos bases que son las más usadas en computación (a parte de las que ya conocemos):
 - Base 8 (Octal): Base 8 usa los símbolos: 0, 1, 2, ... , 7.
 - Base 16 (Hexadecimal): Base 16 es más interesante, sabemos que tenemos los símbolos 0, 1, ... , 9 pero nos faltan 6 símbolos para representar la base, qué podemos hacer? ... seguimos contando con letras.

Base 10	Base 16
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Base 10	Base 16
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11

Representación de números enteros: Base X

- Es la representación de un número en una base definida única?

Representación de números enteros: Base X

- Es la representación de un número en una base definida única? **Si!!! :)**

Representación de números enteros: Base X

- Es la representación de un número en una base definida única? **Si!!! :)**

Si cambiamos cualquier símbolo de un numero cambia su valor numérico.

Lógica Digital

Ahora entendemos conceptualmente como comunicarnos con el computador a través de los numero binarios, pero como se implementa físicamente?

Vamos a definir a través de la **lógica digital** el *hardware* del computador.

Lógica Digital

- Todos los computadores son realmente múltiples **circuitos digitales** y cada uno de ellos están compuestos por millones de **compuertas lógicas**.
- Una **compuerta lógica** realiza una operación sencilla, pero al unir las entre ellas se pueden lograr operaciones super complejas.

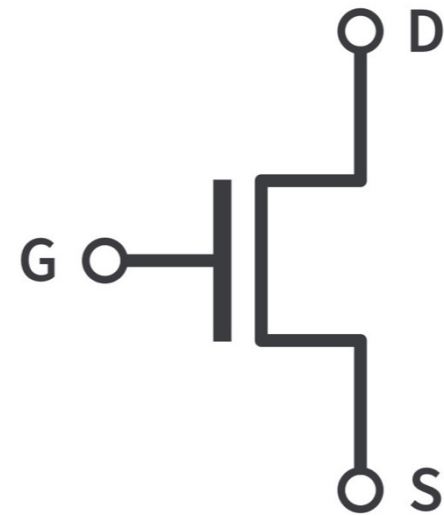
Los valores que existen en estos circuitos son solamente el 0 y 1 lógico (ven cómo se relacionan con el binario? 00)

- Cero lógico: Señal eléctrica de entre 0 a 0.5 volts.
- Uno lógico: Señal eléctrica entre 1 a 1.5 volts.

Las compuertas lógicas toman como entrada una o más de estas señales y su salida corresponde a una señal resultante de una relación sencilla de ellas.

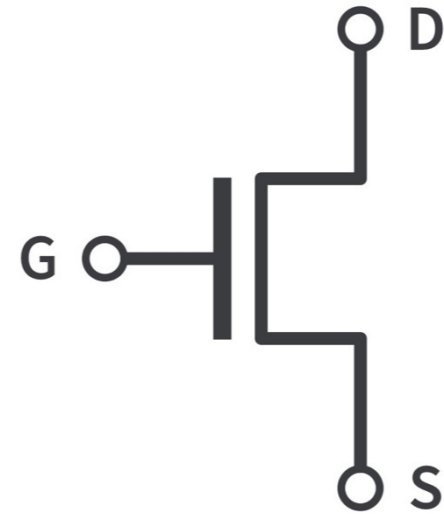
Compuertas Lógicas

- Todas las compuertas lógicas están compuestas por un elemento básico, los **transistores**.
- Qué es un transistor? Es un semiconductor que permite amplificar o bloquear señales eléctricas, pensémoslo como un interruptor.



Compuertas Lógicas

- Todas las compuertas lógicas están compuestas por un elemento básico, los **transistores**.
- Qué es un transistor? Es un semiconductor que permite amplificar o bloquear señales eléctricas, pensémoslo como un interruptor.
- Si en G (gate) se aplica un voltaje mayor a un voltaje crítico, el circuito se cierra (entre D y S) y se comporta como un alambre.
- Si en G se aplica un voltaje menor que el voltaje crítico, el circuito se abre (entre D y S) y se comporta como una resistencia infinita.

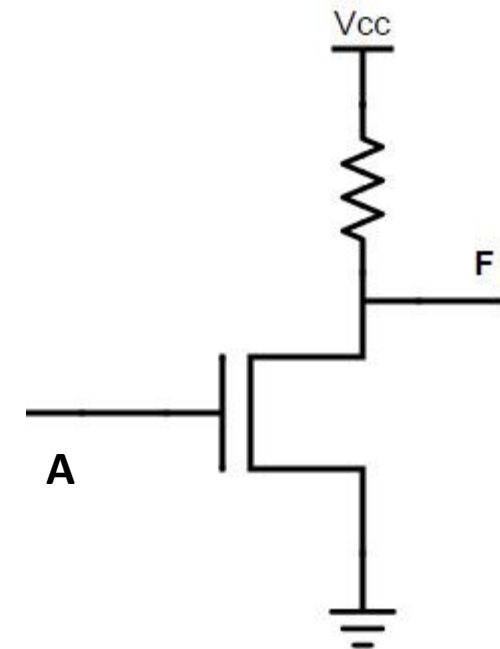


Compuertas Lógicas: NOT

- Tiene solamente un input **A**.
- El output **F**, será el inverso de **A**.

Su tabla de la verdad es la siguiente:

A	NOT A
0	1
1	0



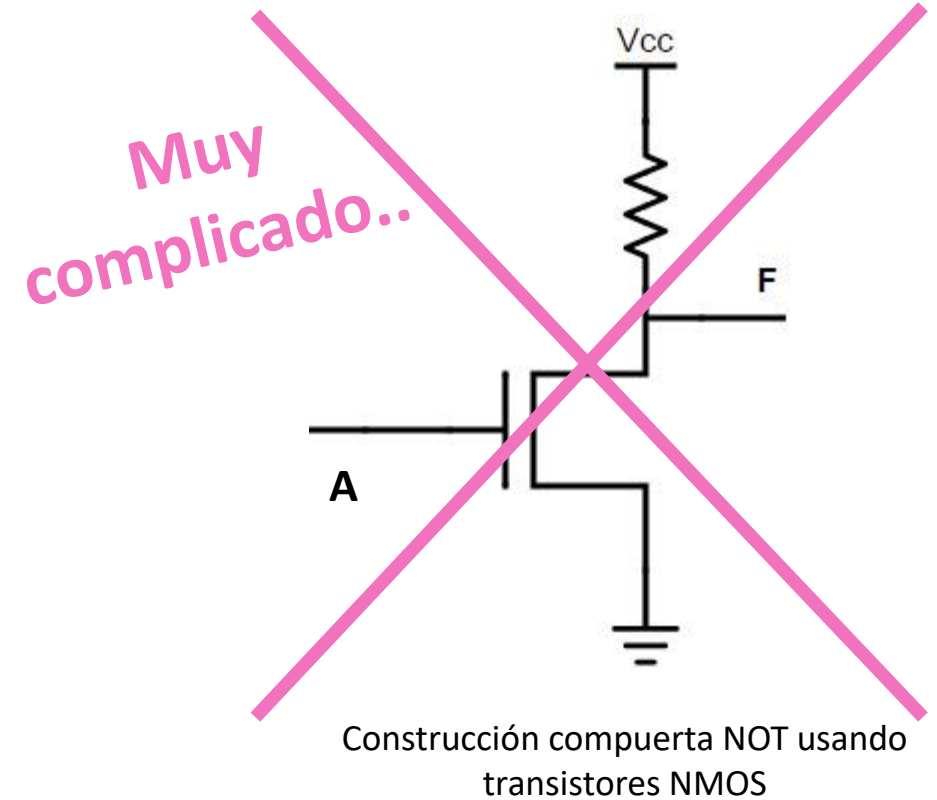
Construcción compuerta NOT usando transistores NMOS

Compuertas Lógicas: NOT

- Tiene solamente un input **A**.
- El output **F**, será el inverso de **A**.

Su tabla de la verdad es la siguiente:

A	NOT A
0	1
1	0



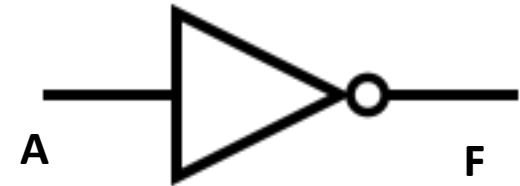
Compuertas Lógicas: NOT

- Tiene solamente un input **A**.
- El output **F**, será el inverso de **A**.

Su tabla de la verdad es la siguiente:

A	NOT A
0	1
1	0

Mucho mejor :)

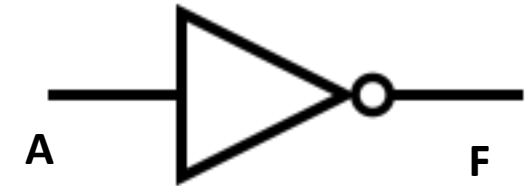


Compuertas Lógicas: NOT

- Tiene solamente un input **A**.
- El output **F**, será el inverso de **A**.

Su tabla de la verdad es la siguiente:

A	NOT A
0	1
1	0



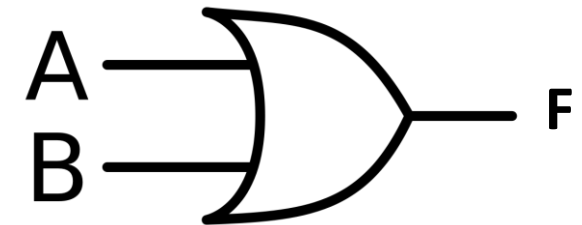
No construiremos el circuito de las compuertas lógicas a partir de transistores porque no nos interesa mucho para este curso.
Usaremos directamente los símbolos.

Compuertas Lógicas: OR

- Tiene dos inputs: **A** y **B**.
- El output **F** será 1, si al menos uno de los inputs es 1.

Su tabla de la verdad es la siguiente:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

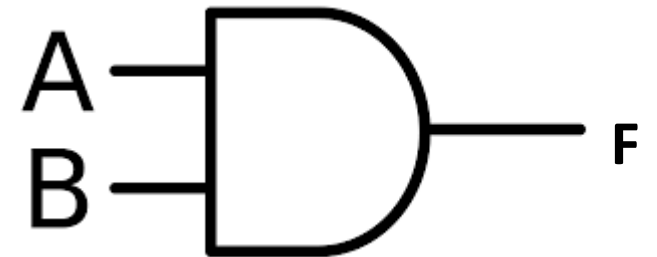


Compuertas Lógicas: AND

- Tiene dos inputs: **A** y **B**.
- El output **F** será 1, si los dos inputs son 1.

Su tabla de la verdad es la siguiente:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

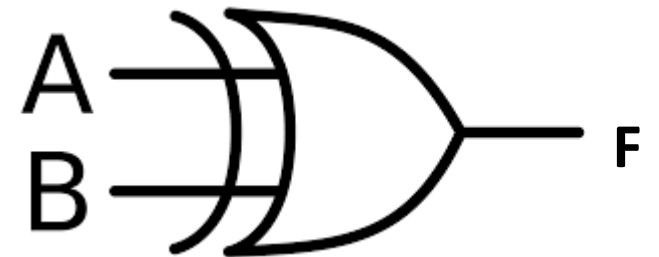


Compuertas Lógicas: XOR

- Tiene dos inputs: **A** y **B**.
- El output **F** será 1, si los inputs son distintos (solo aplicable para compuertas de 2 inputs).
- Corresponde a un **OR** exclusivo.

Su tabla de la verdad es la siguiente:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

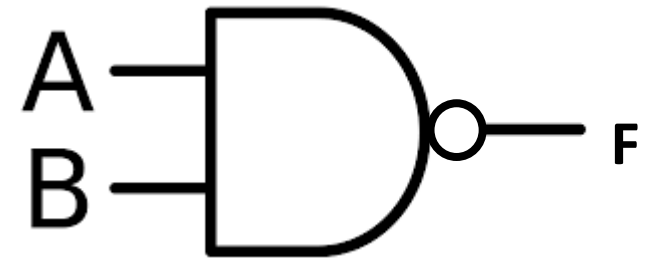


Compuertas Lógicas: NAND

- Tiene dos inputs: **A** y **B**.
- El output **F** será 0, si los dos inputs son 1.

Su tabla de la verdad es la siguiente:

A	B	A AND B
0	0	1
0	1	1
1	0	1
1	1	0

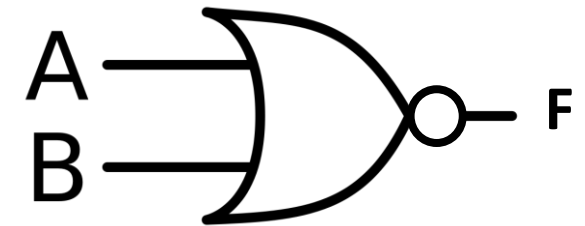


Compuertas Lógicas: NOR

- Tiene dos inputs: **A** y **B**.
- El output **F** será 1, si los dos inputs son 0.

Su tabla de la verdad es la siguiente:

A	B	A OR B
0	0	1
0	1	0
1	0	0
1	1	0

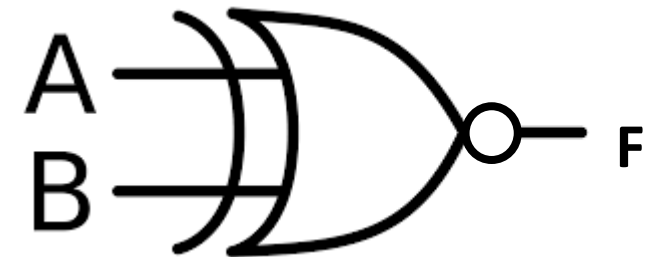


Compuertas Lógicas: XNOR

- Tiene dos inputs: **A** y **B**.
- El output **F** será 1, si los inputs son iguales (solo aplicable para compuertas de 2 inputs).

Su tabla de la verdad es la siguiente:

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1



Compuertas Lógicas: Algebra Booleana

- Para describir los circuitos lógicos utilizando las compuertas mencionadas anteriormente se utilizará algebra de Boole.
- Esto nos permitirá reducir expresiones, que en la práctica equivale a minimizar los circuitos que se implementaran.
- Las leyes básicas son las siguientes:

- **Identidad:** $A + 0 = A$ y $A \cdot 1 = A$
- **Dominación:** $A + 1 = 1$ y $A \cdot 0 = 0$
- **Inverso:** $A + \bar{A} = 1$ y $A \cdot \bar{A} = 0$
- **Conmutatividad:** $A + B = B + A$ y $A \cdot B = B \cdot A$
- **Asociatividad:** $A + (B + C) = (A + B) + C$ y $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- **Distributiva:** $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ y $A + (B \cdot C) = (A + B) \cdot (A + C)$
- **De Morgan:** $\overline{A \cdot B} = \bar{A} + \bar{B}$ y $\overline{A + B} = \bar{A} \cdot \bar{B}$

Compuertas Lógicas: Algebra Booleana

- Para describir los circuitos lógicos utilizando las compuertas mencionadas anteriormente se utilizará algebra de Boole.
- Esto nos permitirá reducir expresiones, que en la práctica equivale a minimizar los circuitos que se implementaran.
- Las leyes básicas son las siguientes:

- **Identidad:** $A + 0 = A$ y $A \cdot 1 = A$
- **Dominación:** $A + 1 = 1$ y $A \cdot 0 = 0$
- **Inverso:** $A + \bar{A} = 1$ y $A \cdot \bar{A} = 0$
- **Conmutatividad:** $A + B = B + A$ y $A \cdot B = B \cdot A$
- **Asociatividad:** $A + (B + C) = (A + B) + C$ y $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- **Distributiva:** $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ y $A + (B \cdot C) = (A + B) \cdot (A + C)$
- **De Morgan:** $\overline{A \cdot B} = \bar{A} + \bar{B}$ y $\overline{A + B} = \bar{A} \cdot \bar{B}$

Ojo: Acá las sumas corresponden a compuertas OR (\vee) y las multiplicaciones a compuertas AND (\wedge)

Circuito Sumador

- A partir de las compuertas lógicas que acabamos de estudiar construiremos paso a paso un circuito que sume dos números binarios de 4 bits.
- Pero primero... cómo se suman los números binarios?

Circuito Sumador

- A partir de las compuertas lógicas que acabamos de estudiar construiremos paso a paso un circuito que sume dos números binarios de 4 bits.
- Pero primero... cómo se suman los números binarios?

Igual que los números decimales!!
Yaayyyyy!

Circuito Sumador

- A partir de las compuertas lógicas que acabamos de estudiar construiremos paso a paso un circuito que sume dos números binarios de 4 bits.
- Pero primero... cómo se suman los números binarios?
- Tenemos el mismo concepto de reserva o **carry**.

Veamos la suma de dos números de 1 bit:

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Para la suma: Que compuerta lógica nos sirve para obtener ese output?

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Para la suma: Que compuerta lógica nos sirve para obtener ese output?

XOR!! Si se fijan, esta compuerta tiene la misma tabla de verdad

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Para la suma: Qué compuerta lógica nos sirve para obtener ese output? **XOR**
- Para la reserva: Qué compuerta lógica nos sirve para obtener ese output?

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Para la suma: Qué compuerta lógica nos sirve para obtener ese output? **XOR**
- Para la reserva: Qué compuerta lógica nos sirve para obtener ese output? **AND**

Circuito Sumador: Half Adder

- El circuito que suma dos números de 1 bit lo llamaremos **half adder**.
- Para construir el circuito debemos mirar la tabla de verdad.
- Lo dividiremos en dos circuitos: circuito de suma y de carry.

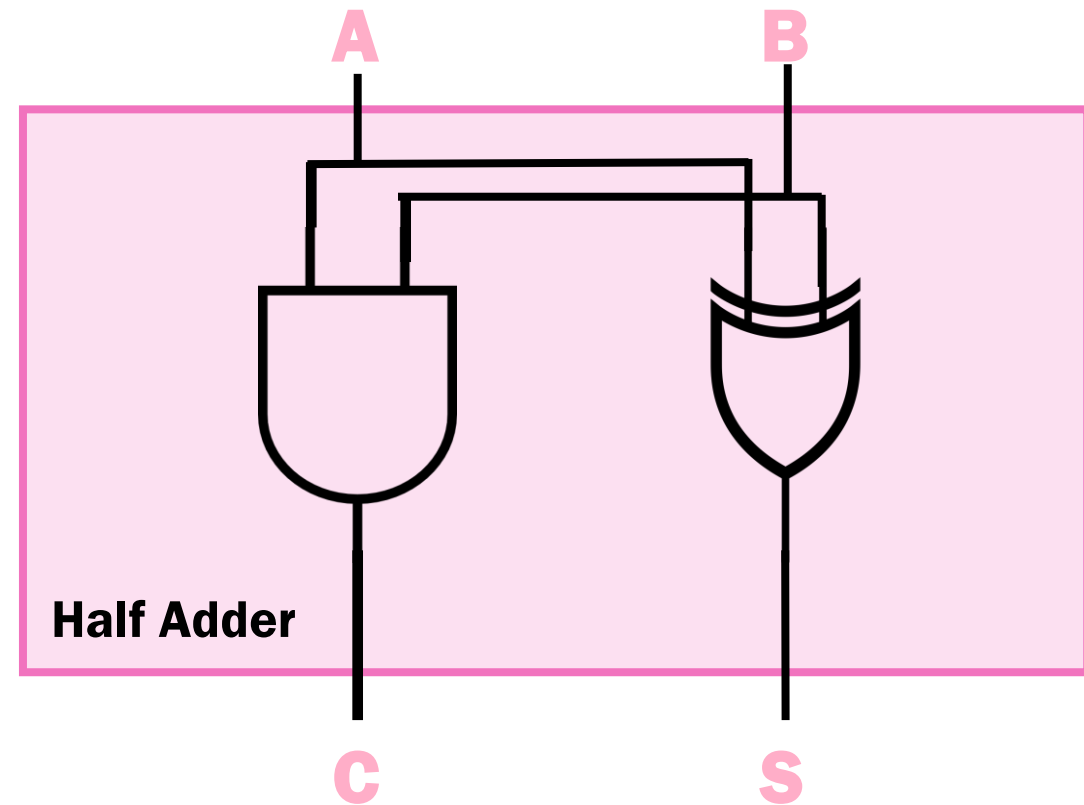
A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Para la suma: Qué compuerta lógica nos sirve para obtener ese output? **XOR**
- Para la reserva: Qué compuerta lógica nos sirve para obtener ese output? **AND**

**Ya tenemos todas las componentes
para hacer el circuito half adder!**

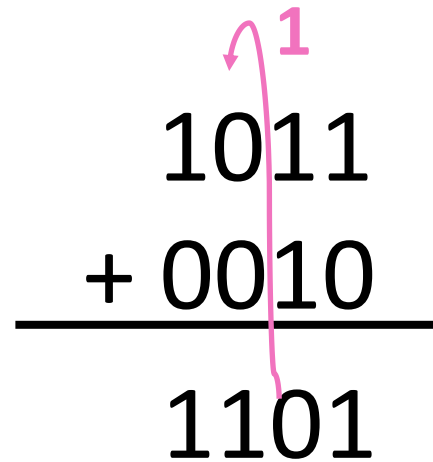
Circuito Sumador: Half Adder

A	B	Carry	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Circuito Sumador

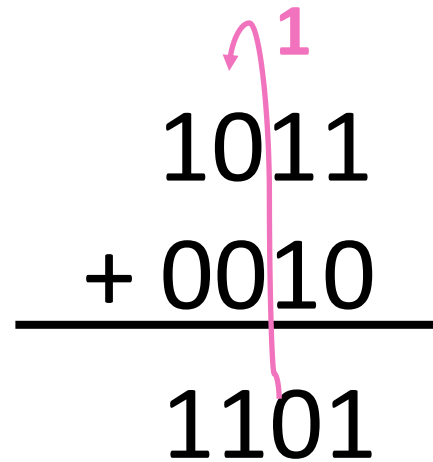
- Ahora sumemos números de 4 bits.
- Sumamos bit a bit y propagamos el carry como lo hacemos en decimal.

$$\begin{array}{r} 1011 \\ + 0010 \\ \hline 1101 \end{array}$$


Circuito Sumador

- Ahora sumemos números de 4 bits.
- Sumamos bit a bit y propagamos el carry como lo hacemos en decimal.

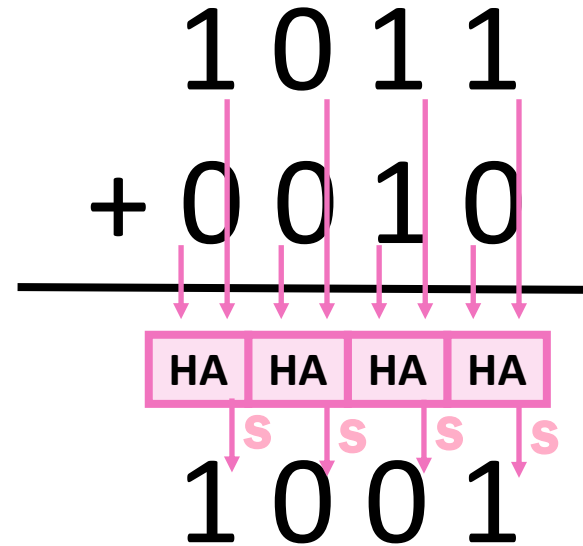
Si se fijan, esta suma bit a bit es como poner un half adder entre los números por cada posición.

$$\begin{array}{r} 1011 \\ + 0010 \\ \hline 1101 \end{array}$$


Circuito Sumador

- Ahora sumemos números de 4 bits.
- Sumamos bit a bit y propagamos el carry como lo hacemos en decimal.

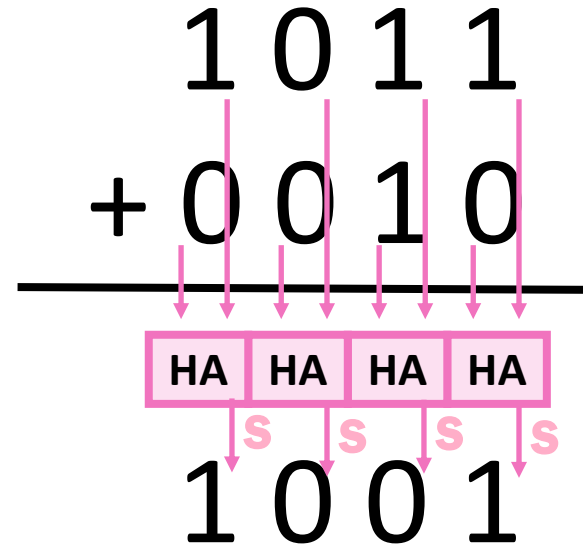
Si se fijan, esta suma bit a bit es como poner un half adder entre los números por cada posición.



Circuito Sumador

- Ahora sumemos números de 4 bits.
- Sumamos bit a bit y propagamos el carry como lo hacemos en decimal.

Si se fijan, esta suma bit a bit es como poner un half adder entre los números por cada posición.

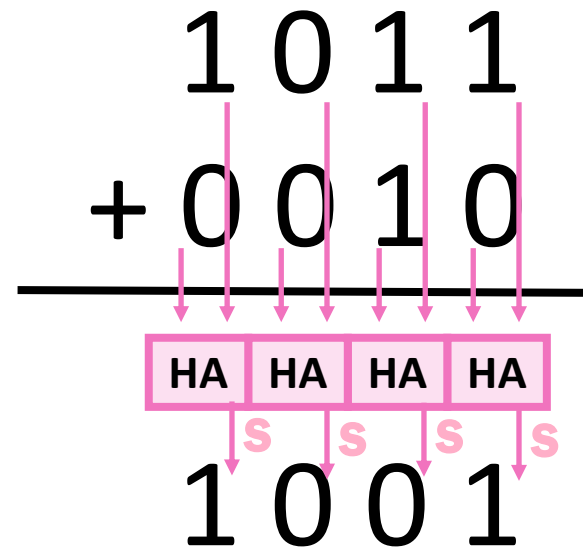


Qué problema tenemos??

Circuito Sumador

- Ahora sumemos números de 4 bits.
- Sumamos bit a bit y propagamos el carry como lo hacemos en decimal.

Si se fijan, esta suma bit a bit es como poner un half adder entre los números por cada posición.



Qué problema tenemos??
No temos como incorporar el carry anterior a nuestro circuito! :(

Circuito Sumador: Full Adder

- El circuito que suma dos números de 1 bit, más el carry, lo llamaremos **full adder**.
- Este circuito nos va a permitir propagar el carry de la suma de la forma correcta.

A	B	Carry In	Carry	Suma
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Circuito Sumador: Full Adder

- El circuito que suma dos números de 1 bit, más el carry, lo llamaremos **full adder**.
- Este circuito nos va a permitir propagar el carry de la suma de la forma correcta.

A	B	Carry In	Carry	Suma
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Cómo hacemos ahora el circuito??

Circuito Sumador: Full Adder

- El circuito que suma dos números de 1 bit, más el carry, lo llamaremos **full adder**.
- Este circuito nos va a permitir propagar el carry de la suma de la forma correcta.

A	B	Carry In	Carry	Suma
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

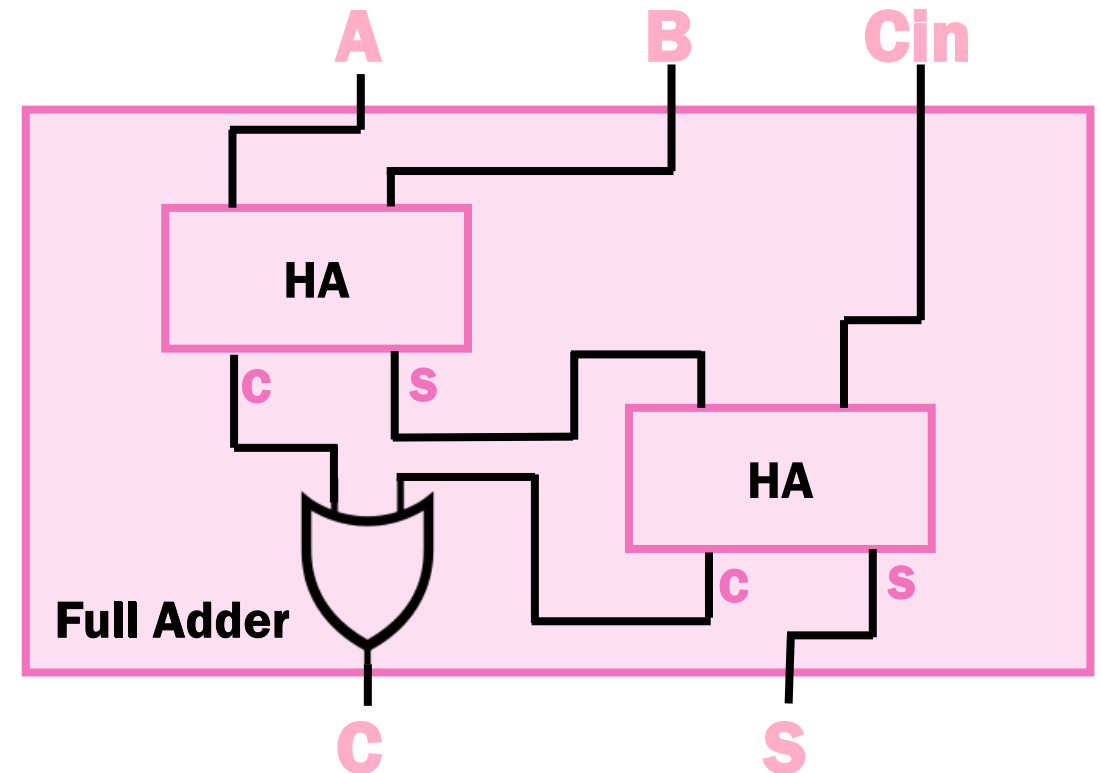
Cómo hacemos ahora el circuito??

En vez de partir desde cero como lo hicimos con el half adder, usaremos HA's para construirlo!

Circuito Sumador: Full Adder

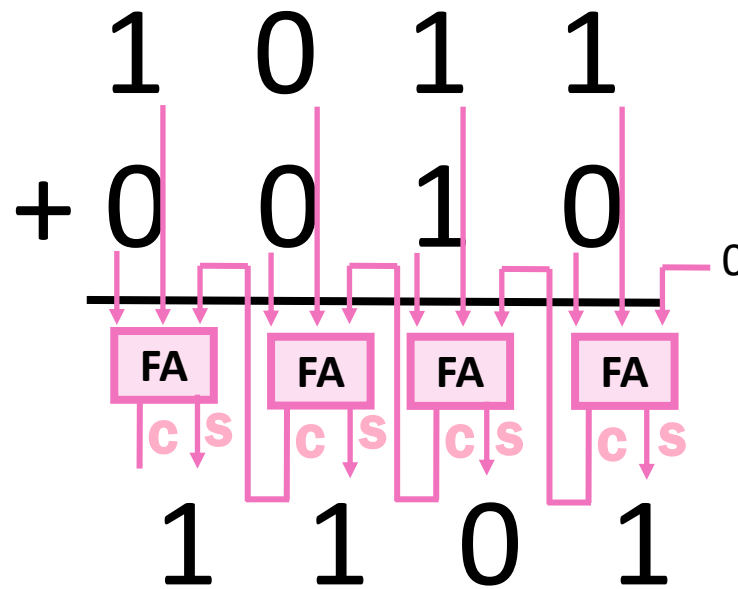
- El circuito que suma dos números de 1 bit, más el carry, lo llamaremos **full adder**.
- Este circuito nos va a permitir propagar el carry de la suma de la forma correcta.

A	B	Carry In	Carry	Suma
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1



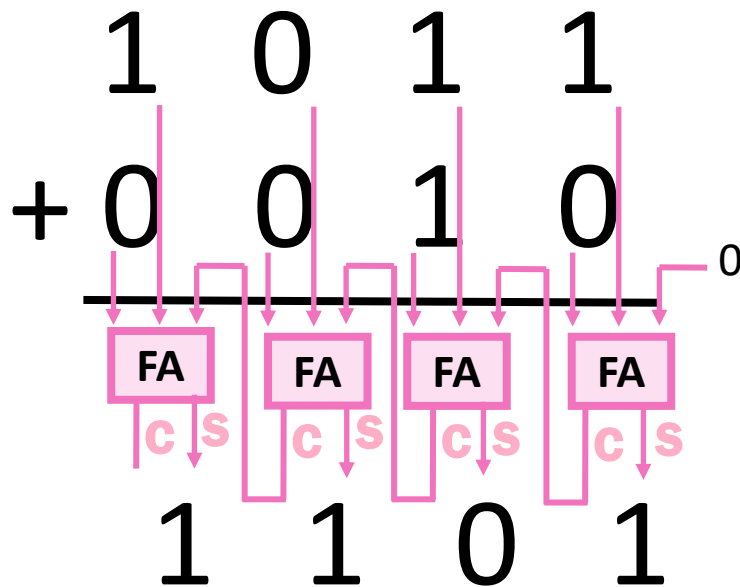
Circuito Sumador

- Ahora si podemos sumar correctamente!!!



Circuito Sumador

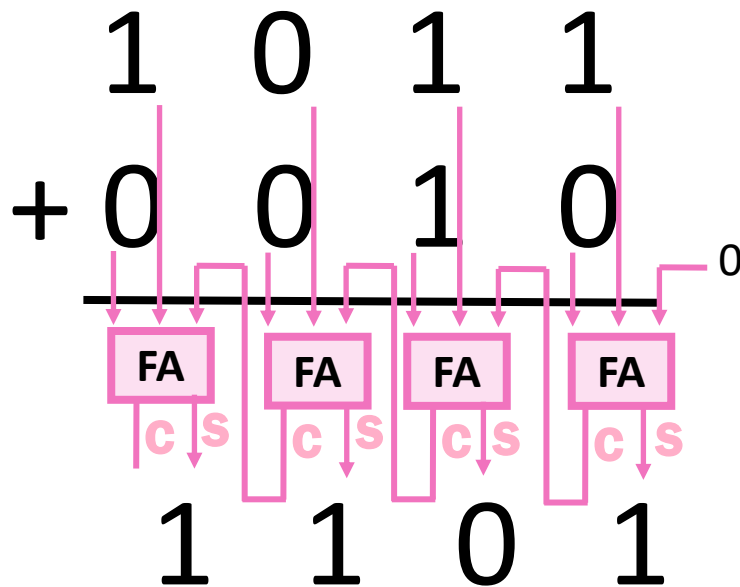
- Ahora si podemos sumar correctamente!!!



Cosas buenas de este circuito:

Circuito Sumador

- Ahora si podemos sumar correctamente!!!



Cosas buenas de este circuito:

- Es muy fácil de extender a un número arbitrario de bits, solo se requiere tener tantos full adders como cantidad de bits de los números a sumar.
- Es suuuper rápido para el computador sumar números en binario y no depende del largo de los números.

Circuito Sumador: General

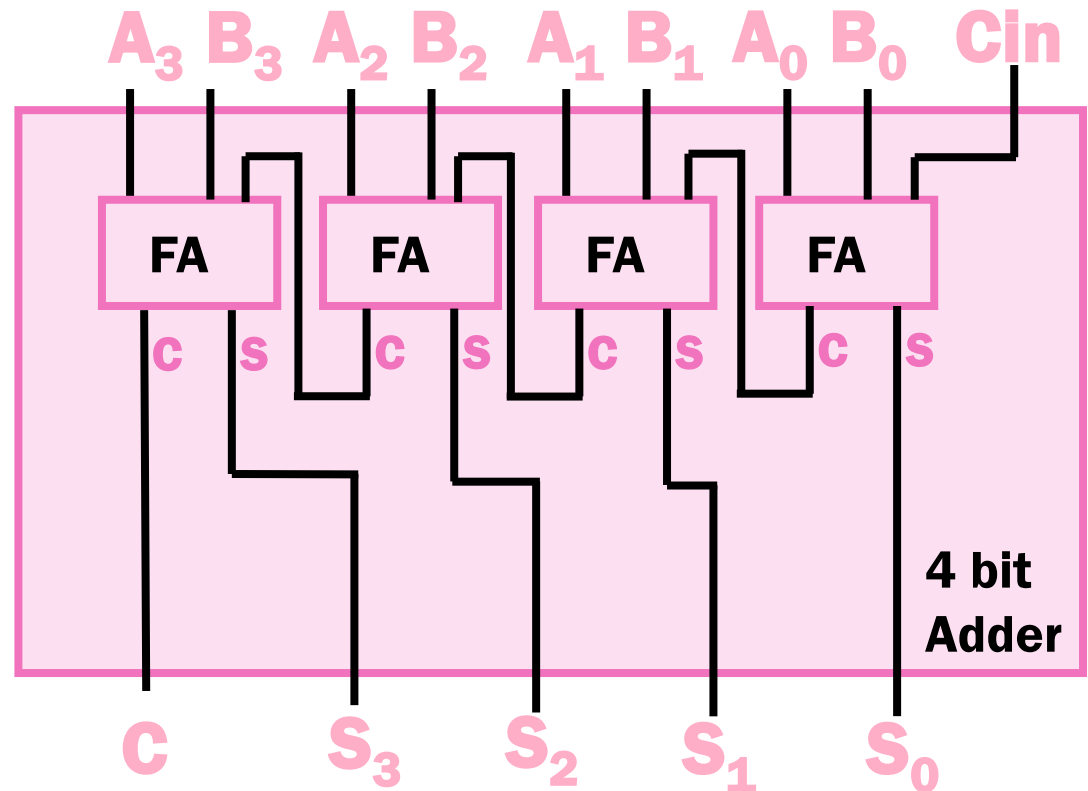
- Forma general: Para un circuito sumador de $n+1$ bits:

$$A_n \dots A_1 A_0 + B_n \dots B_1 B_0$$

Se necesitan $n+1$ **full adders**, conectados como el diagrama a la derecha.

El C_{out} del full adder para el i -ésimo bit se conecta al C_{in} del full adder para el $(i+1)$ -ésimo bit.

El C_{in} del full adder para el bit 0, es 0.



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:

$$\begin{array}{lcl} 3_{10} & \xrightarrow{\text{Base 2}} & 11_2 \xrightarrow{S/M} 011_2 \\ -3_{10} & \xrightarrow{\text{Base 2}} & -11_2 \xrightarrow{S/M} 111_2 \end{array}$$

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:

$$\begin{array}{lcl} 3_{10} & \xrightarrow{\text{Base 2}} & 11_2 \xrightarrow{S/M} 011_2 \\ -3_{10} & \xrightarrow{\text{Base 2}} & -11_2 \xrightarrow{S/M} 111_2 \end{array}$$

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

$$\begin{aligned} 0101 + (1101) \\ = 10010 \end{aligned}$$

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

$$\begin{aligned} 0101 + (1101) \\ = 10010 \end{aligned}$$

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

$$\begin{aligned} 0101 + (1101) \\ = 10010 \end{aligned}$$

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,

$$\begin{aligned} & \bullet \quad 5_{10} \xrightarrow{\text{Base 2}} 101_2 \xrightarrow{C_1} 0101_2 \\ & \bullet \quad -5_{10} \xrightarrow{\text{Base 2}} -101_2 \xrightarrow{C_1} 1010_2 \end{aligned}$$

- 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.

$$\begin{aligned} 0101 + (1010) \\ = 1111 \end{aligned}$$

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,

$$\begin{aligned} & \bullet \quad 5_{10} \xrightarrow{\text{Base 2}} 101_2 \xrightarrow{C_2} 0101_2 \\ & \bullet \quad -5_{10} \xrightarrow{\text{Base 2}} -101_2 \xrightarrow{C_2} 1010_2 + 1_2 = 1011_2 \end{aligned}$$

- 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.

$$\begin{aligned}0101 + (1011) \\ = 10000\end{aligned}$$

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.

$$\begin{aligned}0101 + (1011) \\ = 10000\end{aligned}$$

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

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.

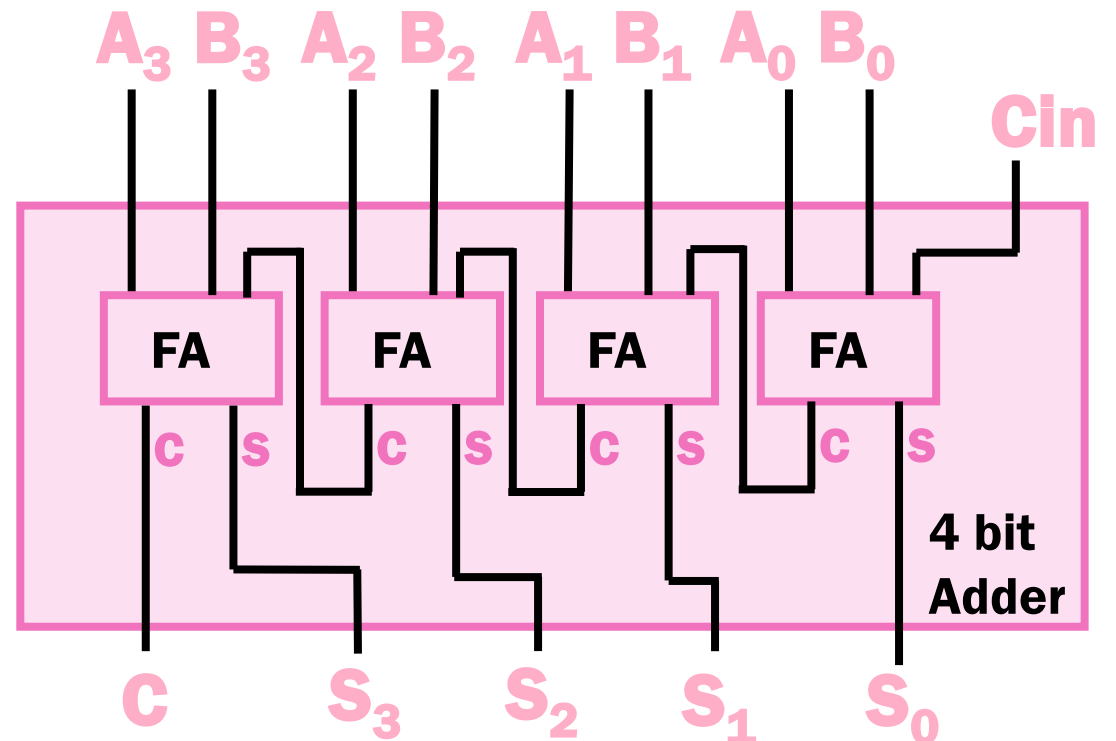
$$\begin{aligned} 0101 + (1011) \\ = 10000 \end{aligned}$$

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

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).

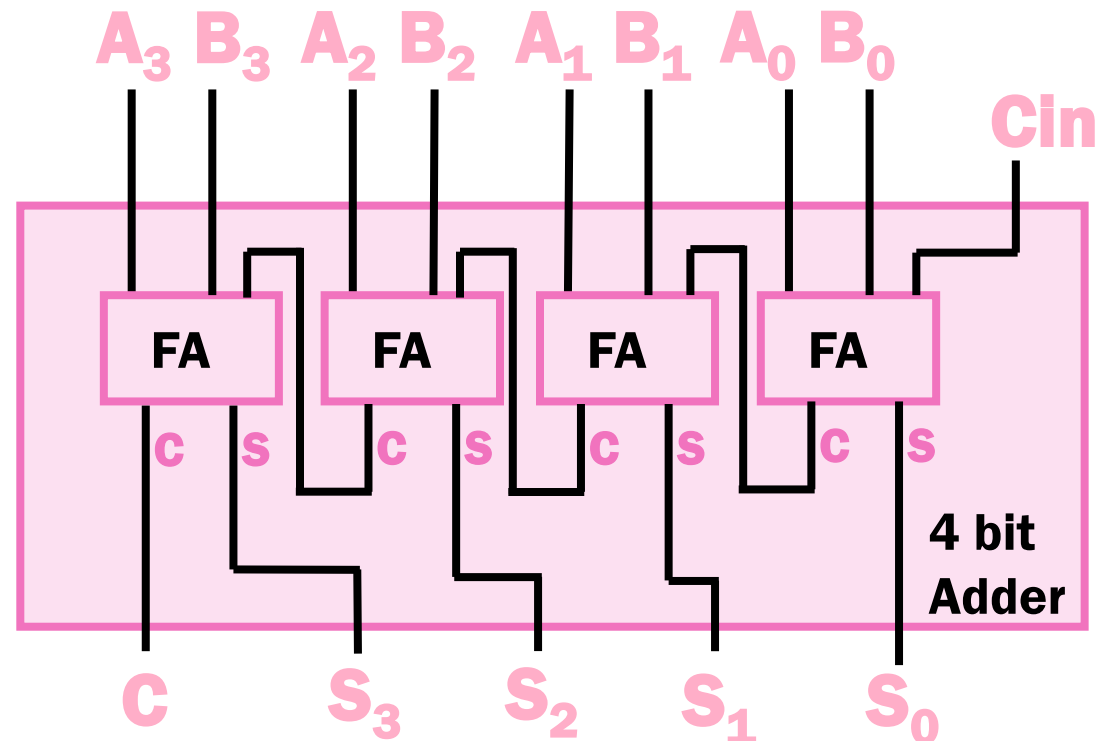
Circuito Restador

- 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.



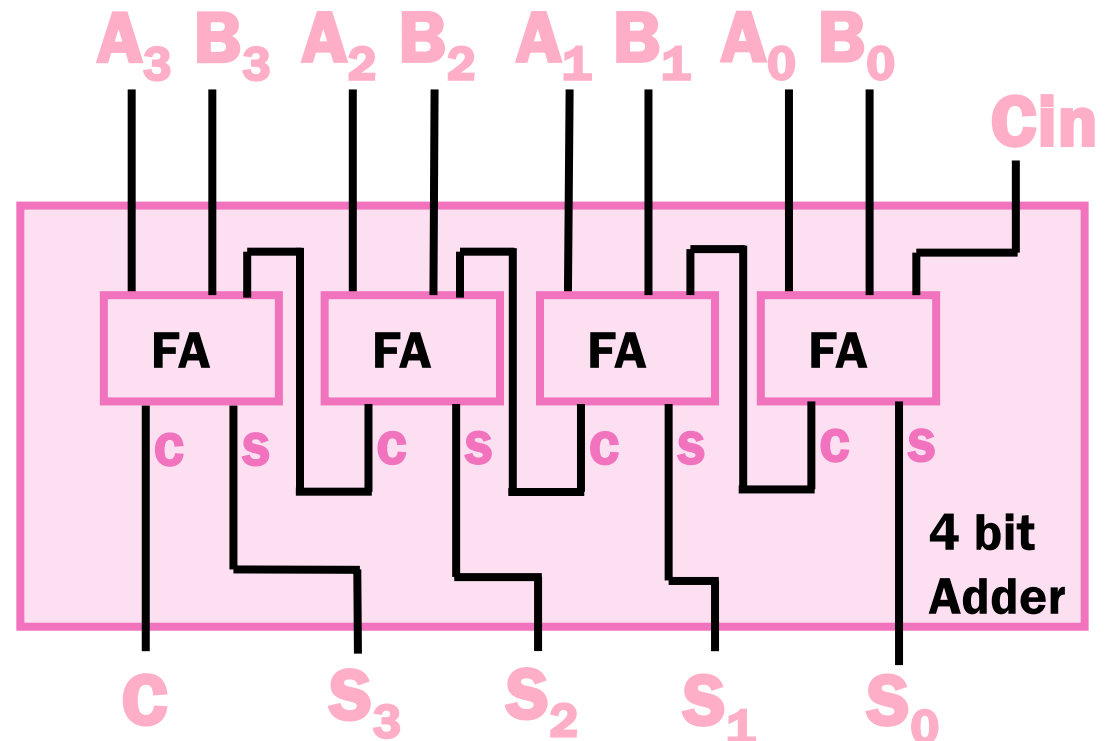
Circuito Restador

- 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

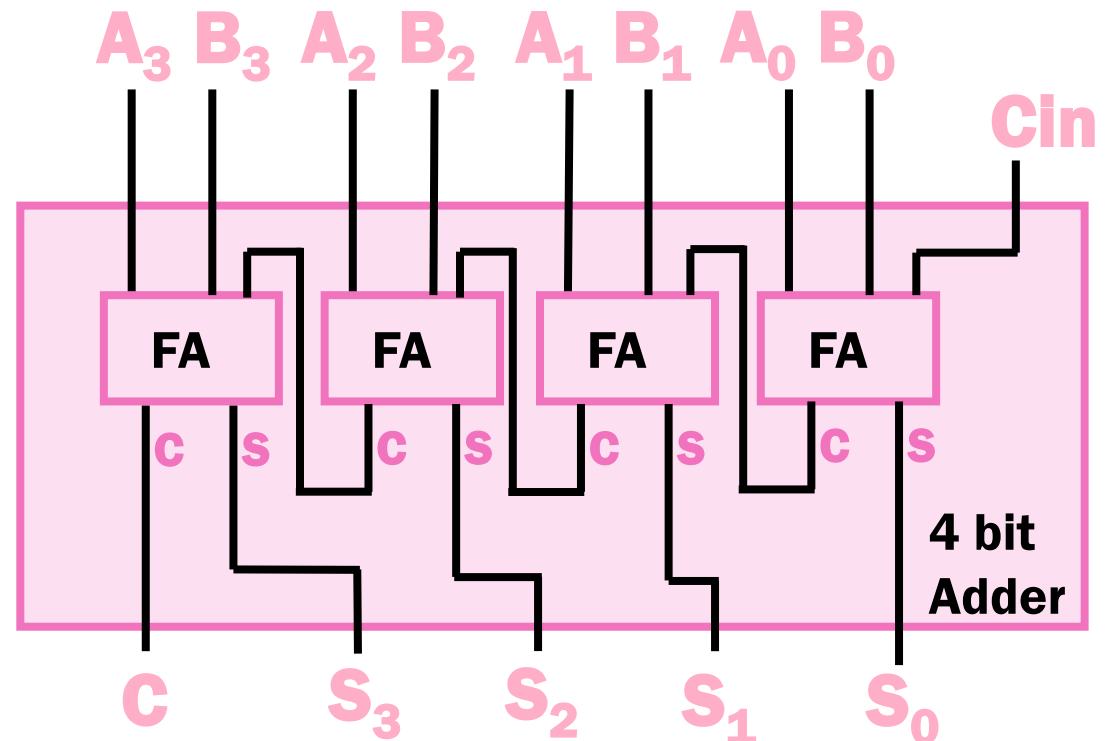
- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
 - $A - B = A + (-B)$
- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.
- Cuál modificación? Ya 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,
 - $A - B = A + (-B)$

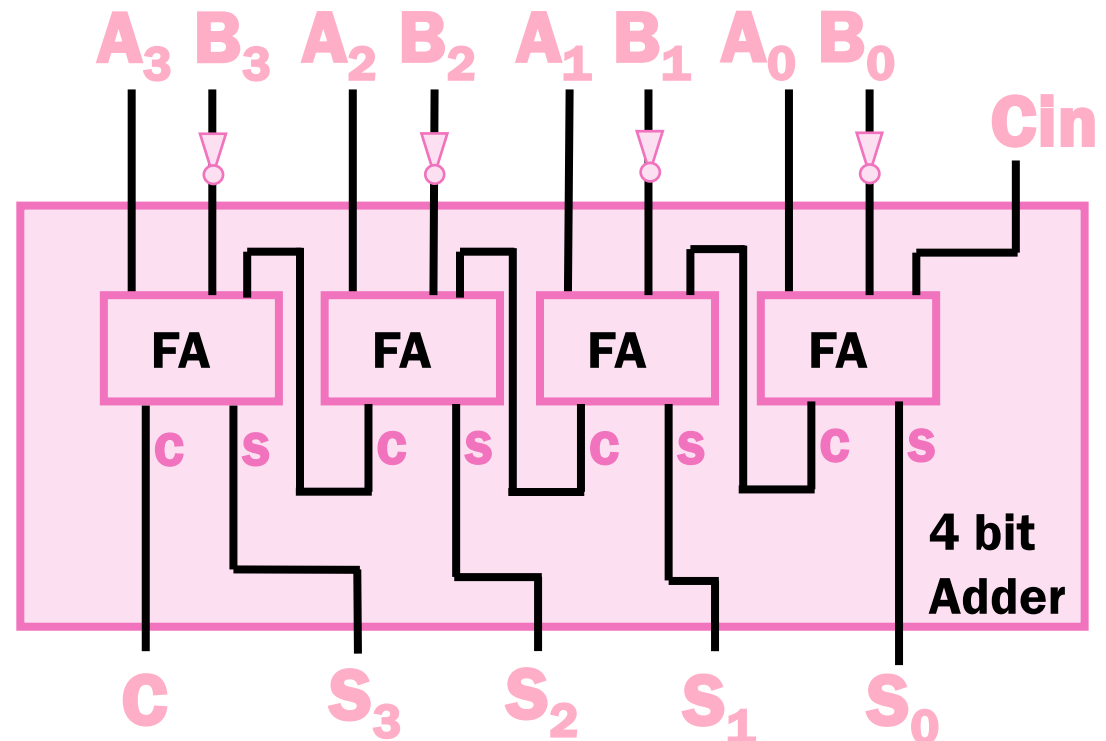
- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.
- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.
- Cómo podemos hacer que el input sea en complemento de 2?



Circuito Restador

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

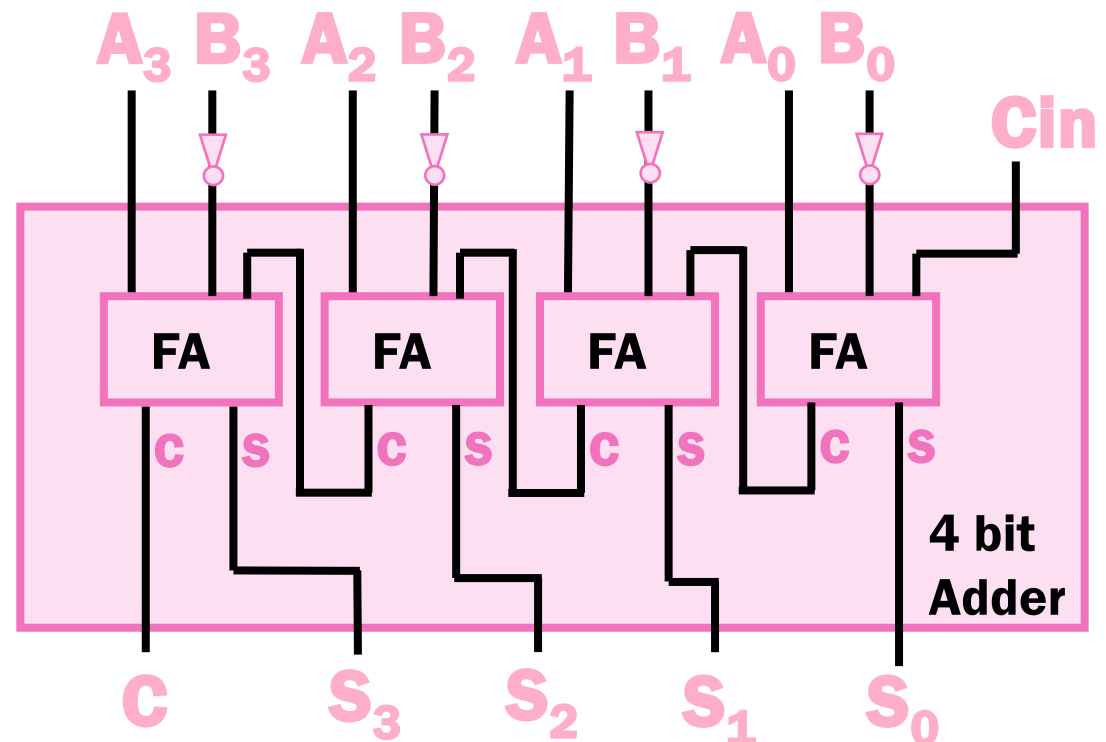
- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.
- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.
- Cómo podemos hacer que el input sea 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,
 - $A - B = A + (-B)$

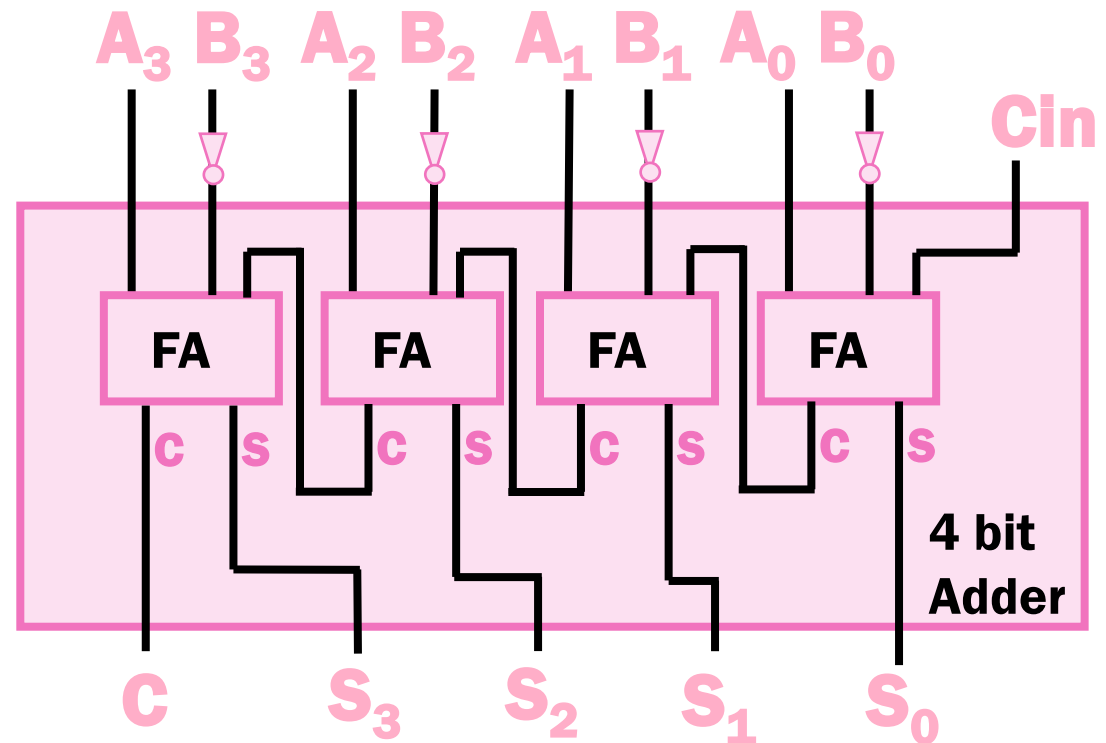
- Como sabemos que podemos restar, sumando A con el inverso aditivo de B, podemos usar el mismo circuito sumador, pero con una modificación en los inputs.
- Cuál modificación? Ya no queremos tener a B como input del circuito sumador, si no que su complemento de 2.
- Cómo podemos hacer que el input sea en complemento de 2? Necesitamos invertir B (negar) y sumar 1.



Circuito Restador

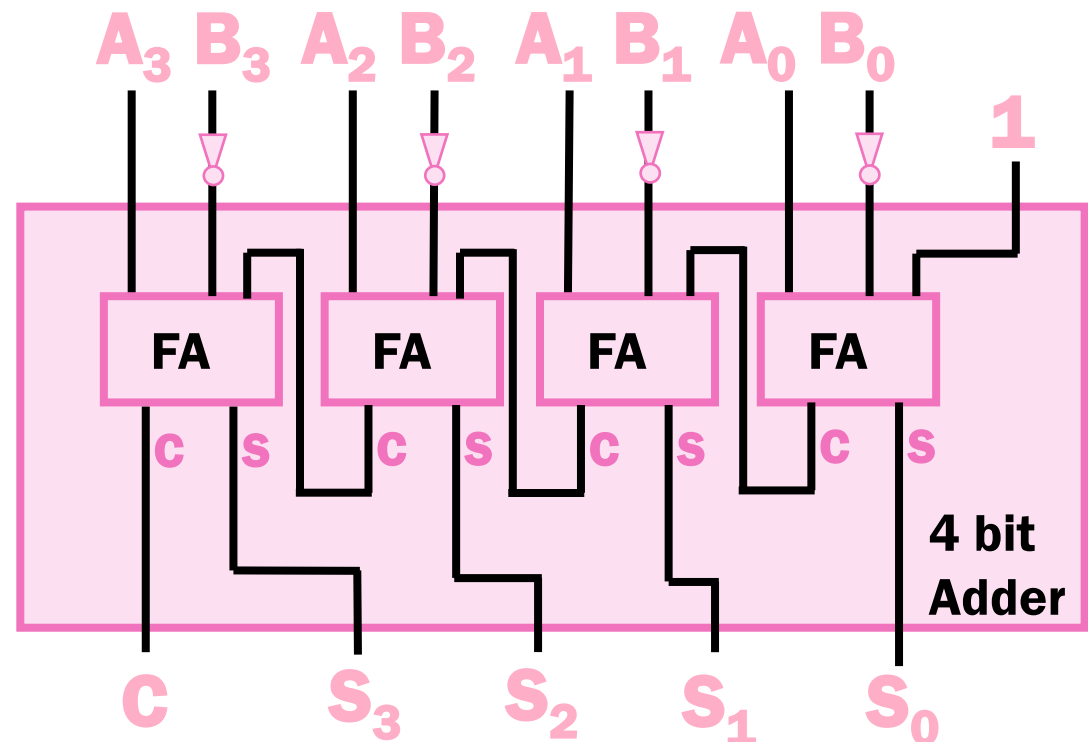
- Entonces ahora tenemos una forma de representar números negativos de tal manera que podamos usar,
 - $A - B = A + (-B)$

- 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 (\wedge).
 - 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 (\wedge).
 - Cuando se debe negar el input? Cuando aparece como un 0.
- 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.

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

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 (\wedge).
 - Cuando se debe negar el input? Cuando aparece como un 0.
- 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.

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

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

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 (\wedge).
 - Cuando se debe negar el input? Cuando aparece como un 0.
- 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.

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

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

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

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 (\wedge).
 - Cuando se debe negar el input? Cuando aparece como un 0.
- 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.

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

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

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

$$m_8 = x \cdot y \cdot z$$

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.

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

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

$$M_0 = x + y + z$$

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

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

$$M_0 = x + y + z$$

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

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

$$M_0 = x + y + z$$

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

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

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

$$M_0 = x + y + z$$

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

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

$$M_4 = x + y + \bar{z}$$

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 (\wedge).
- 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 (\wedge).
- 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



DCC
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343 – Arquitectura de Computadores

Clase 1 – Lógica Digital

Susana Figueroa