DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Organización del Computador 1
Lógica Digital 1: circuitos combinacionales

Dr. Marcelo Risk

23 de agosto de 2022

Índice

Lógica digital

- La computadoras necesitan almacenar datos e instrucciones en memoria.

- Sistema binario: solo dos estados posibles.

- ¿Por qué?
  - Es mucho más sencillo identificar entre solo dos estados.
  - Es menos propenso a errores

Diseño de circuitos

- Circuitos que operan con valores lógicos:
  - Verdadero = 1
  - Falso = 0

- **Idea**: realizar diferentes operaciones lógicas y matemáticas combinando circuitos.

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.
  - 1 / 0.

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.
  - 1 / 0.
- Tres operadores:

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.
  - 1 / 0.
- Tres operadores:
  - AND (y).

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.
  - 1 / 0.
- Tres operadores:
  - AND (y).
  - OR (o).

Álgebra de Boole

Figura: George Boole (1815-1864).

- George Boole, desarrolló un sistema algebraico para formular proposiciones con símbolos.
- Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios:
  - verdadero / falso.
  - `on` / `off`.
  - 1 / 0.
- Tres operadores:
  - AND (y).
  - OR (o).
  - NOT (no).

Álgebra de Boole

- Las variables Booleanas **solo** pueden tomar los valores binarios: 1 ó 0.
- Una variable Booleana **representa un bit**.
- Este álgebra se presenta formalmente en la materia *Lógica y Computabilidad*, acá vamos a dar los conceptos prácticos para utilizarla en Lógica Digital.

Álgebra de Boole

- Las variables Booleanas **solo** pueden tomar los valores binarios: 1 ó 0.
- Una variable Booleana **representa un bit**.
- Este álgebra se presenta formalmente en la materia *Lógica y Computabilidad*, acá vamos a dar los conceptos prácticos para utilizarla en Lógica Digital.

Definición

bit: se popularizó como una abreviatura de **Binary digIT**.

Operadores básicos: **AND**

- Un operador booleano puede ser completamente descripto usando tablas de verdad.

- El operador **AND** es conocido como producto booleano (.). También se lo nota como $\wedge$ (una letra 'v' invertida):

X	Y	X **AND** Y ($X \wedge Y$)
0	0	0
0	1	0
1	0	0
1	1	1

Operadores básicos: **OR**

- El operador **OR** es conocido como suma booleana (+). También se lo nota como $\vee$:

X	Y	X **OR** Y ($X \vee Y$)
0	0	0
0	1	1
1	0	1
1	1	1

Operadores básicos: **NOT**

- El operador NOT se nota con una barra $\overline{X}$. Otra forma de notarlo es $\neg X$.

X	$\overline{X}$ $(\neg X)$
0	1
1	0

Funciones booleanas

- Queremos hacer la tabla de verdad de esta función
  $F(x, y, z) = x\bar{z} + y$

- El **NOT** tiene mayor precedencia que el resto de los operadores

- El **AND** mayor precedencia que el **OR**

Funciones booleanas

- Queremos hacer la tabla de verdad de esta función
  $F(x, y, z) = x\bar{z} + y$
- El **NOT** tiene mayor precedencia que el resto de los operadores
- El **AND** mayor precedencia que el **OR**

Comenzamos con la operación que tiene mayor precedencia:

x	y	z	$\bar{z}$
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Funciones booleanas

- Queremos hacer la tabla de verdad de esta función
  $F(x, y, z) = x\bar{z} + y$
- El **NOT** tiene mayor precedencia que el resto de los operadores
- El **AND** mayor precedencia que el **OR**

Seguimos resolviendo el **AND** usando lo que recién calculamos

x	y	z	$\bar{z}$	$x\bar{z}$
0	0	0	1	**0**
0	0	1	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
1	0	0	1	**1**
1	0	1	0	**0**
1	1	0	1	**1**
1	1	1	0	**0**

Funciones booleanas

- Queremos hacer la tabla de verdad de esta función
  $F(x, y, z) = x\bar{z} + y$
- El **NOT** tiene mayor precedencia que el resto de los operadores
- El **AND** mayor precedencia que el **OR**

Finalmente hacemos el **OR**

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	**0**
0	0	1	0	0	**0**
0	1	0	1	0	**1**
0	1	1	0	0	**1**
1	0	0	1	1	**1**
1	0	1	0	0	**0**
1	1	0	1	1	**1**
1	1	1	0	0	**1**

Propiedades

Identidad	$1.A = A$	$0 + A = A$
Nula	$0.A = 0$	$1 + A = 1$
Idempotencia	$A.A = A$	$A + A = A$
Inversa	$A.\overline{A} = 0$	$A + \overline{A} = 1$
Conmutativa	$A.B = B.A$	$A + B = B + A$
Asociativa	$(A.B).C = A.(B.C)$	$(A + B) + C = A + (B + C)$
Distributiva	$A + B.C = (A + B).(A + C)$	$A.(B + C) = A.B + A.C$
Absorción	$A.(A + B) = A$	$A + A.B = A$
de Morgan	$\overline{A.B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$ de Morgan

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva
$(x+x.\overline{y}+y.x+0).(\overline{x}+z)$	Idempotencia e inversa

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva
$(x+x.\overline{y}+y.x+0).(\overline{x}+z)$	Idempotencia e inversa
$(x+x.(\overline{y}+y)).(\overline{x}+z)$	Nula y distributiva

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva
$(x+x.\overline{y}+y.x+0).(\overline{x}+z)$	Idempotencia e inversa
$(x+x.(\overline{y}+y)).(\overline{x}+z)$	Nula y distributiva
$x.(\overline{x}+z)$	Inversa, identidad y nula

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva
$(x+x.\overline{y}+y.x+0).(\overline{x}+z)$	Idempotencia e inversa
$(x+x.(\overline{y}+y)).(\overline{x}+z)$	Nula y distributiva
$x.(\overline{x}+z)$	Inversa, identidad y nula
$x.\overline{x}+x.z$	Distributiva

Identidades: ejemplo de aplicación

- Usando identidades booleanas podemos reducir esta función:
  $F(x,y,z) = (x+y).(x+\overline{y}).\overline{(x.\overline{z})}$

$(x+y).(x+\overline{y}).(\overline{x}+z)$	de Morgan
$(x.x+x.\overline{y}+y.x+y.\overline{y}).(\overline{x}+z)$	Distributiva
$(x+x.\overline{y}+y.x+0).(\overline{x}+z)$	Idempotencia e inversa
$(x+x.(\overline{y}+y)).(\overline{x}+z)$	Nula y distributiva
$x.(\overline{x}+z)$	Inversa, identidad y nula
$x.\overline{x}+x.z$	Distributiva
$x.z$	Inversa e identidad

Fórmulas equivalentes

- Varias fórmulas pueden tener la misma tabla de verdad:
  - Son lógicamente equivalentes.

- En general se suelen elegir las formas **canónicas**
  - Suma de productos:
    - $F_1(x, y, z) = x.y + z.x + y.z$
  - Producto de sumas:
    - $F_2(x, y, z) = (x + y).(z + x).(y + z)$

Suma de productos

- Es fácil convertir una función a una suma de productos usando la tabla de verdad.
- Elegimos los valores que dan 1 y hacemos un producto (AND) de la fila (negando si aparece un 0)?
- Luego sumamos todo (OR):

	x	y	z	$x\bar{z}+y$	
	0	0	0	0	
	0	0	1	0	
→	0	1	0	1	←
→	0	1	1	1	←
→	1	0	0	1	←
	1	0	1	0	
→	1	1	0	1	←
→	1	1	1	1	←

$$F(x,y,z) = (\bar{x}y\bar{z}) + (\bar{x}yz) + (x\bar{y}\bar{z}) + (xy\bar{z}) + (xyz)$$

Circuitos booleanos

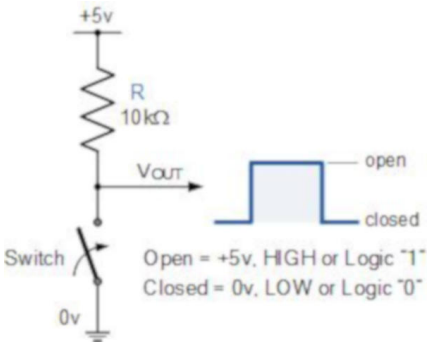
- Las computadores digitales contienen circuitos que implementan funciones booleanas.

- Cuando más simple la función más chico el circuito.

Circuitos booleanos

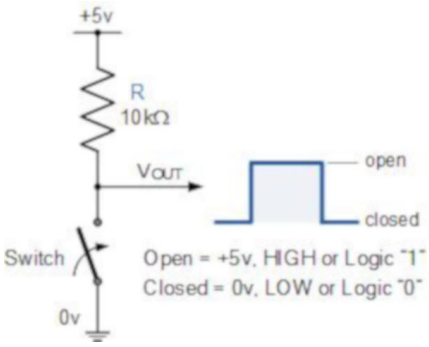
- Las computadores digitales contienen circuitos que implementan funciones booleanas.
- Cuando más simple la función más chico el circuito.
- Son más baratos, consumen menos, y ¡son mas rápidos!
- Podemos usar las identidades del álgebra de Boole para reducir estas funciones y hacer circuitos más simples.

¿Qué hay dentro?

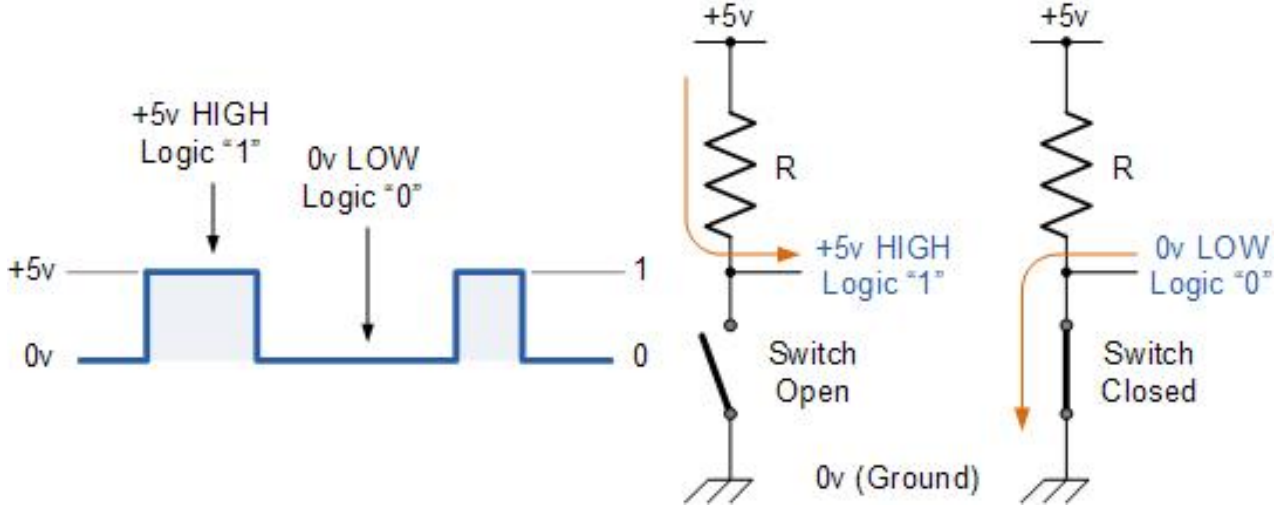


- Ya vimos que hay dos valores: 1 (5V) y 0 (0V).
- Este circuito se comporta como una **llave**.
- Se mira lo qué ocurre en **Vout** (tensión de salida).
- Cuando la llave está cerrada, `Vout` está conectada con la tierra (GROUND - 0 V), tiene un **0** o **False**.
- Cuando la llave está abierta, `Vout` está desconectada y `Vout` tiene **5V** o un **1** o **True**.

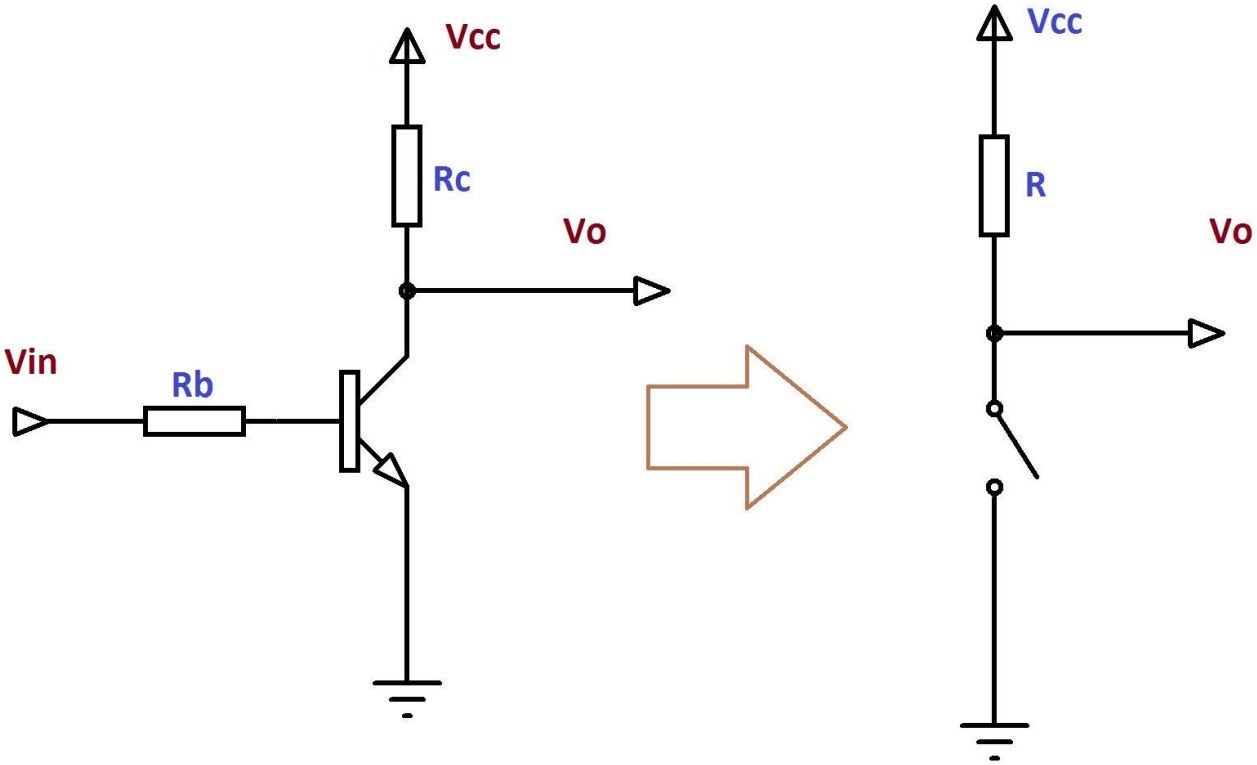
¿Qué hay dentro?



- Ya vimos que hay dos valores: 1 (5V) y 0 (0V).
- Este circuito se comporta como una **llave**.
- Se mira lo qué ocurre en **Vout** (tensión de salida).
- Cuando la llave está cerrada, `Vout` está conectada con la tierra (GROUND - 0 V), tiene un **0** o **False**.
- Cuando la llave está abierta, `Vout` está desconectada y `Vout` tiene **5V** o un **1** o **True**.

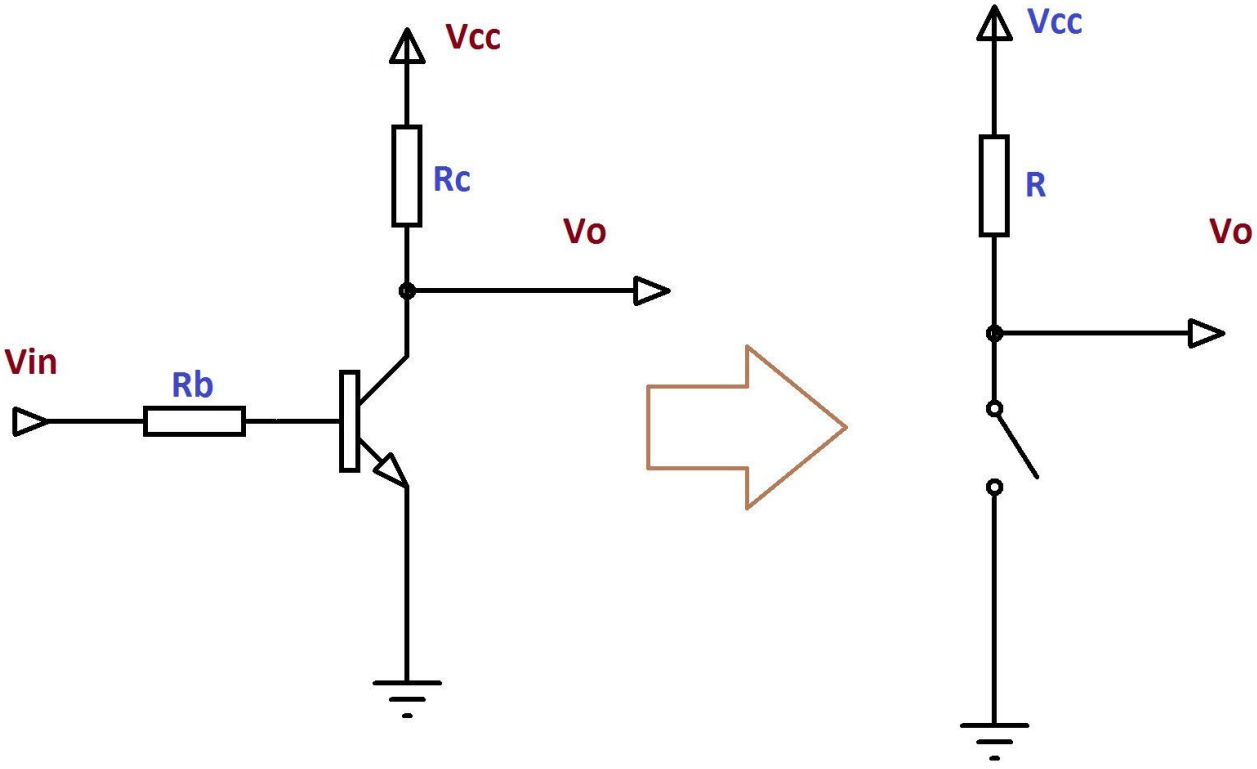


Pero, ¿lo digital dónde está?



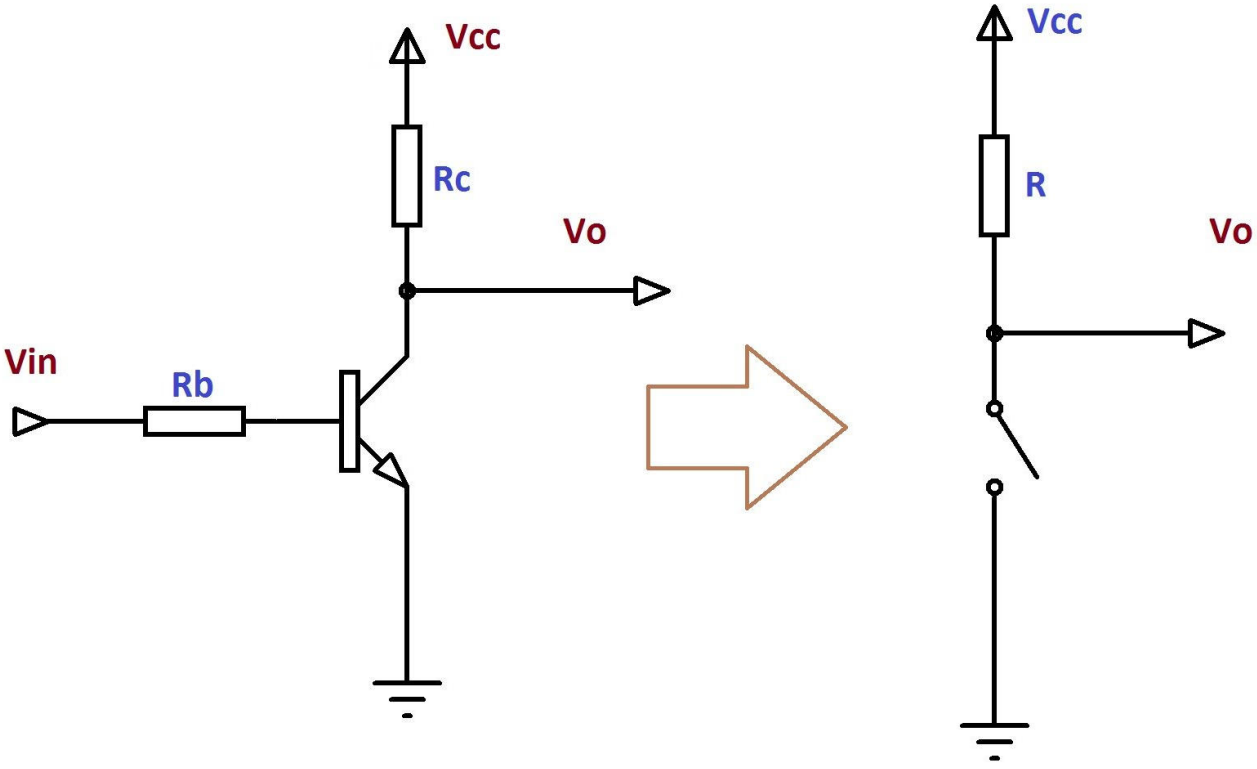
- El dispositivo del centro del circuito es un **transistor**, dispositivo que revolucionó nuestra historia en los años 70.

Pero, ¿lo digital dónde está?



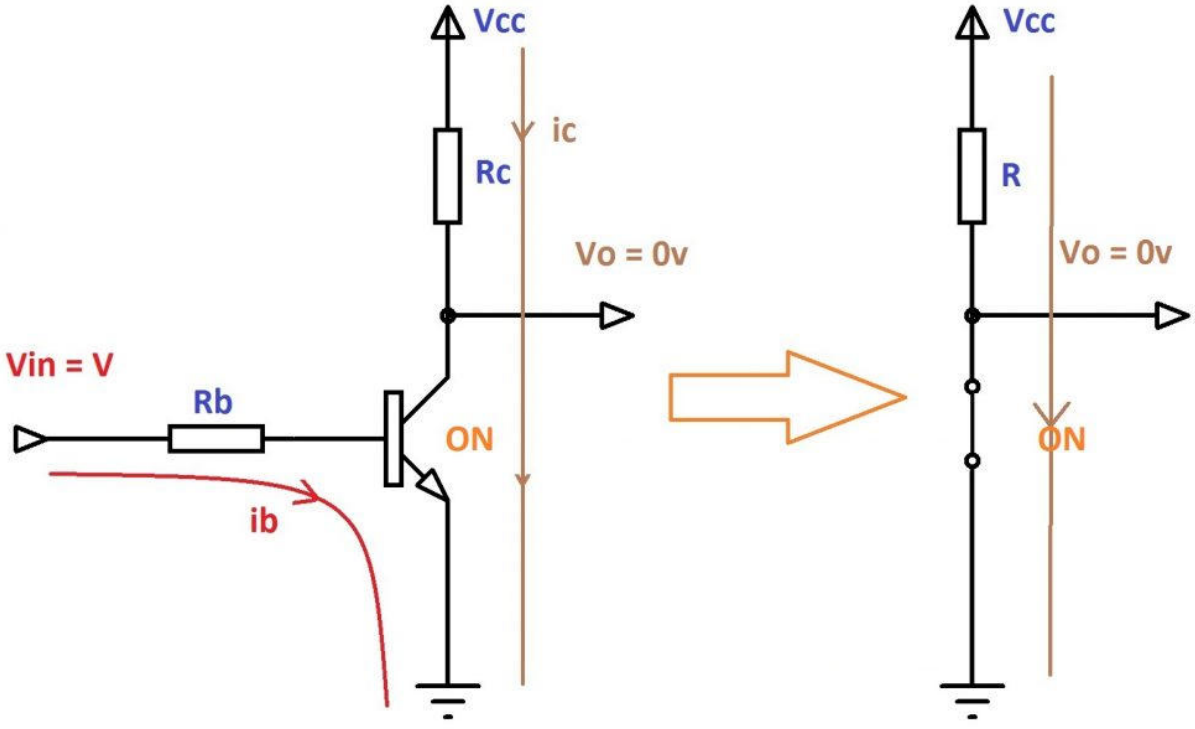
- El dispositivo del centro del circuito es un **transistor**, dispositivo que revolucionó nuestra historia en los años 70.
- Se lo puede usar como **amplificador** de una señal (lo que entra en `Vin`).

Pero, ¿lo digital dónde está?



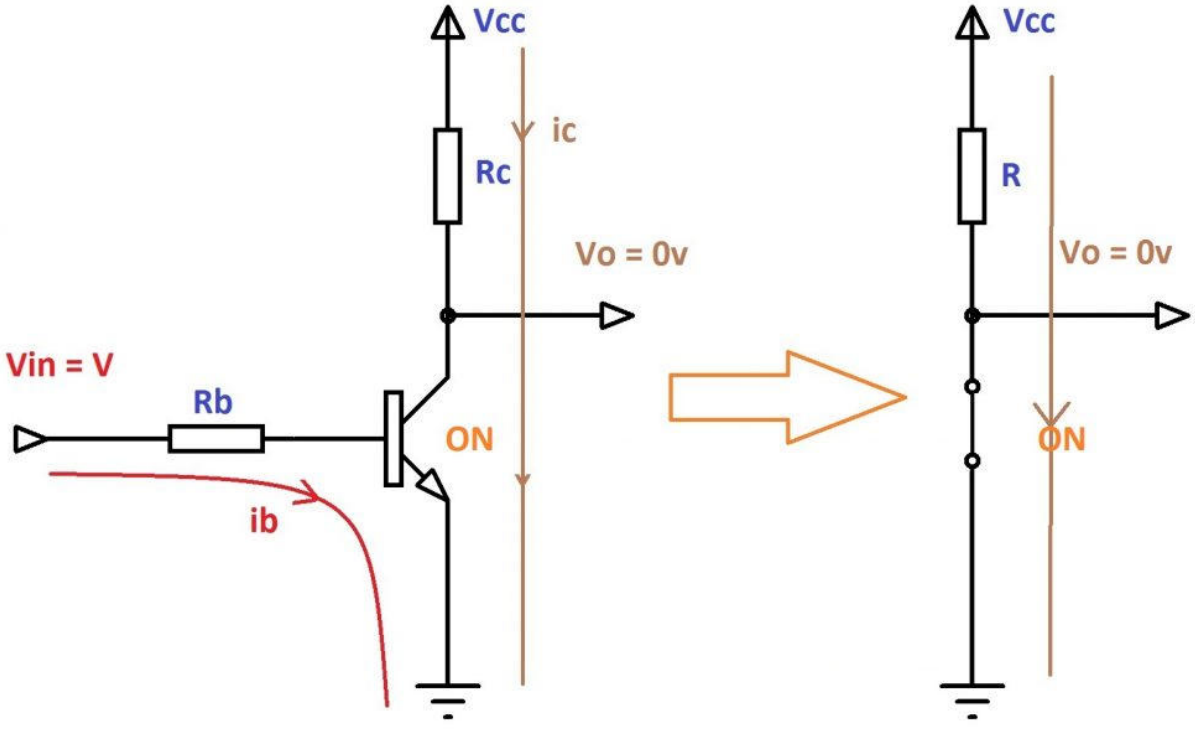
- El dispositivo del centro del circuito es un **transistor**, dispositivo que revolucionó nuestra historia en los años 70.
- Se lo puede usar como **amplificador** de una señal (lo que entra en `Vin`).
- También puede ser usado como una llave controlada por una señal.

Llave cerrada



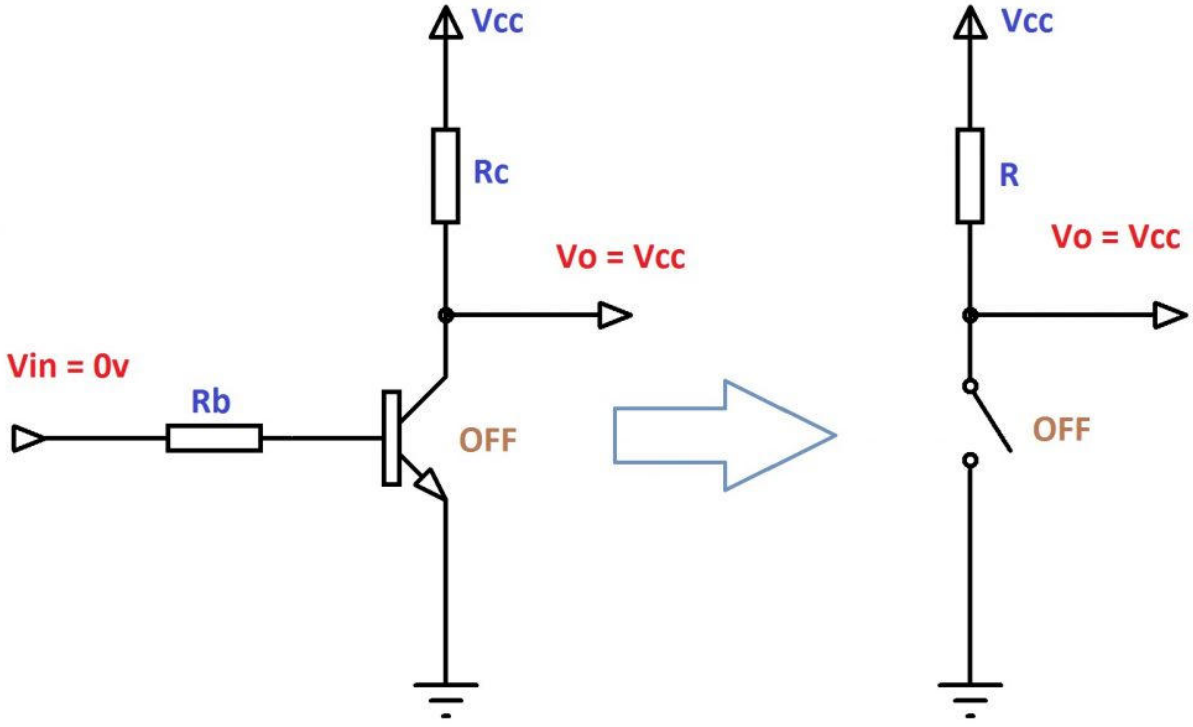
- El transistor se satura (se lo conoce como *saturated state*) cuando se aplica un potencial suficientemente alto en `Vin`.

Llave cerrada



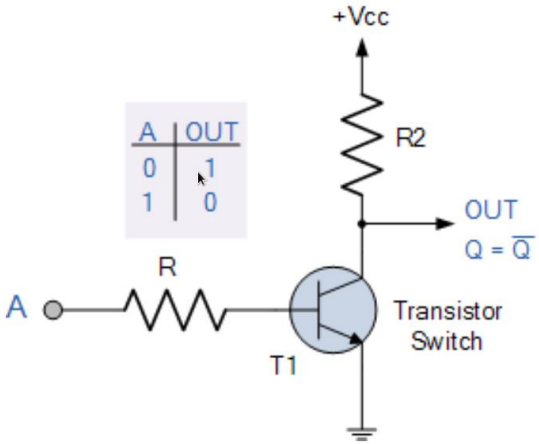
- El transistor se satura (se lo conoce como *saturated state*) cuando se aplica un potencial suficientemente alto en `Vin`.
- Durante esta condición el transistor actúa (casi) como si fuera un cortocircuito.
- Circula una corriente que depende de la resistencia.

Llave abierta



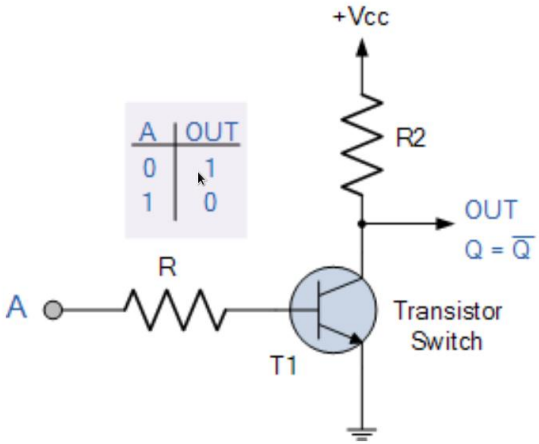
- El transistor está apagado (se lo conoce también como *cut-off state* cuando el potencial aplicado a `Vin` es 0V.
- En este estado, el transistor actúa como si fuera un circuito abierto.
- No hay circulación de corriente entre VCC y GROUND.

Pero... ¿Qué tiene que ver con la computadora?



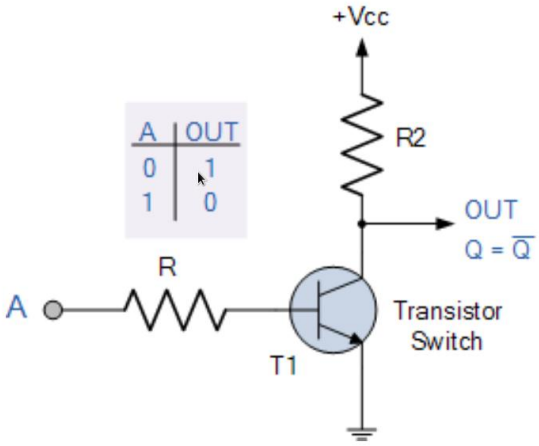
- Se mira la señal marcada en `Out`.

Pero... ¿Qué tiene que ver con la computadora?



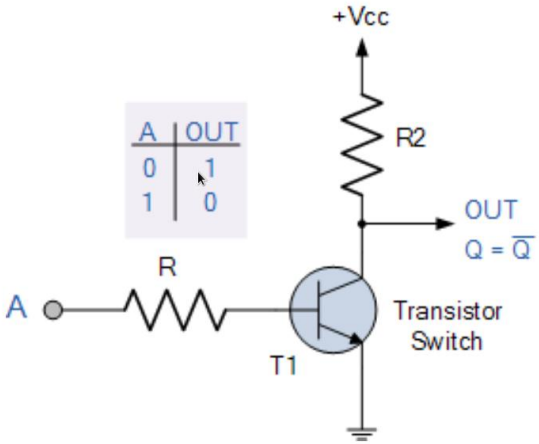
- Se mira la señal marcada en `Out`.
- Cuando se aplica un **1** en A, en la salida tenemos un **0**.

Pero... ¿Qué tiene que ver con la computadora?



- Se mira la señal marcada en `Out`.
- Cuando se aplica un **1** en A, en la salida tenemos un **0**.
- Cuando se aplica un **0** en A, en la salida tenemos un **1**.

Pero... ¿Qué tiene que ver con la computadora?



- Se mira la señal marcada en `Out`.
- Cuando se aplica un **1** en A, en la salida tenemos un **0**.
- Cuando se aplica un **0** en A, en la salida tenemos un **1**.

Este circuito implementa la **negación** de una variable booleana.

Compuertas lógicas

Definición

Una **compuerta** es un dispositivo electrónico que produce un resultado en base a un conjunto de valores de entrada.

Compuertas lógicas

Definición

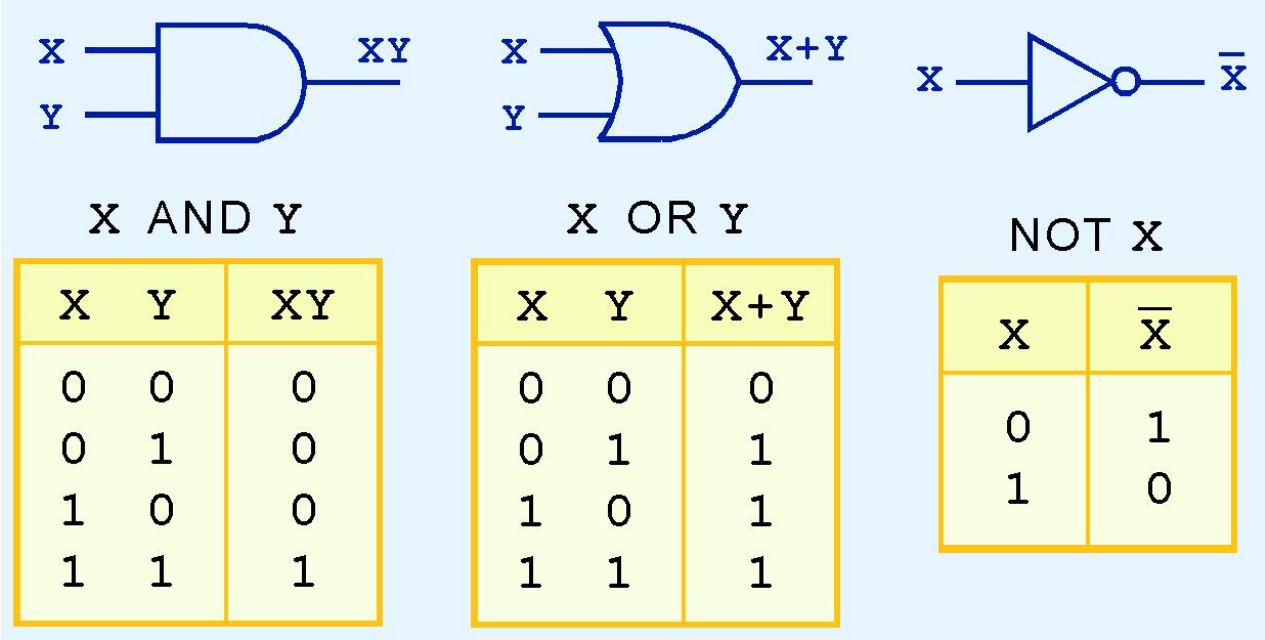
Una **compuerta** es un dispositivo electrónico que produce un resultado en base a un conjunto de valores de entrada.

- En realidad, están formadas por uno o varios transistores, pero lo podemos ver como una **unidad**.

- Los circuitos integrados contienen colecciones de compuertas conectadas con algún propósito.

Compuertas lógicas

- Las más simples: AND, OR, y NOT:



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

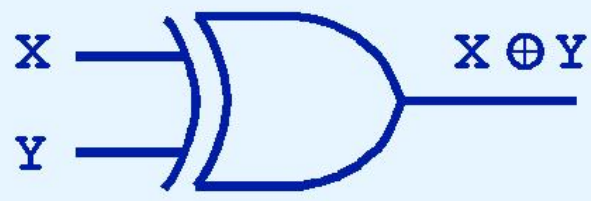
- Se corresponden exactamente con las funciones booleanas que vimos.

Compuertas lógicas

- Una compuerta muy útil: el OR exclusivo => XOR.
- La salida es 1 cuando los valores de entrada difieren.

X XOR Y

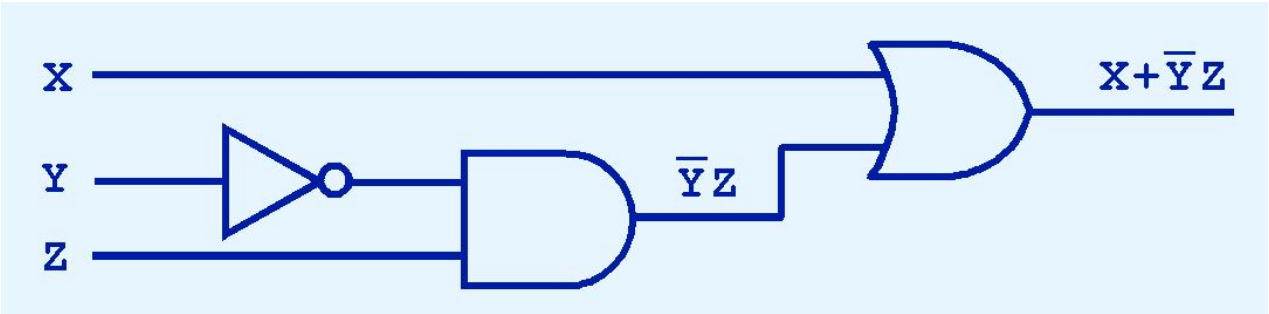
X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0



Implementación de funciones booleanas

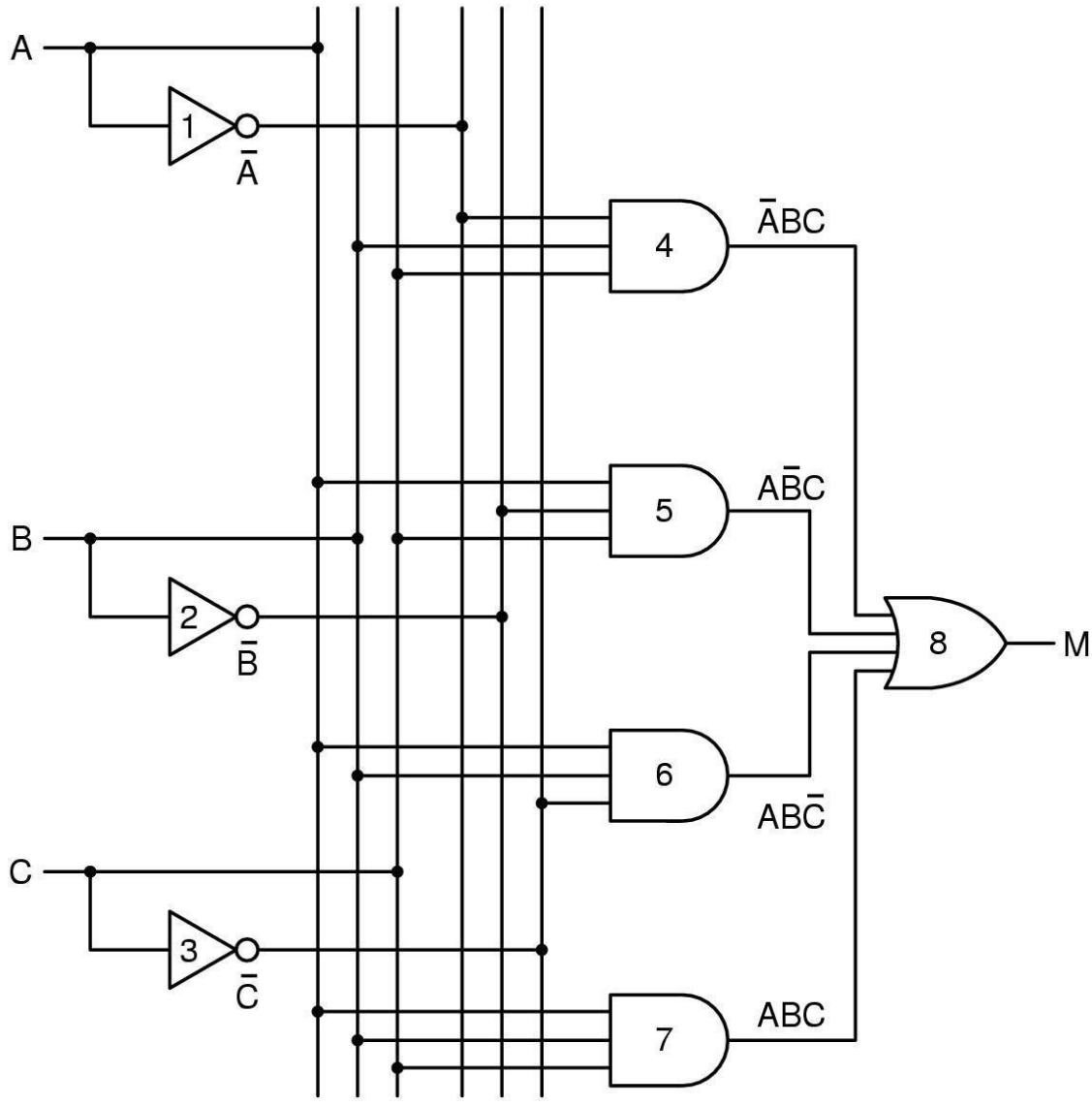
- Combinando compuertas se pueden implementar funciones booleanas.

- Este circuito implementa la siguiente función: $F(x, y, z) = x + \overline{y}z$



Ejemplo: función mayoría

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

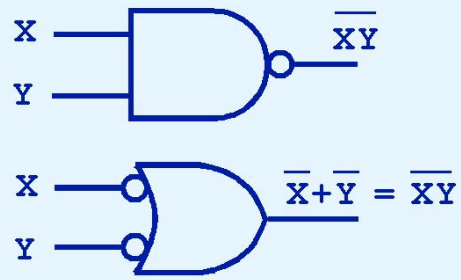


Compuertas lógicas combinadas

- NAND y NOR son dos compuertas lógicas combinadas.
- Con la identidad de Morgan se pueden implementar con AND u OR.
- Son más baratas y cualquier operación básica se puede representar usándolas cualquiera de ellas (¡sin usar la otra!).

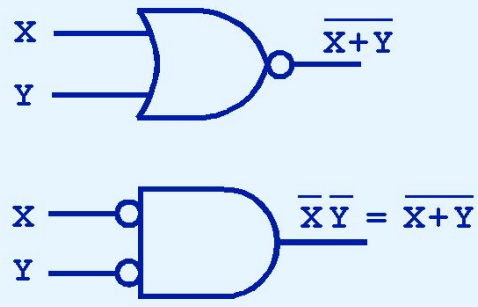
X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

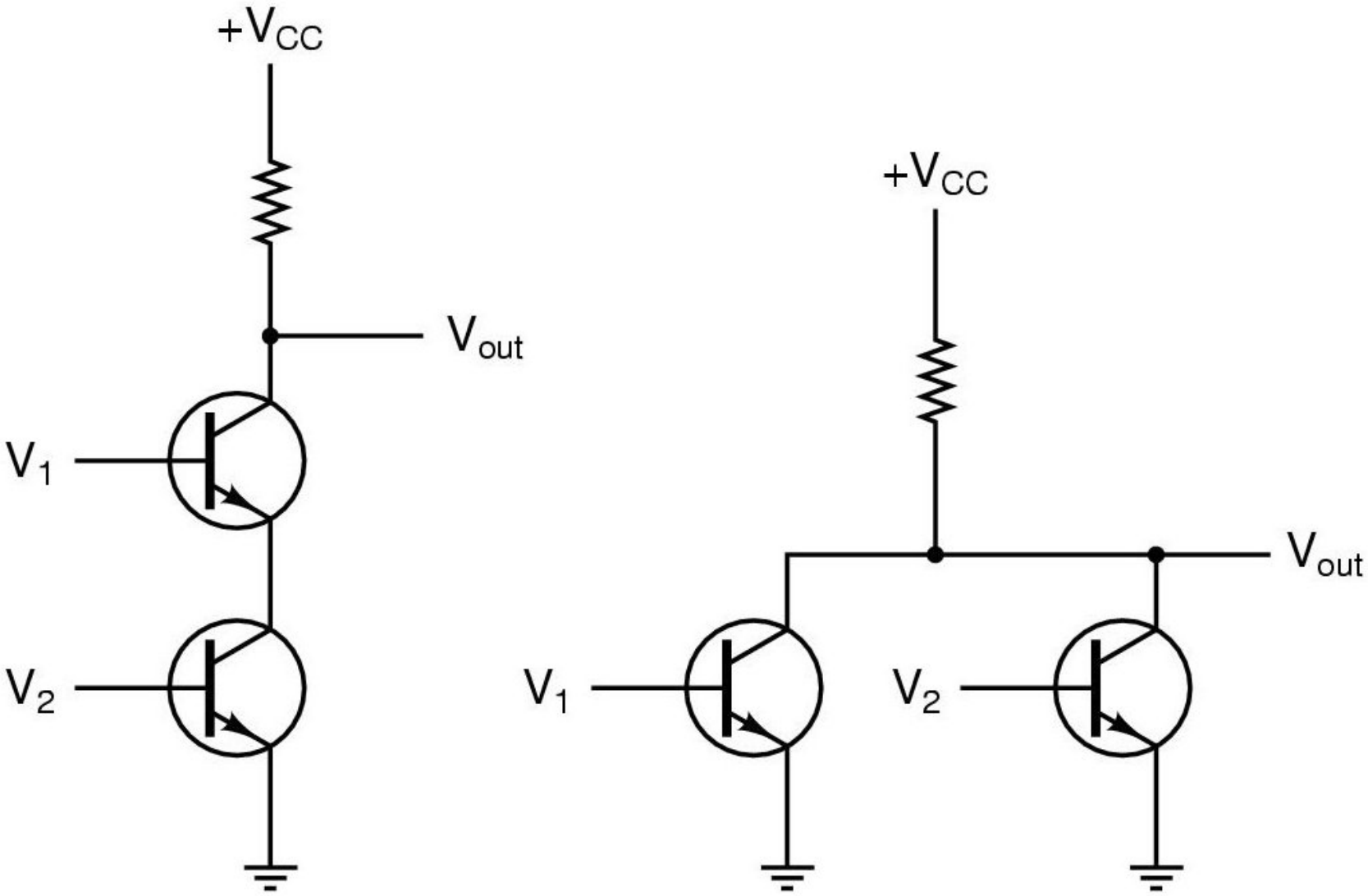


X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

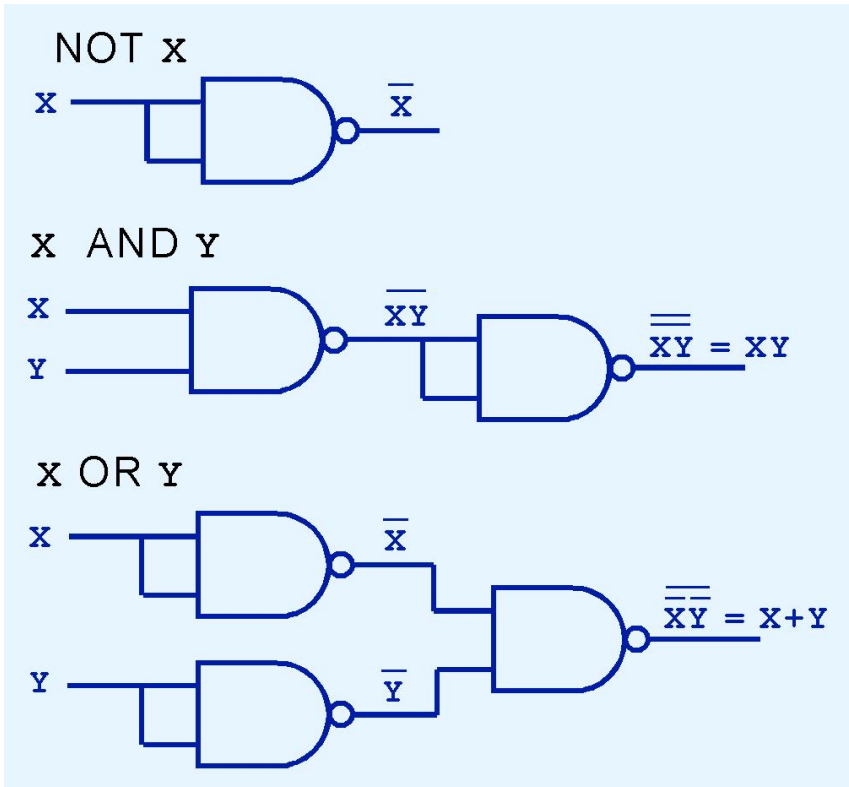


NAND y NOR



Ejemplos

- NOT usando NAND: simplemente unir las dos entradas.
- Utilizando solo NAND o NOR se pueden realizar circuitos con la misma funcionalidad que el AND y OR.



Circuitos combinatorios

- Producen una salida específica al (casi) instante que se le aplican valores de entrada.
- Implementan funciones booleanas.
- La aritmética y la lógica de la CPU se implementan con estos circuitos.

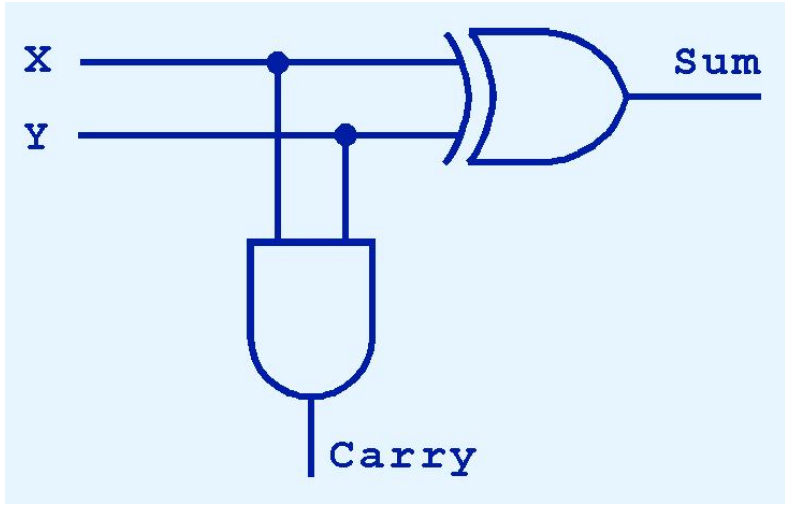
Sumador

- ¿Cómo podemos construir un circuito que sume dos bits `X` e `Y`?
- $F(X, Y) = X + Y$ (suma aritmética)
- ¿Qué pasa si $X = 1$ e $Y = 1$?

X	Y	*Suma*	*carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

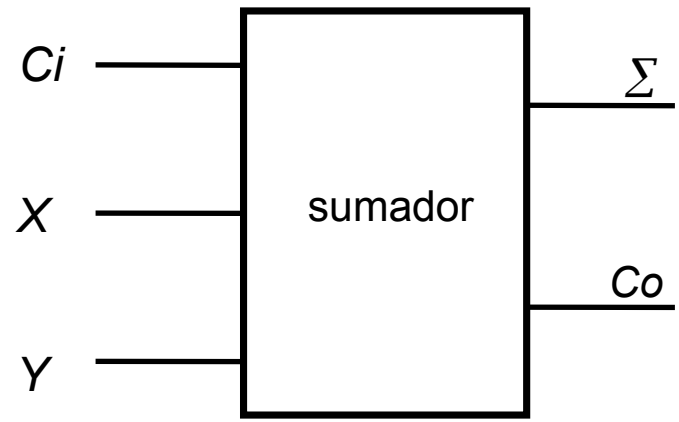
Semi-Sumador

- Podemos usar un XOR para la suma y un AND para el carry.
- A este circuito se lo llama semi-sumador (*half-adder*).

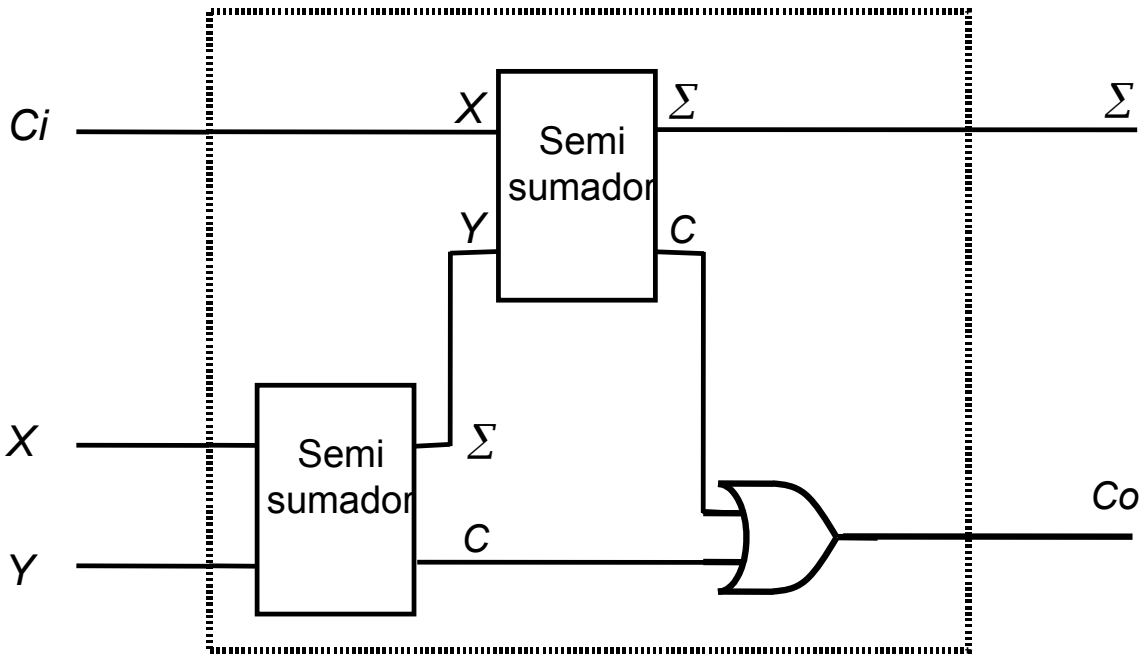


Sumador

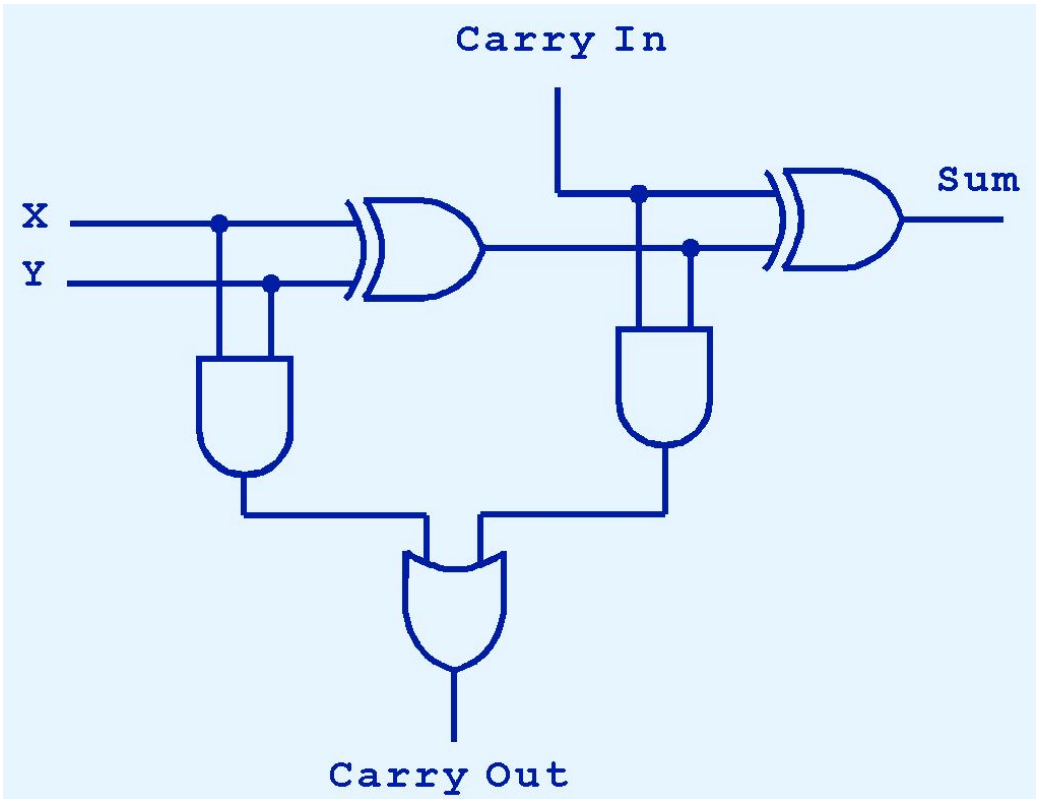
Co	1	
X		1
Y		1
	1	0



Sumador: estructura interna



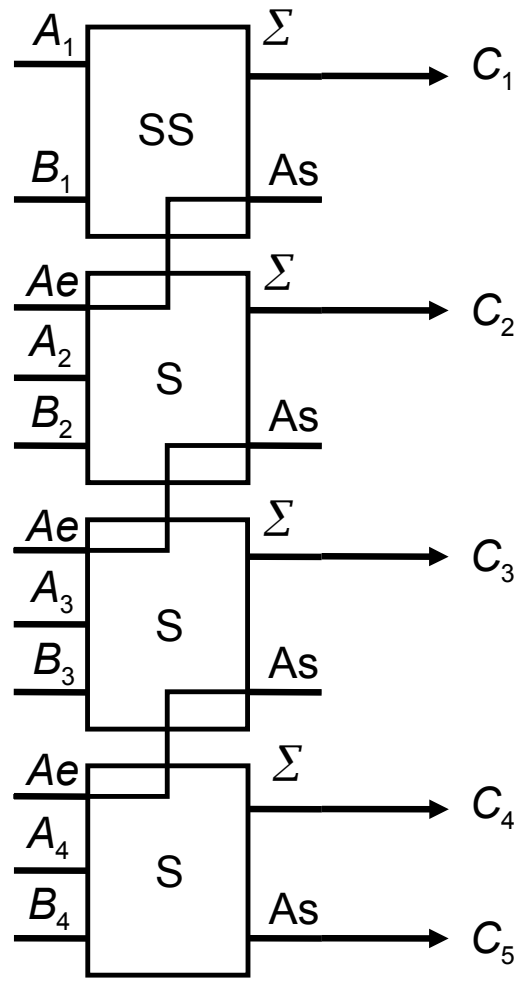
Sumador: estructura interna y tabla de verdad



X	*Y*	*Ci*	*Suma*	*Co*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sumador de 4 bits

	A_4	A_3	A_2	A_1
+	B_4	B_3	B_2	B_1
C_5	C_4	C_3	C_2	C_1

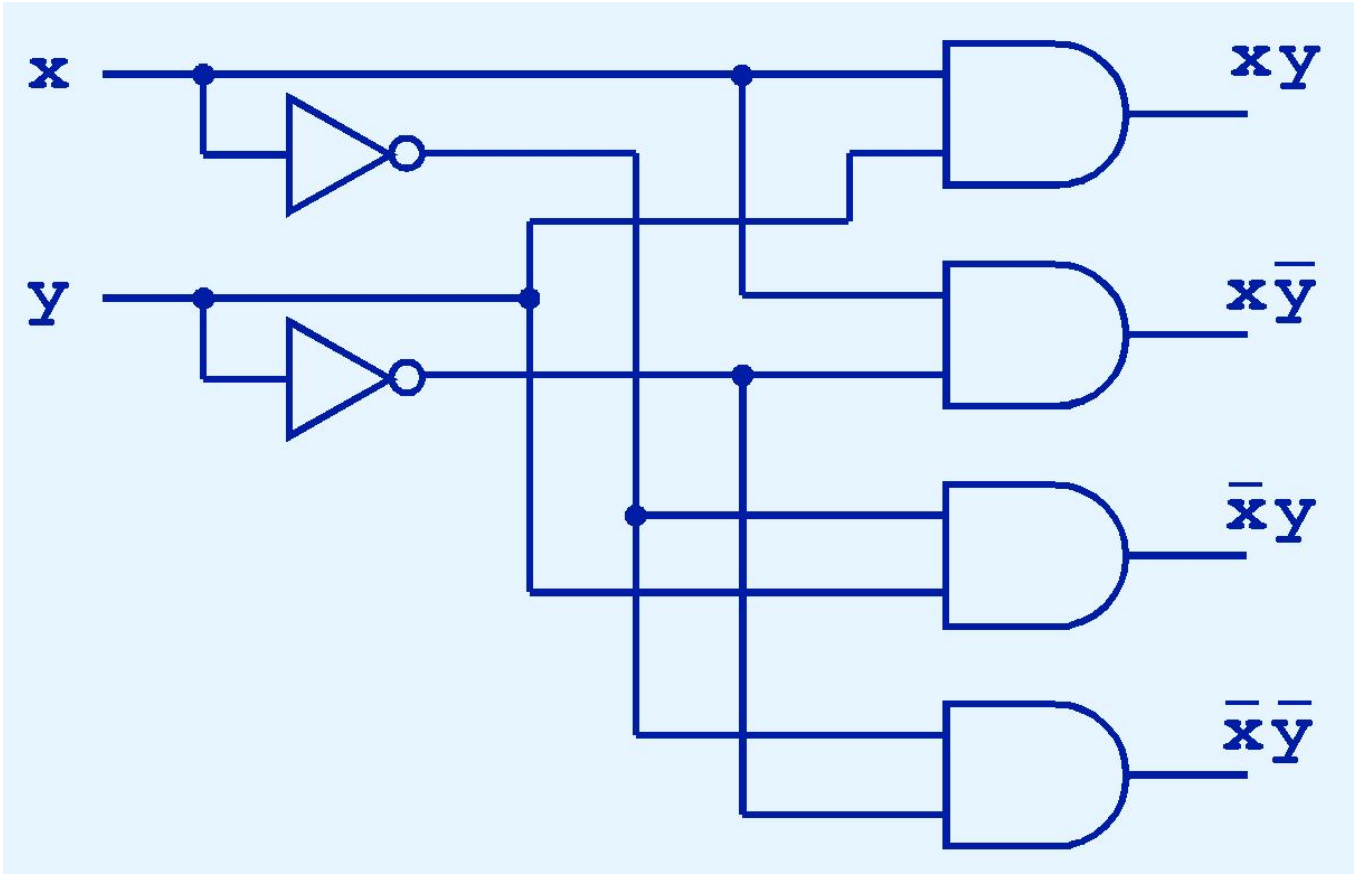


Decodificadores

- Los decodificadores de n entradas pueden seleccionar una de 2^n salidas.
- Son ampliamente utilizados.
- Por ejemplo:
  - Seleccionar una locación en una memoria a partir de una dirección colocada en el bus memoria.

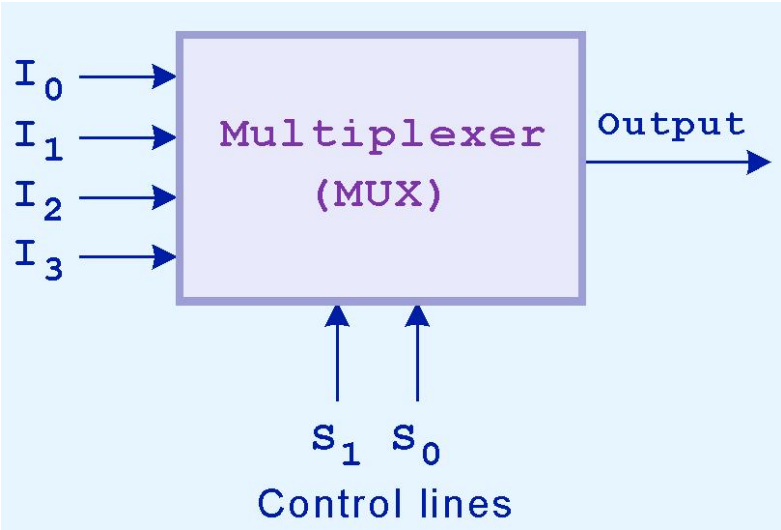
Decodificadores: ejemplo

- Decodificador 2-a-4:



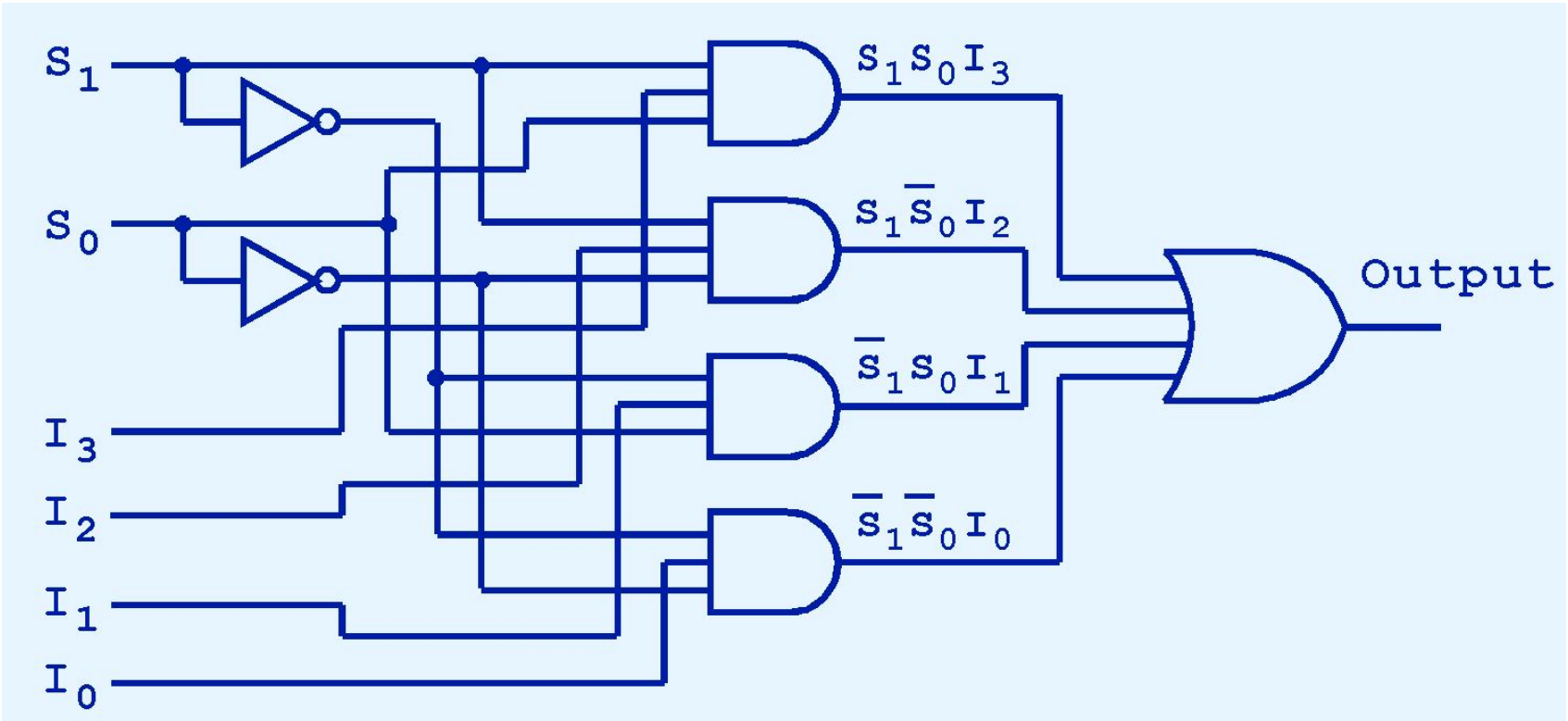
Multiplexores

- Selecciona una salida a de varias entradas.
- La entrada que es seleccionada como salida es determinada por las líneas de control.
- Para seleccionar entre n entradas, se necesitan $log_2 n$ líneas de control.
- Demultiplexor
  - Exactamente lo contrario al multiplexor.
  - Dada una entrada la direcciona entre n salidas, usando $log_2 n$ líneas de control.



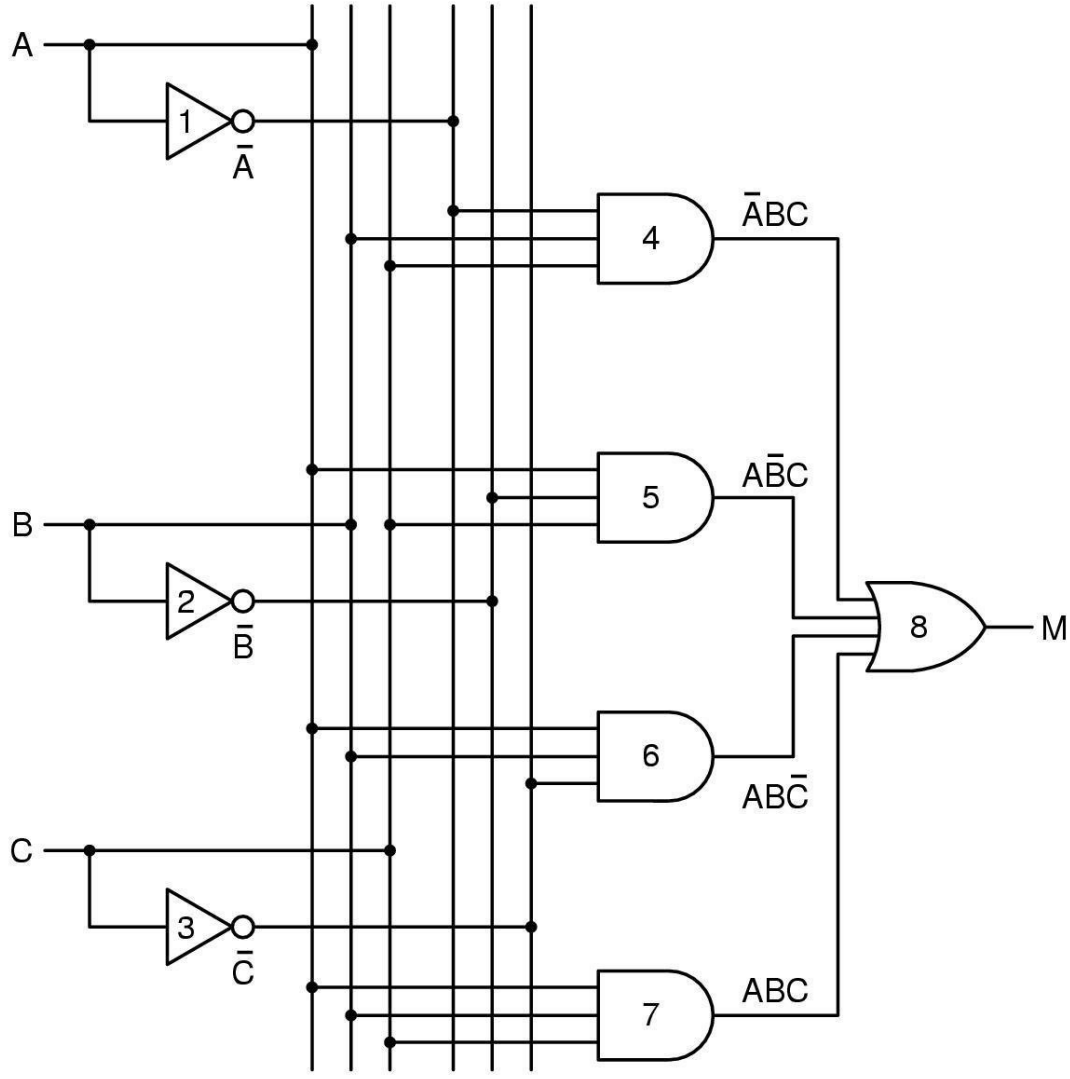
Multiplexor: ejemplo

- Multiplexor 4-a-1:

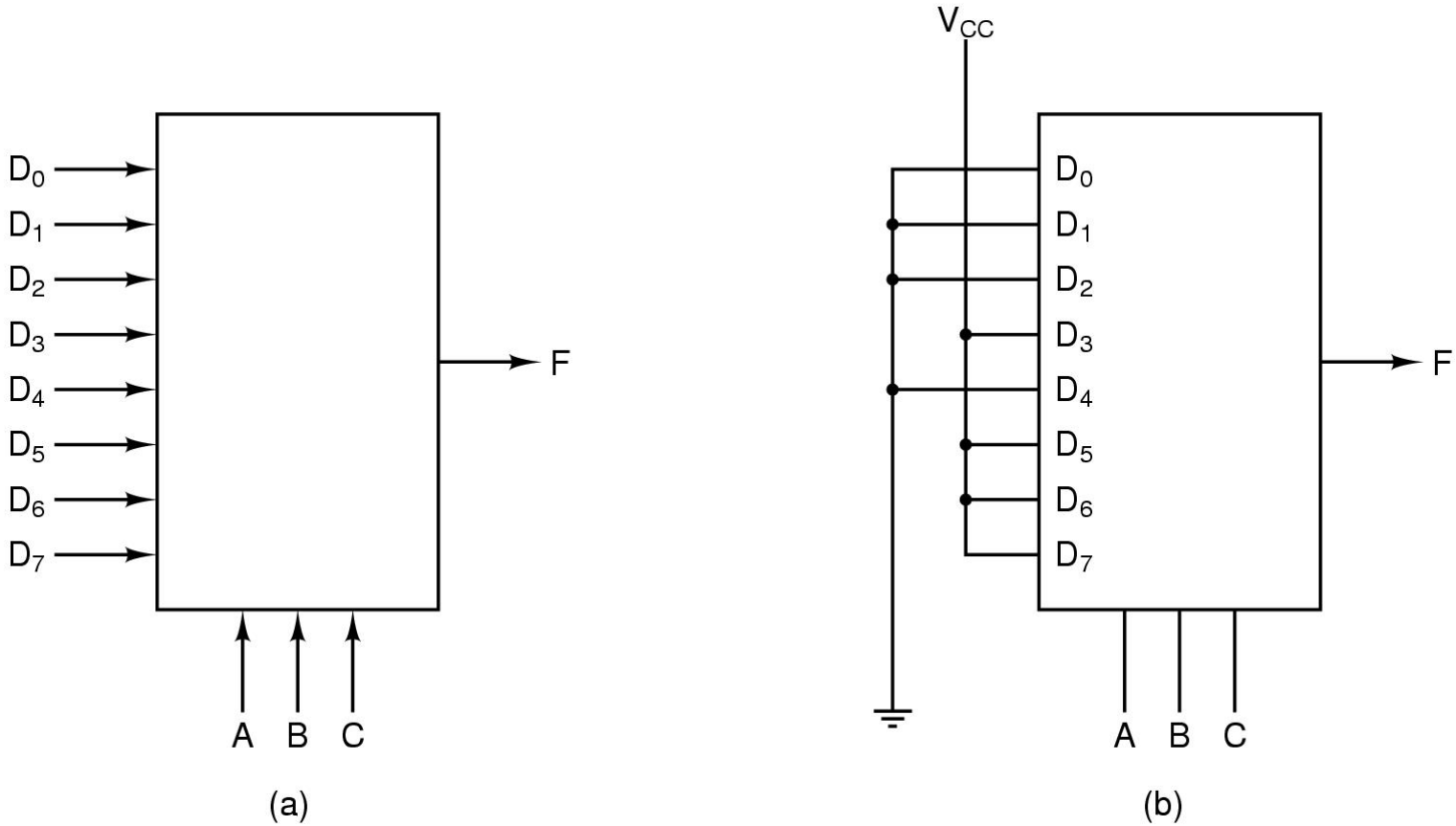


Función mayoría

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



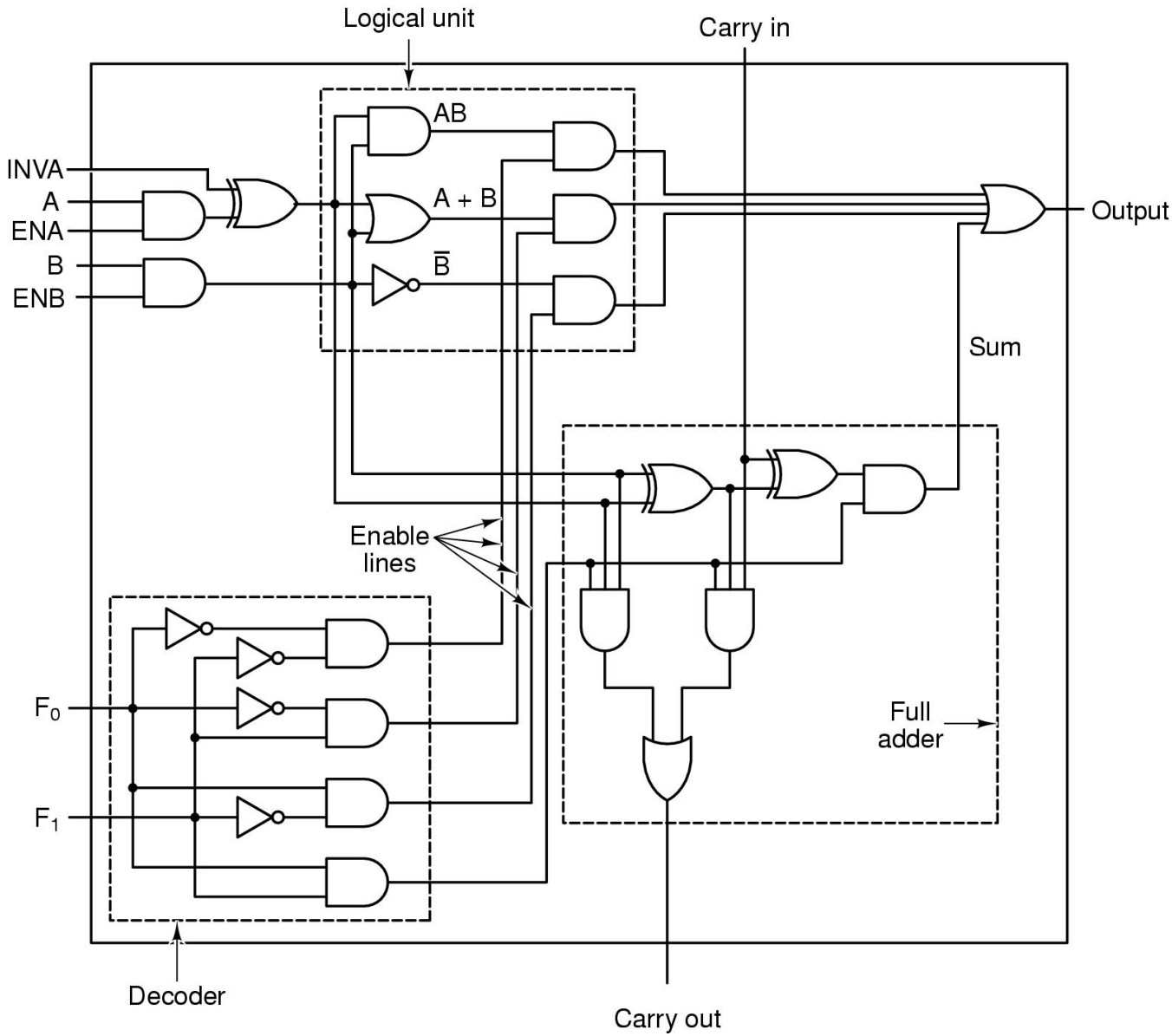
Función mayoría con multiplexor



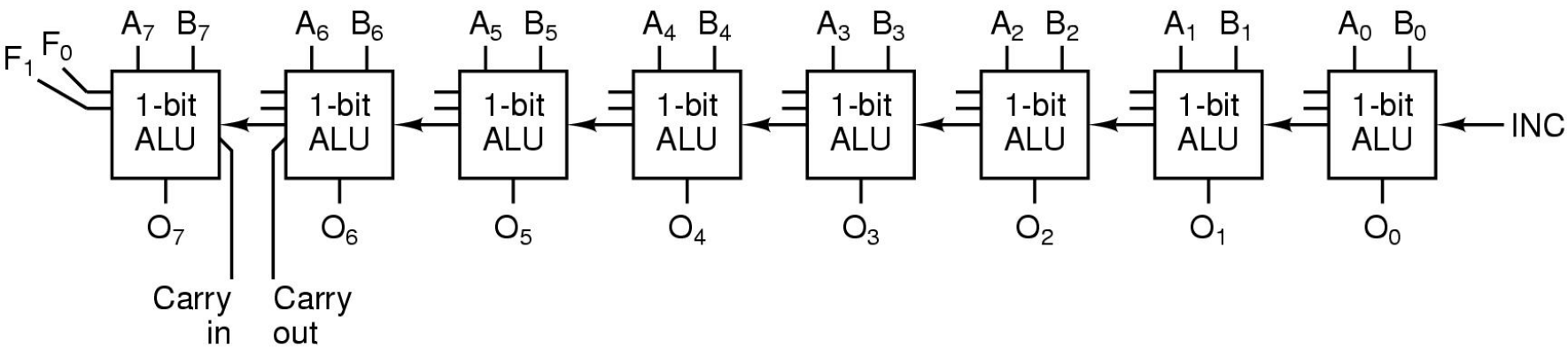
Ejemplo

- Construir una ALU de 1 bit.
- 3 entradas:
  - A, B, carry.
- 4 operaciones:
  - A.B, A+B, NOT B, Suma(A,B,Carry).
- Salidas:
  - Resultado, Carry out.

ALU de 1 bit


Logical unit
Carry in
AB
A + B
$\overline{B}$
INVA
A
ENA
B
ENB
Output
Sum
Enable lines
F_0
F_1
Full adder
Decoder
Carry out

ALU de 8 bits


A7 B7
A6 B6
A5 B5
A4 B4
A3 B3
A2 B2
A1 B1
A0 B0
F1 F0
1-bit ALU
INC
O7
O6
O5
O4
O3
O2
O1
O0
Carry in
Carry out

Memoria ROM

Entradas					Salidas							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.							.			
		.							.			
		.							.			
1	1	1	1	0	0	1	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1	0	0	1	1

ROM con decoder