

IIC2343 - Arquitectura de Computadores  
26 de Marzo

# Ayudantía 2

Aritmética, números y control digital

$( * ^ - ^ * )$

Hecha por:

- Tomás Contreras
- Susana Figueroa
- Laurence Golborne

## Temas:

- Complemento a 2
- Suma en complemento a 2
- Inverso aditivo
- Resta en complemento a 2
- Overflow
- Multiplexores
- Enablers
- Decoders
- Comparadores

# Complemento a 2

Cero (0) en el bit más significativo: positivo.

Uno (1) en el bit más significativo: negativo.

Origen del nombre:

La suma de un número de  $n\text{-bits}$  cualquiera, con su negativo de  $n\text{-bits}$ , en complemento a dos, da como resultado  $2^n$ . Por lo tanto, el complemento o inverso aditivo, de un número  $x$  es  $2^n - x$ , su complemento a dos.

# Complemento a 2

Para enteros de 8 bytes (8\*8 bits) tenemos:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	$0_{10}$
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001	$1_{10}$

# Complemento a 2

Para enteros de 8 bytes (8\*8 bits) tenemos:

01111111 11111111 11111111 11111111 11111111 11111111 11111111 11111110
9.223.372.036.854.775.806 <sub>10</sub> → Orden de magnitud? 10 <sup>18</sup> <sub>10</sub>
01111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
9.223.372.036.854.775.807 <sub>10</sub>
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
-9.223.372.036.854.775.808 <sub>10</sub>
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
-9.223.372.036.854.775.807 <sub>10</sub>

# Complemento a 2

Para enteros de 8 bytes (8\*8 bits) tenemos:

11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111110	- $2_{10}$
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111	- $1_{10}$
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	$0_{10}$
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001	$1_{10}$


# Suma

Para simplificar el procedimiento, lo haremos con enteros de un byte (8 bits).

Supongamos que queremos sumar  $15_{10} + 21_{10}$  en el computador:

$$- 15_{10} = 00001111_2 ; 21_{10} = 00010101_2$$

Carry	0	0	1	1	1	1	1	
Sumando	0	0	0	0	1	1	1	1
Sumando	0	0	0	1	0	1	0	1
Suma	0	0	1	0	0	1	0	0



# Suma

## Resultado

$$15_{10} + 21_{10} = 36_{10}$$

$$00001111_2 + 00010101_2 = 00100100_2$$

$$00100100_2 =$$

$$\begin{aligned} &0*2^8 + 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 \\ &+ 1*2^2 + 0*2^1 + 0*2^0 = 36_{10} \end{aligned}$$



# Inverso aditivo

Cómo podemos saber, para un número  $x_2$  de  $n$ -bits, cuál es su inverso aditivo?

Dado que estamos en complemento a dos, basta negar cada bit y sumar 1:

00000011 11001001	$969_{10}$
11111100 00110110	$-970_{10}$
11111100 00110110 + 00000000 00000001	

# Inverso aditivo

Cómo podemos saber, para un número  $x_2$  de  $n$ -bits, cuál es su inverso aditivo?

Dado que estamos en complemento a dos, basta negar cada bit y sumar 1:

00000011 11001001	$969_{10}$
11111100 00110110	$-970_{10}$
11111100 00110111	$-969_{10}$

$(*^{\wedge}-^{\wedge}*)$

# Resta

A través del complemento, tenemos que:

$$- \quad 0011101_2 + 11110011_2 = ??$$

Carry								
Sumando	0	0	0	1	1	1	0	1
Sumando	1	1	1	1	0	0	1	1
Suma								

Donde  $11110011_2$  es el inverso aditivo de  $00001101_2$  en complemento a 2

# Resta

A través del complemento, tenemos que:

$$\begin{aligned} - \quad 0011101_2 + 11110011_2 &= 00010000_2 = 0011101_2 - 00001101_2 \\ &= 16_{10} = 29_{10} - 13_{10} \end{aligned}$$

Carry	1	1	1	1	1	1	1	
Sumando	0	0	0	1	1	1	0	1
Sumando	1	1	1	1	0	0	1	1
Suma	0	0	0	1	0	0	0	0

Esto permite simplificar considerablemente el hardware, ya que solo necesitamos poder sumar.

# OVERFLOW

(°—° // )

# Overflow

Supongamos, nuevamente, que estamos trabajando con complemento a 2, y solo disponemos de registros de 1 byte (8 bits) para cada número.

Qué ocurre si sumamos  $105_{10} + 60_{10}$ ?

Tenemos la suma  $01101001_2 + 00111100_2 = 10100101_2$

Dado que estamos operando con complemento a 2, cuál sería el valor decimal de dicho número?

$$10100101_2 = -91_{10} \quad ??$$

(´·ω·`)?

# Overflow

Supongamos, nuevamente, que estamos trabajando con complemento a 2, y solo disponemos de registros de 1 byte (8 bits) para cada número.

Qué ocurre si sumamos  $105_{10} + 60_{10}$ ?

Tenemos la suma  $01101001_2 + 00111100_2 = 10100101_2$

Dado que estamos operando con complemento a 2, cuál sería el valor decimal de dicho número?

$$10100101_2 = -91_{10}$$

En complemento a 2, un 1 en el bit más significativo indica que es un número negativo .

# Overflow

Supongamos, nuevamente, que estamos trabajando con complemento a 2, y solo disponemos de registros de 1 byte (8 bits) para cada número.

Qué ocurre si sumamos  $105_{10} + 60_{10}$ ?

Tenemos la suma  $01101001_2 + 00111100_2 = 10100101_2$

Dado que estamos operando con complemento a 2, cuál sería el valor decimal de dicho número?

$$10100101_2 \rightarrow 01011010_2 \rightarrow 01011010_2 + 1_2 = 01011011_2 = -91$$



# Overflow

Entonces, tenemos que  $105_{10} + 60_{10}$ :

$$01101001_2 + 00111100_2 = 10100101_2 = -91_{10} ??$$

\\ (O\_o) /\

YOU DIED

.: ^ \ ( > \_ < ) ^ \ .:

# Overflow

Overflow ocurre cuando el resultado de una operación no puede ser representado con el hardware disponible.

En otras palabras, ocurre overflow cuando “nos pasamos de largo”, es decir, al sumar, restar, o hacer alguna otra operación, nos quedamos sin bits para representar el resultado o la representación del resultado no es la que esperábamos.

Al computador NO le importa, ni se va a dar cuenta, que ha ocurrido overflow.

ಥ\_ಥ

Es responsabilidad de ustedes como programadores, hacerse cargo de estas situaciones.

o((>ω< ))o

# Overflow

Pero, no hubo overflow cuando restamos  $29_{10} - 13_{10}$  ?

# Overflow

Pero, no hubo overflow cuando restamos  $29_{10} - 13_{10}$  ?

Carry	1	1	1	1	1	1	1	1	
Sumando	?	0	0	0	1	1	1	0	1
Sumando	?	1	1	1	1	0	0	1	1
Suma	1 (?)	0	0	0	1	0	0	0	0

Sí, pero en realidad no. Como estábamos sumando una representación negativa con una positiva, el resultado es el número esperado, y el carry simplemente se desecha.

Volviendo a la definición, el hardware disponible es suficiente para representar correctamente el resultado de dicha operación

# Overflow

Por lo tanto, si bien puede ser que nos “sobre” un bit, esto no siempre causa overflow.

Cuándo ocurre overflow?

Operación	Operando A	Operando B	Resultado indicando overflow
A + B	$\geq 0$	$\geq 0$	$< 0$
A + B	$< 0$	$< 0$	$\geq 0$
A - B	$\geq 0$	$< 0$	$< 0$
A - B	$< 0$	$\geq 0$	$\geq 0$

# Decoders

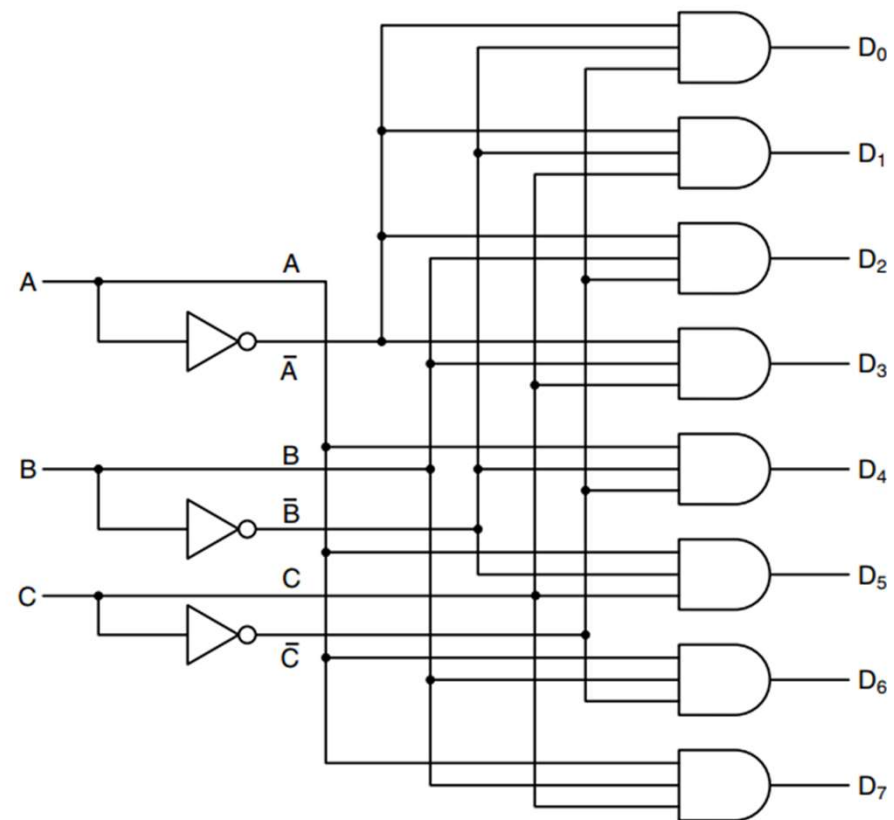
Es un circuito que posee  $n$  inputs y a lo más  $2^n$  outputs, y utiliza los inputs para seleccionar alguno(s) de los outputs.



# Decoders

Un decoder puede ser de tipo one-hot, en que se selecciona únicamente uno de sus outputs.

i1	i2	d3	d2	d1	d0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



**Figure 3-13.** A three-to-eight decoder circuit.

# Enablers

Son señales que nos permiten seleccionar si se deja pasar un input, o se fuerza a que sea 0 o 1.

En el caso de un decoder, también permiten forzar un estado, i.e. permitir que el decoder funcione normalmente o que emita permanentemente un estado.

# Enabler

Al agregar un enabler a un decoder, podemos tener dos casos:

- Active high, si el enabler es 1, se genera un output, en cambio, si es 0, el output queda fijo en 0 para todas las salidas.
- Active low, si el enabler es 0, se genera un output, en cambio, si es 1, se genera un output fijo.

## Decoder one-hot con enabler

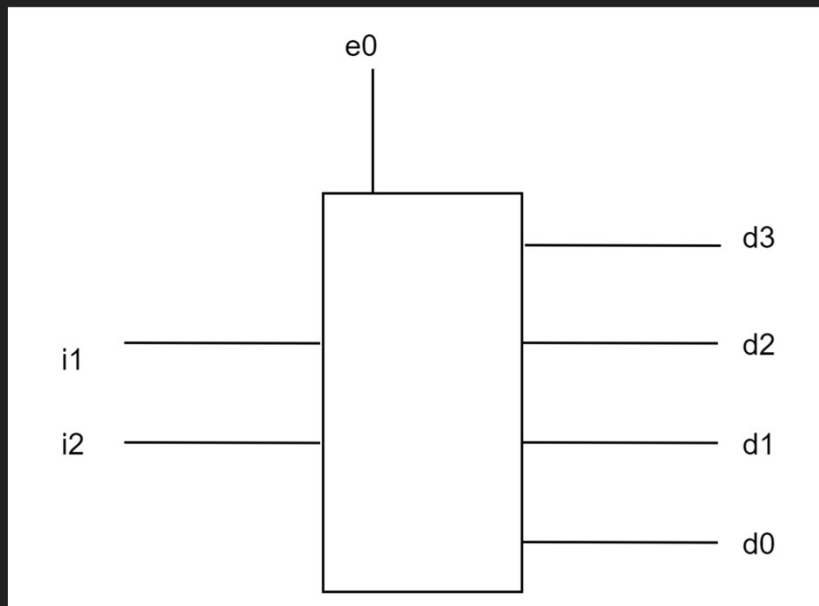


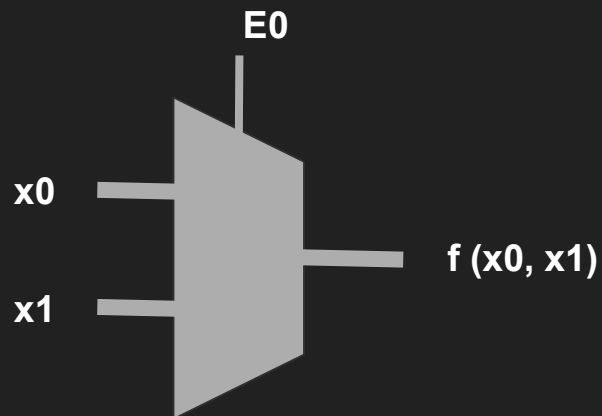
Tabla de valores para el decoder de la ilustración:

e0	i1	i2	d3	d2	d1	d0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	i1	i2	0	0	0	0

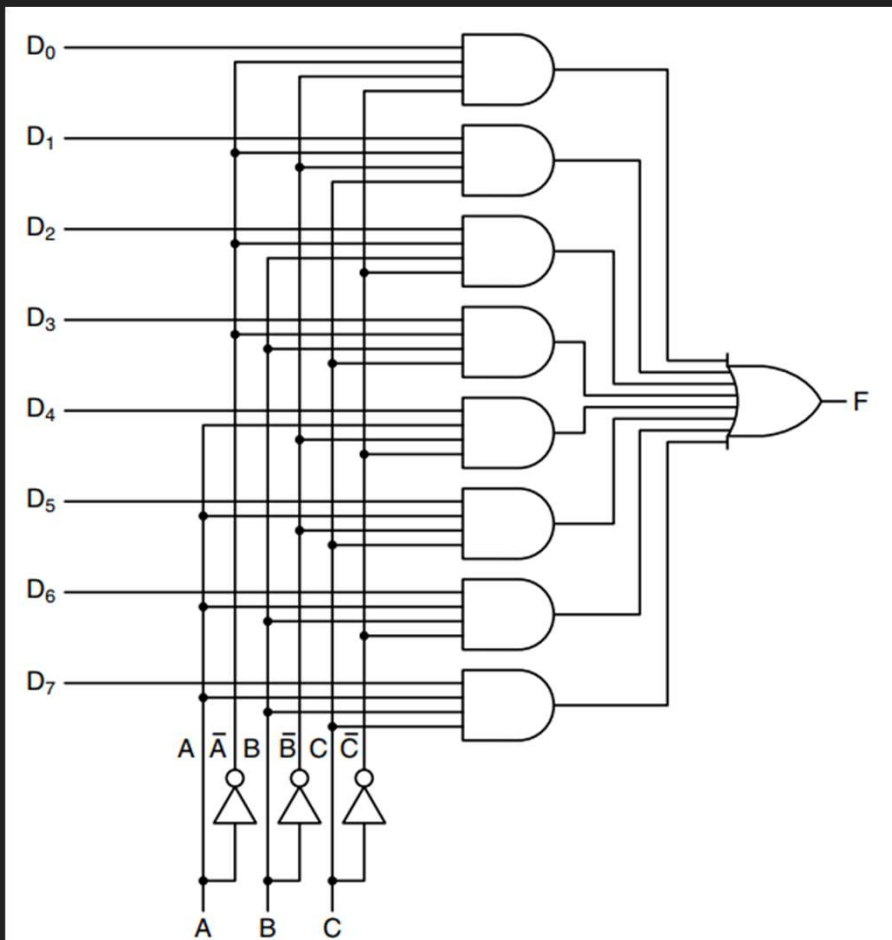
# Multiplexores

Permiten convertir un número  $n$  de inputs a un único output\*. En otras palabras, permite seleccionar una única entrada y ponerla como salida.

\*: el output puede ser un bus de datos (más de un bit de salida).



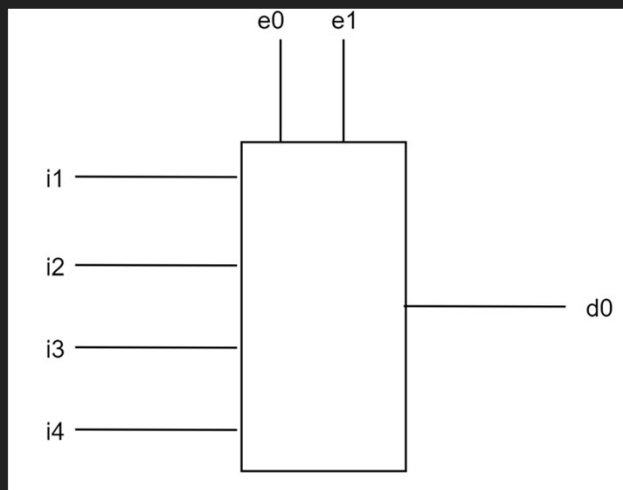
$E_0$	$f(x_0, x_1)$
0	$x_0$
1	$x_1$



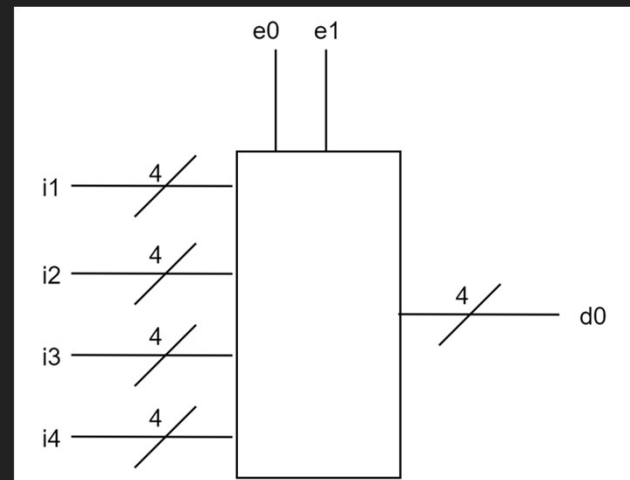
**Figure 3-11.** An eight-input multiplexer circuit.

Structured Computer  
Organization - Andrew S.  
Tanenbaum

# Multiplexores



4 Input Mux



4 Input Mux, 4 bit bus



# Multiplexores

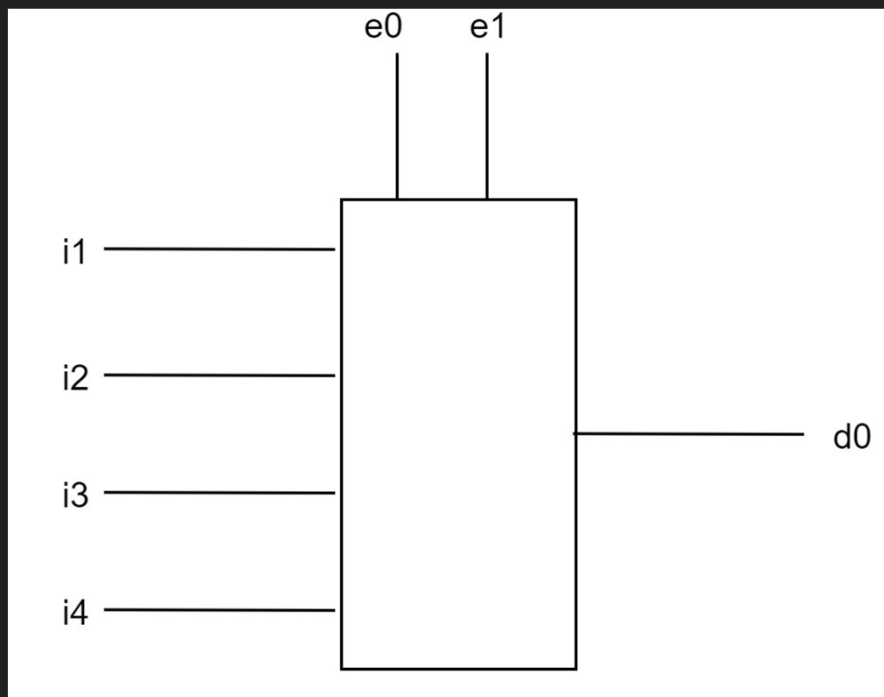


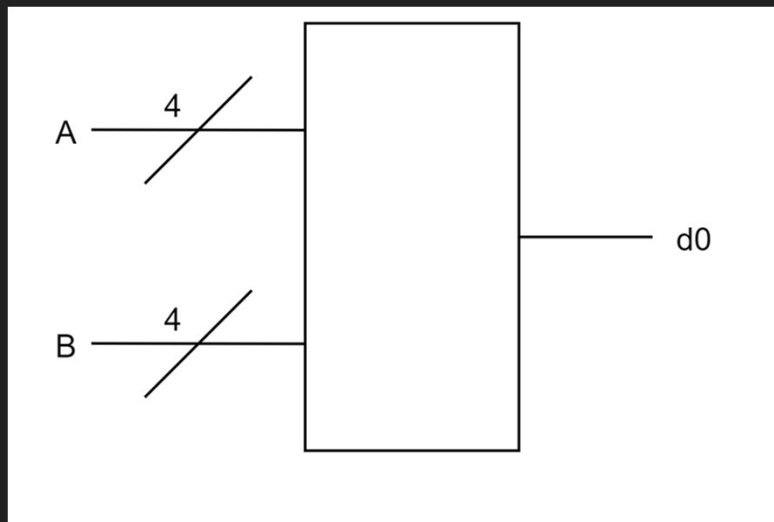
Tabla de valores para el  
Mux de la ilustración:

e0	e1	d0
0	0	i1
0	1	i2
1	0	i3
1	1	i4

# Comparadores

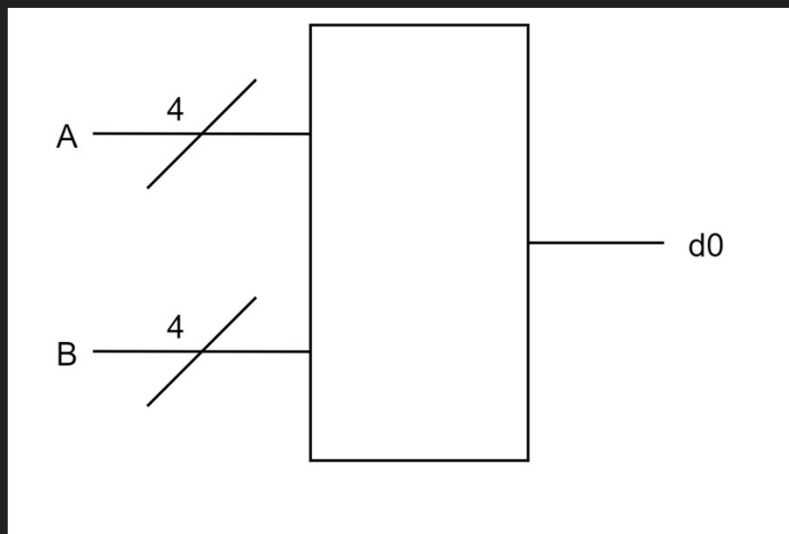
Compara dos patrones de bits del mismo largo y emite como output 1 si ambos patrones son iguales y 0 en cualquier otro caso.

# Comparadores



Para este comparador, tenemos dos buses de 4 bits de entrada y un bit de salida.

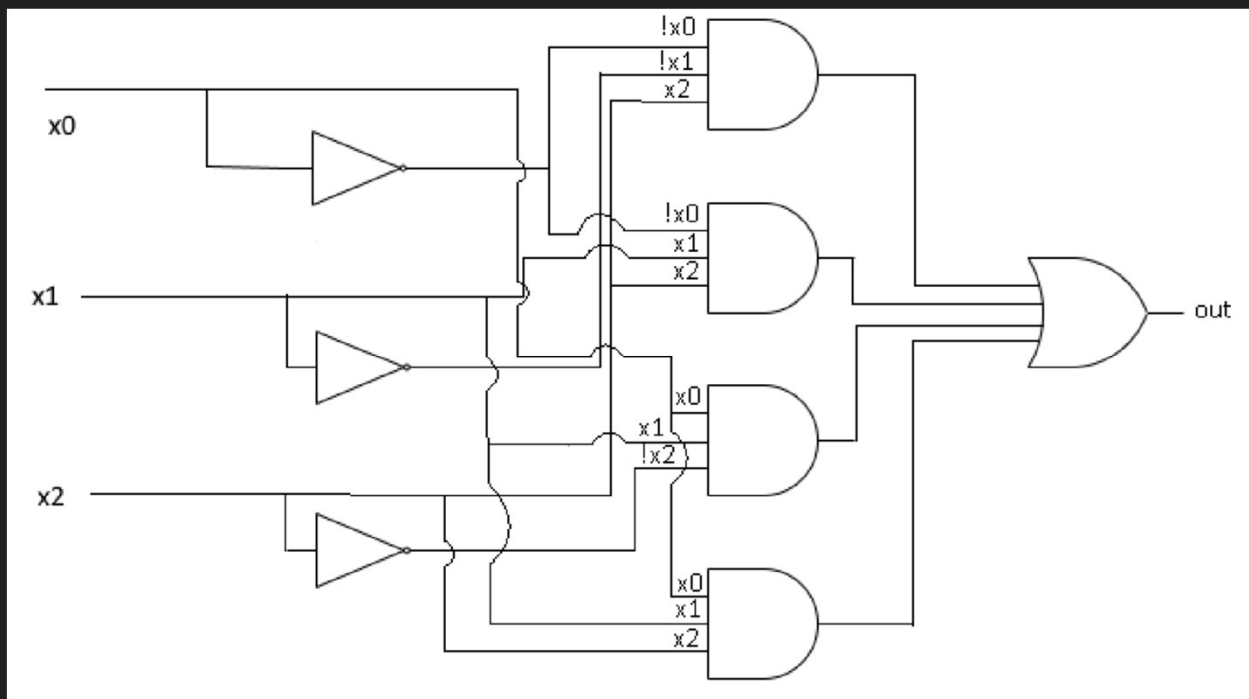
# Comparadores



Para este comparador, tenemos dos buses de 4 bits de entrada.

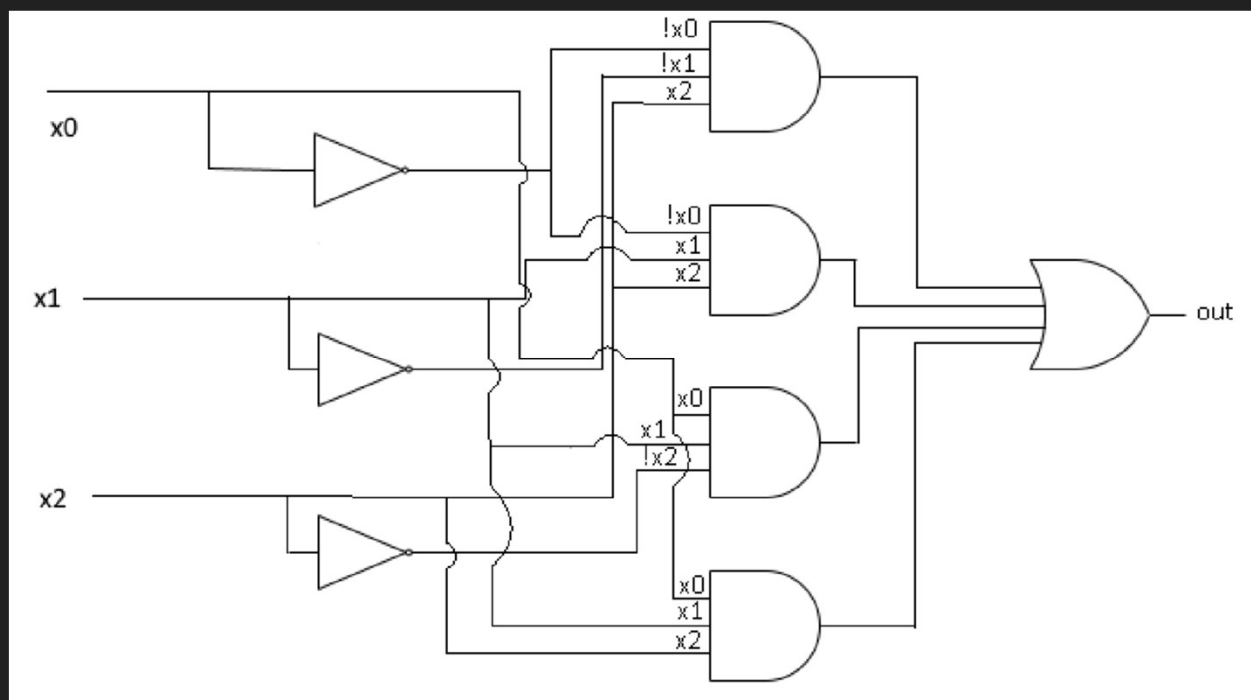
- $d0 = 1$  ssi.  $A = B$ ; 0 en otro caso.

## Ejercicio: complete la tabla de valores



$x_0$	$x_1$	$x_2$	out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

# Ejercicio



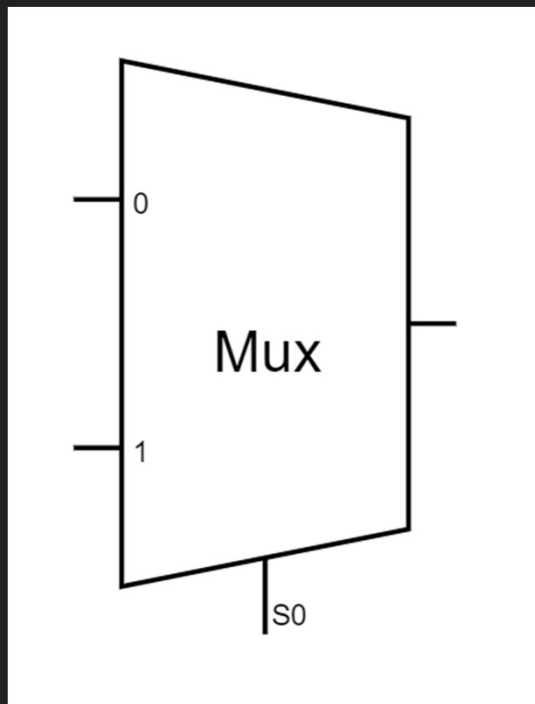
$x_0$	$x_1$	$x_2$	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

## Ejercicio

Se les ocurre cómo simplificar el circuito anterior?

Aparte de ordenar los cables  $O(T \wedge T_0)$

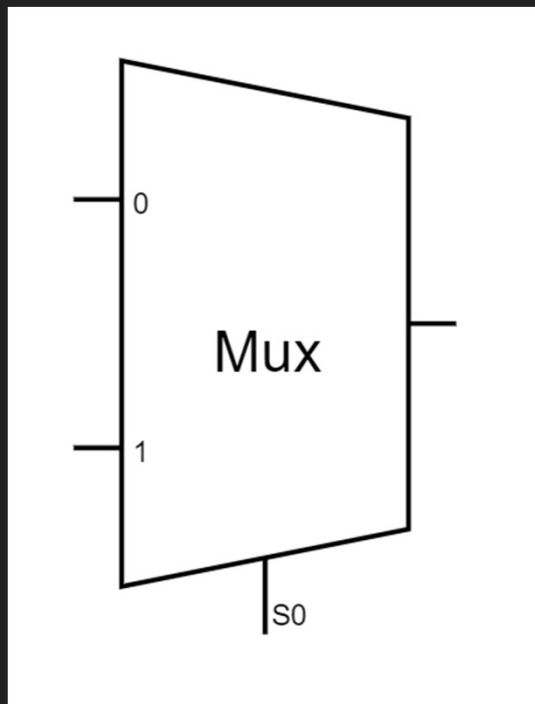
## Ejercicio



S0	i0	i1	out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



## Ejercicio



S0	i0	i1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

x0	x1	x2	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S0	i0	i1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

x0	x1	x2	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

==

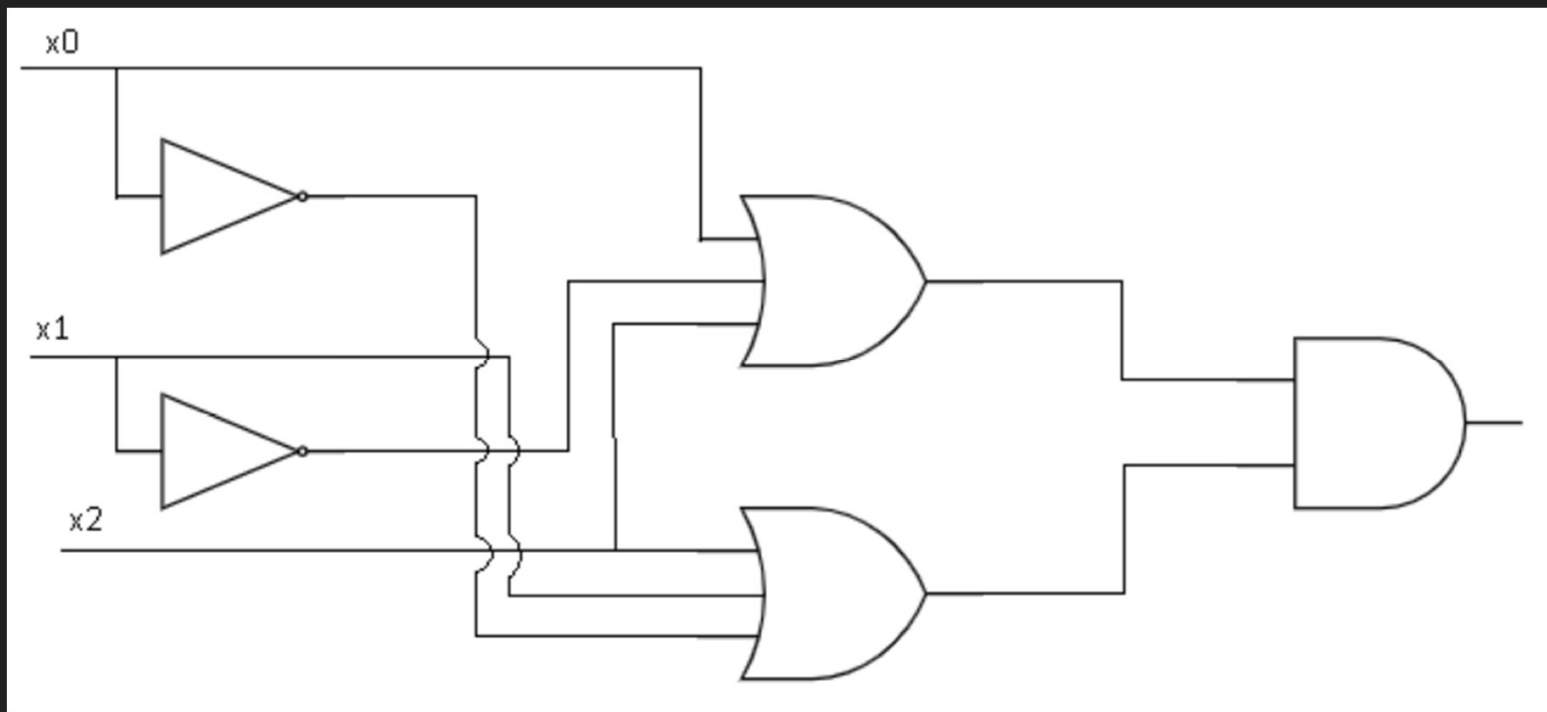
S0	i0	i1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

## Ejercicio propuesto

Dibuje el circuito que corresponde a la siguiente tabla de verdad:

x0	x1	x2	out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Ejercicio propuesto, una solución:



Muchas gracias!

👋 (¯ ▽ ¯) Bye~Bye~