

Ayudantía 2

Aritmética, números y control digital

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

Hecha por:

- Tomás Contreras
- Susana Figueroa
- Andrés González
- Laurence Golborne

Temas:

- Complemento a 2
- Suma en complemento a 2
- Inverso aditivo
- Resta en complemento a 2
- Overflow
- Multiplexores
- Ejercicios

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 ₁₀ → Orden de magnitud? 10 ¹⁸ ₁₀
01111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
9.223.372.036.854.775.807 ₁₀
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
-9.223.372.036.854.775.808 ₁₀
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
-9.223.372.036.854.775.807 ₁₀

Complemento a 2

Para enteros de 8 bytes (8×8 bits) tenemos:

[illegible]

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:

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


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}

Resta

A través del complemento, tenemos que:

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

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:

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

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.

ಠ_ಠ

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

$o((\omega <))o$

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.

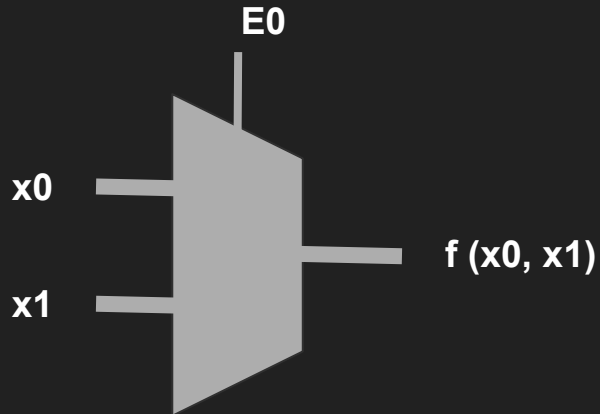
Cuándo ocurre overflow?

Operación	Operando A	Operando B	Resultado indicando overflow
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A - B	≥ 0	< 0	< 0
A - B	< 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

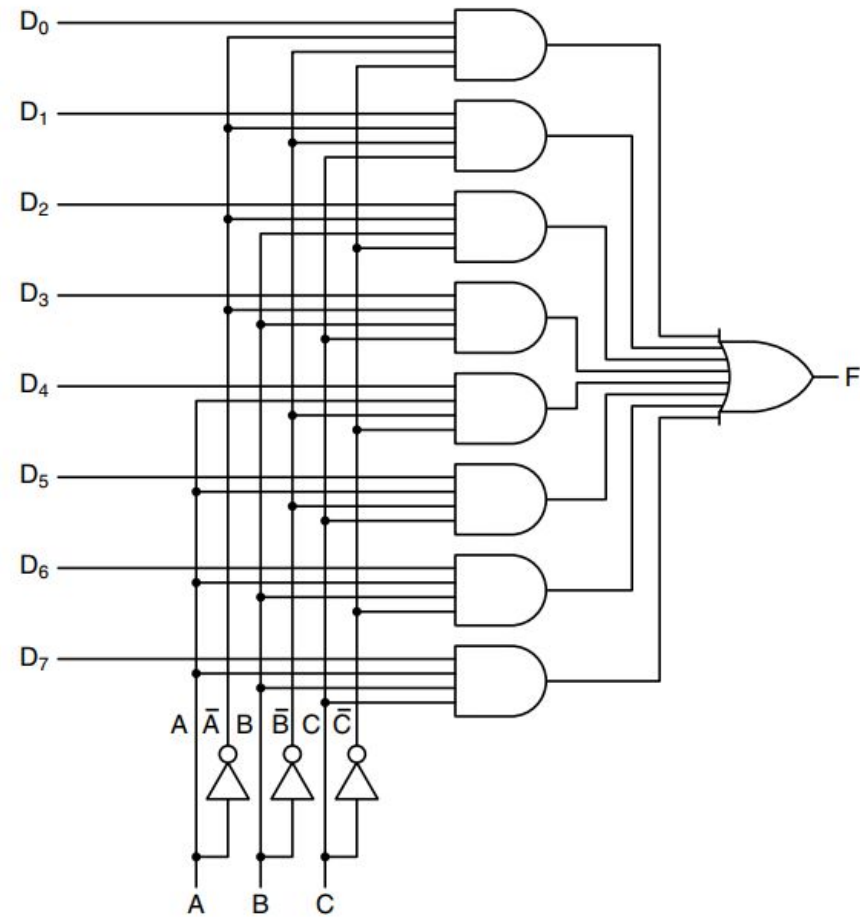
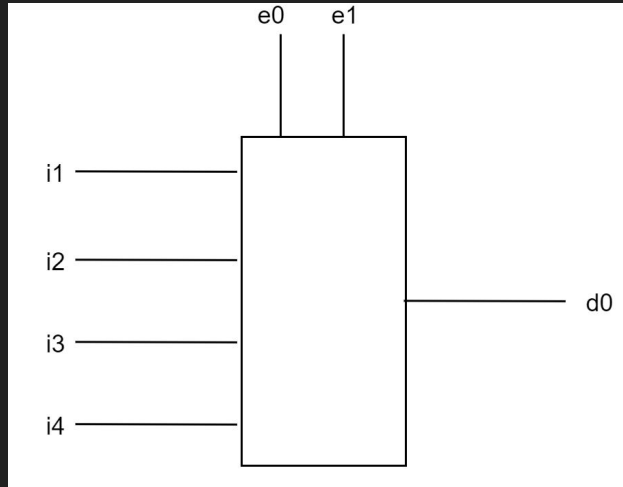
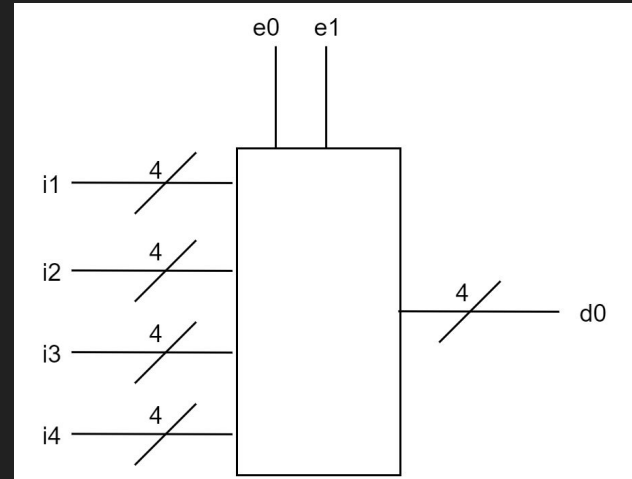


Figure 3-11. An eight-input multiplexer circuit.

Multiplexores



4 Input Mux



4 Input Mux, 4 bit bus

Multiplexores

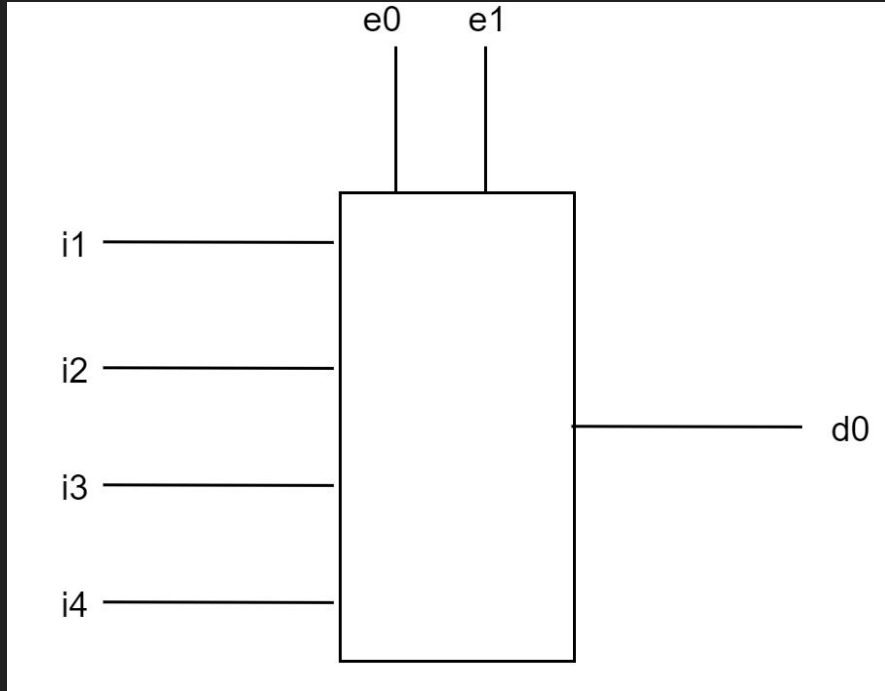
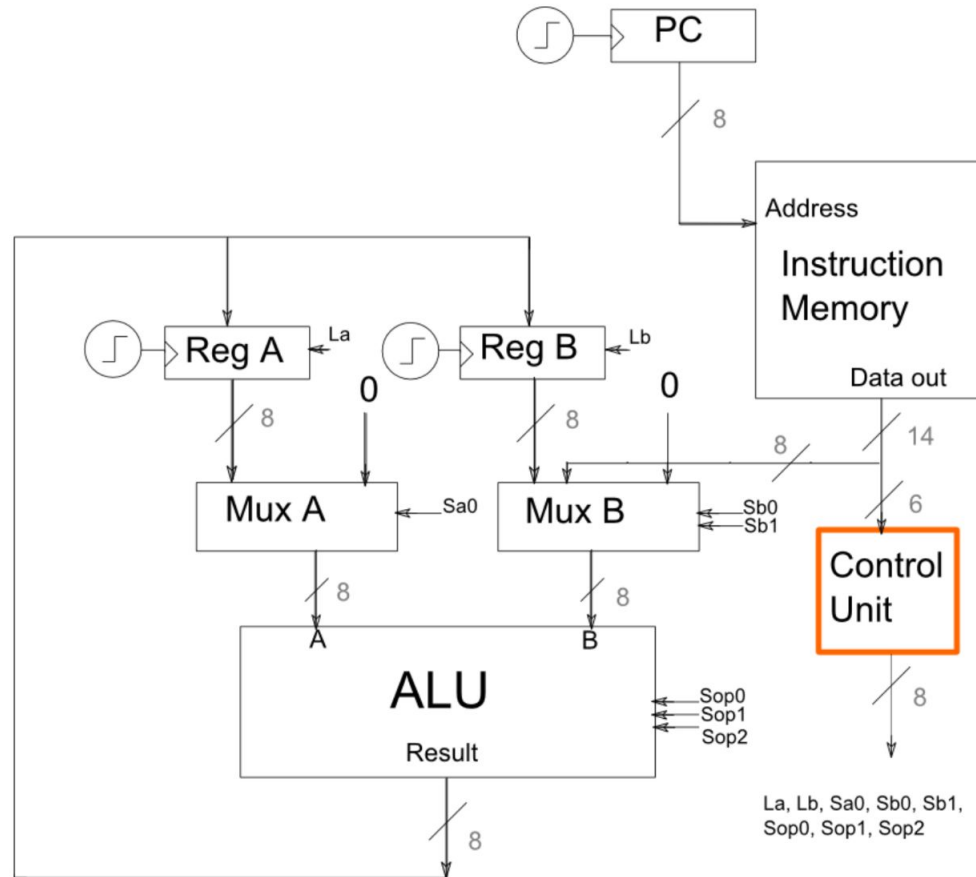


Tabla de valores para el
Mux de la ilustración:

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

Componentes del computador



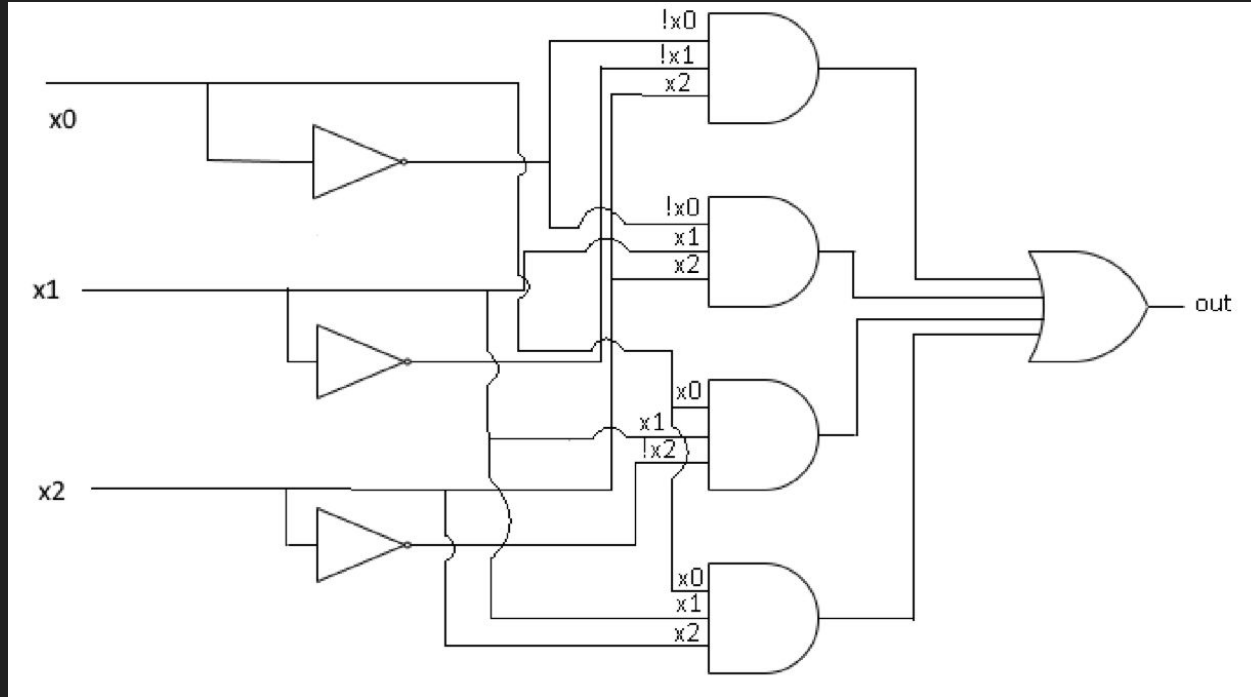
Ejercicio

Considera la siguiente versión del computador básico y su correspondiente set de instrucciones (página 3). Queremos agregar un tercer registro, *C*, al computador básico, y las correspondientes instrucciones **ADD A,C** y **MOV C,B**.

- Dibuja y explica de manera clara y precisa los cambios necesarios en el circuito del computador básico: destaca claramente componentes, cables, buses de datos nuevos/modificados y cualquier otro cambio que hagas. La ALU no cambia en cuanto a que sigue teniendo dos inputs, *A* y *B*, y un output, *Result*, y que todos son de 8 bits.
- Especifica las nuevas instrucciones en cuanto a opcodes, señales de control, y operación, similarmente a las otras instrucciones en el set y de manera coherente con tu dibujo en a).

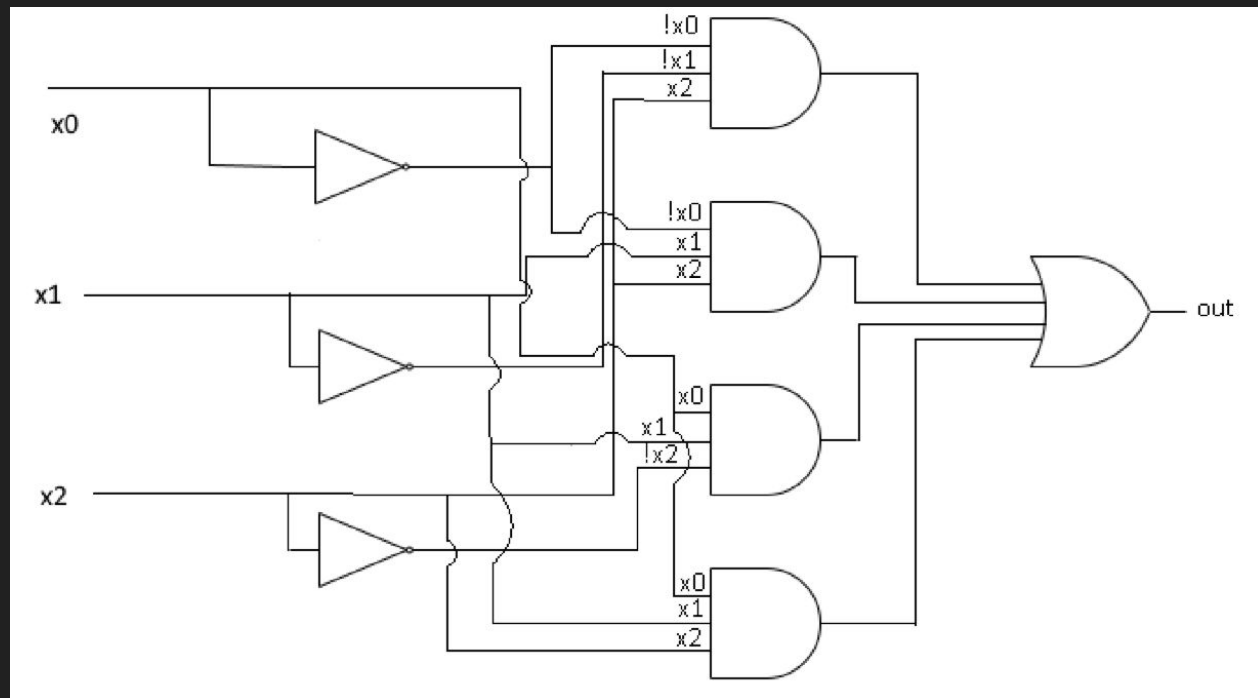
Instrucción	Operandos	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
MOV	A,B	000000	1	0	1	0	0	0	0	0	A=B
	B,A	000001	0	1	0	1	1	0	0	0	B=A
	A,Lit	000010	1	0	0	0	1	0	0	0	A=Lit
	B,Lit	000011	0	1	0	0	1	0	0	0	B=Lit
ADD	A,B	000100	1	0	0	0	0	0	0	0	A=A+B
	B,A	000101	0	1	0	0	0	0	0	0	B=A+B
	A,Lit	000110	1	0	0	0	1	0	0	0	A=A+Lit
SUB	A,B	000111	1	0	0	0	0	0	0	1	A=A-B
	B,A	001000	0	1	0	0	0	0	0	1	B=A-B
	A,Lit	001001	1	0	0	0	1	0	0	1	A=A-Lit
AND	A,B	001010	1	0	0	0	0	0	1	0	A=A and B
	B,A	001011	0	1	0	0	0	0	1	0	B=A and B
	A,Lit	001100	1	0	0	0	1	0	1	0	A=A and Lit
OR	A,B	001101	1	0	0	0	0	0	1	1	A=A or B
	B,A	001110	0	1	0	0	0	0	1	1	B=A or B
	A,Lit	001111	1	0	0	0	1	0	1	1	A=A or Lit
NOT	A,A	010000	1	0	0	0	0	1	0	0	A=notA
	B,A	010001	0	1	0	0	0	1	0	0	B=notA
	A,Lit	010010	1	0	0	0	1	1	0	0	A=notLit
XOR	A,A	010011	1	0	0	0	0	1	0	1	A=A xor B
	B,A	010100	0	1	0	0	0	1	0	1	B=A xor B
	A,Lit	010101	1	0	0	0	1	1	0	1	A=A xor Lit
SHL	A,A	010110	1	0	0	0	0	1	1	0	A=shift left A
	B,A	010111	0	1	0	0	0	1	1	0	B=shift left A
	A,Lit	011000	1	0	0	0	1	1	1	0	A=shift left Lit
SHR	A,A	011001	1	0	0	0	0	1	1	1	A=shift right A
	B,A	011010	0	1	0	0	0	1	1	1	B=shift right A
	A,Lit	011011	1	0	0	0	1	1	1	1	A=shift right Lit

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



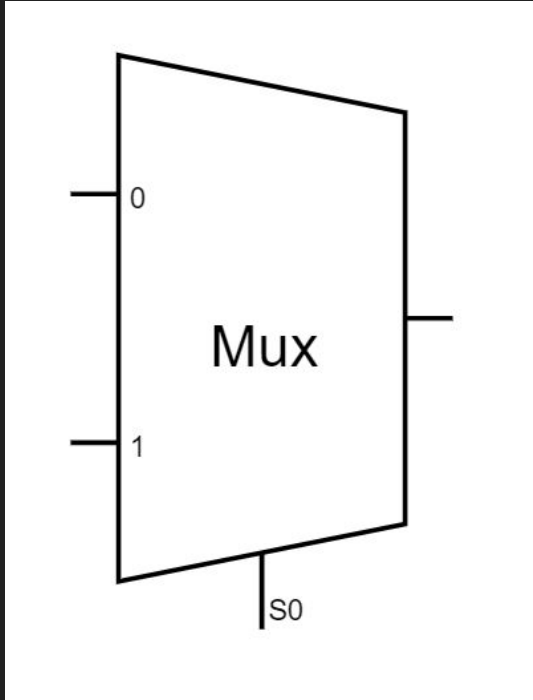
x_0	x_1	x_2	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Ejercicio

Se les ocurre cómo simplificar el circuito anterior?

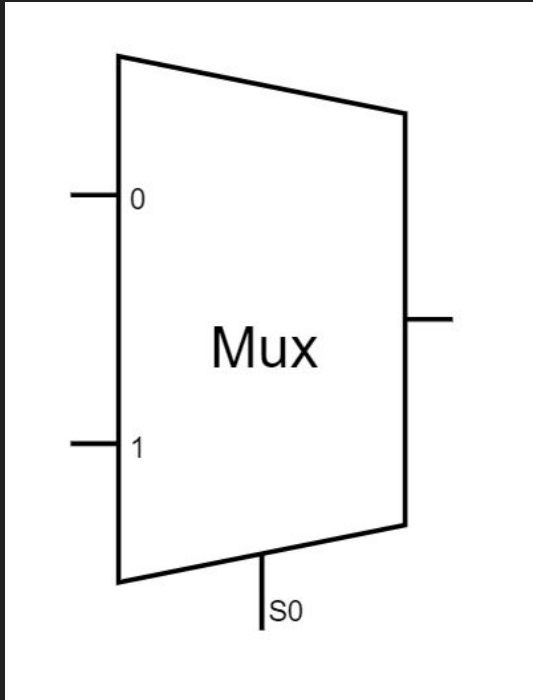
Aparte de ordenar los cables $o(T \wedge T_o)$

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

==

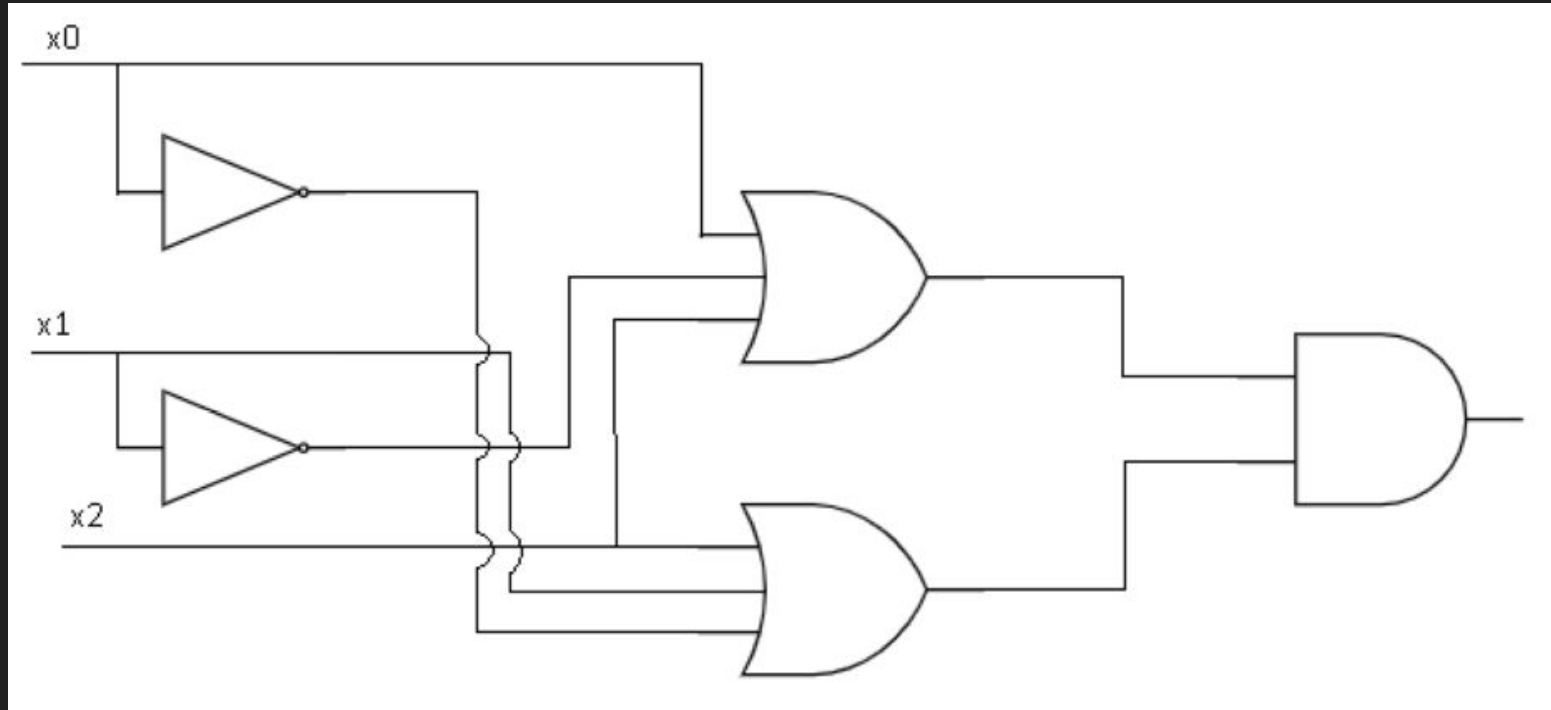
S0	i0	i1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

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!

ゞ(¯▽¯) Bye~Bye~