Tema 4:

Módulos combinacionales básicos

Fundamentos de computadores I

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

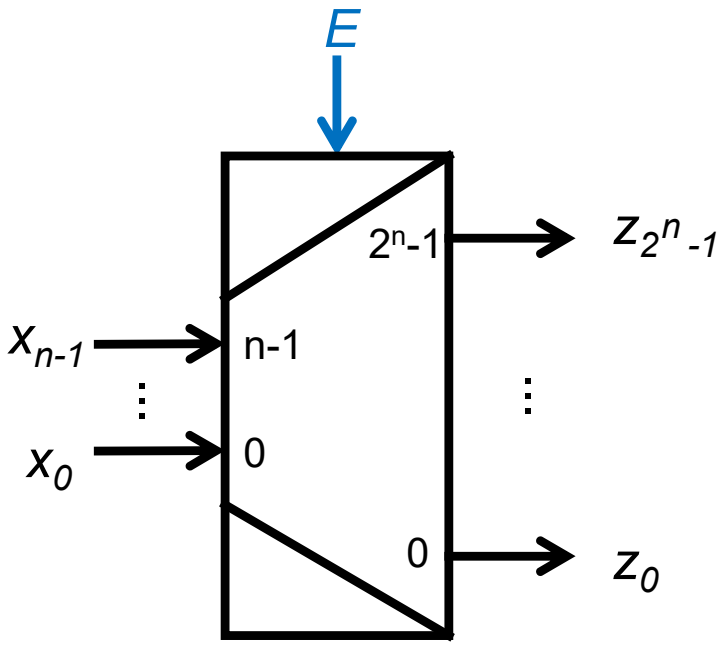
Contenidos

- ✓ Decodificador.
- ✓ Multiplexor.
- ✓ Bus.
- ✓ Codificador.
- ✓ ROM (Read Ony Memory).
- ✓ Comparador.
- ✓ Sumador/Restador.
- ✓ Extensor de signo.
- ✓ ALU (Arithmetic Logic Unit).
- ✓ Apéndice tecnológico.

Transparencias basadas en los libros:

- R. Hermida, F. Sánchez y E. del Corral. *Fundamentos de computadores.*
- D. Gajsky. *Principios de diseño digital.*

Decodificador



Decodificador n a 2^n

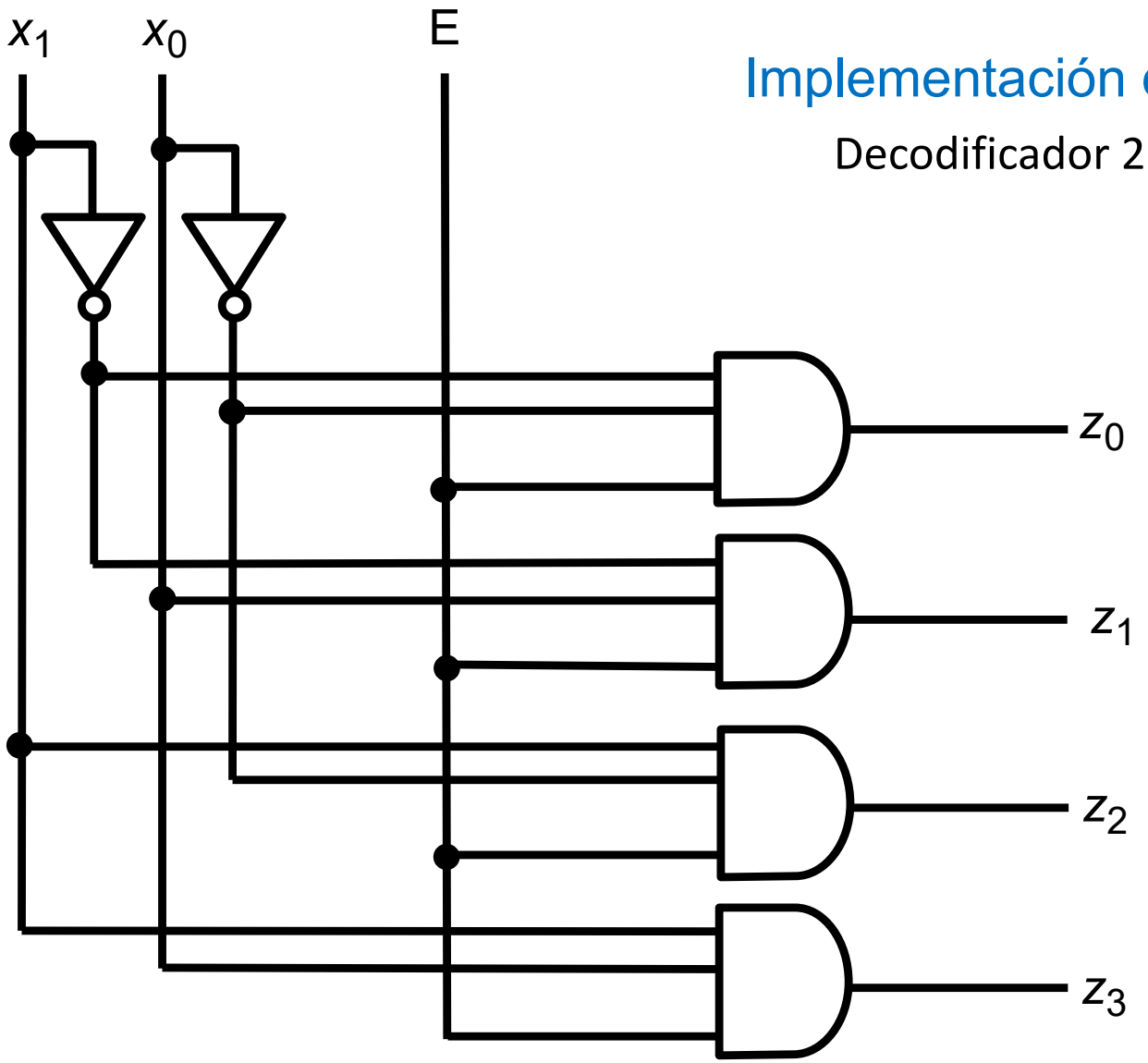
$\underline{x}$	n entradas de datos
$\underline{z}$	2^n salidas de datos
E	1 entrada de capacitación (op)

si la entrada toma la configuración binaria p, la salida $(p)_{10}$-ésima se activa

$$z_i = \begin{cases} 1 & \text{si } E=1 \text{ y } (\underline{x})_{10} = i \\ 0 & \text{en otro caso} \end{cases}$$

$$z_i = E \cdot m_i(\underline{x})$$

Decodificador



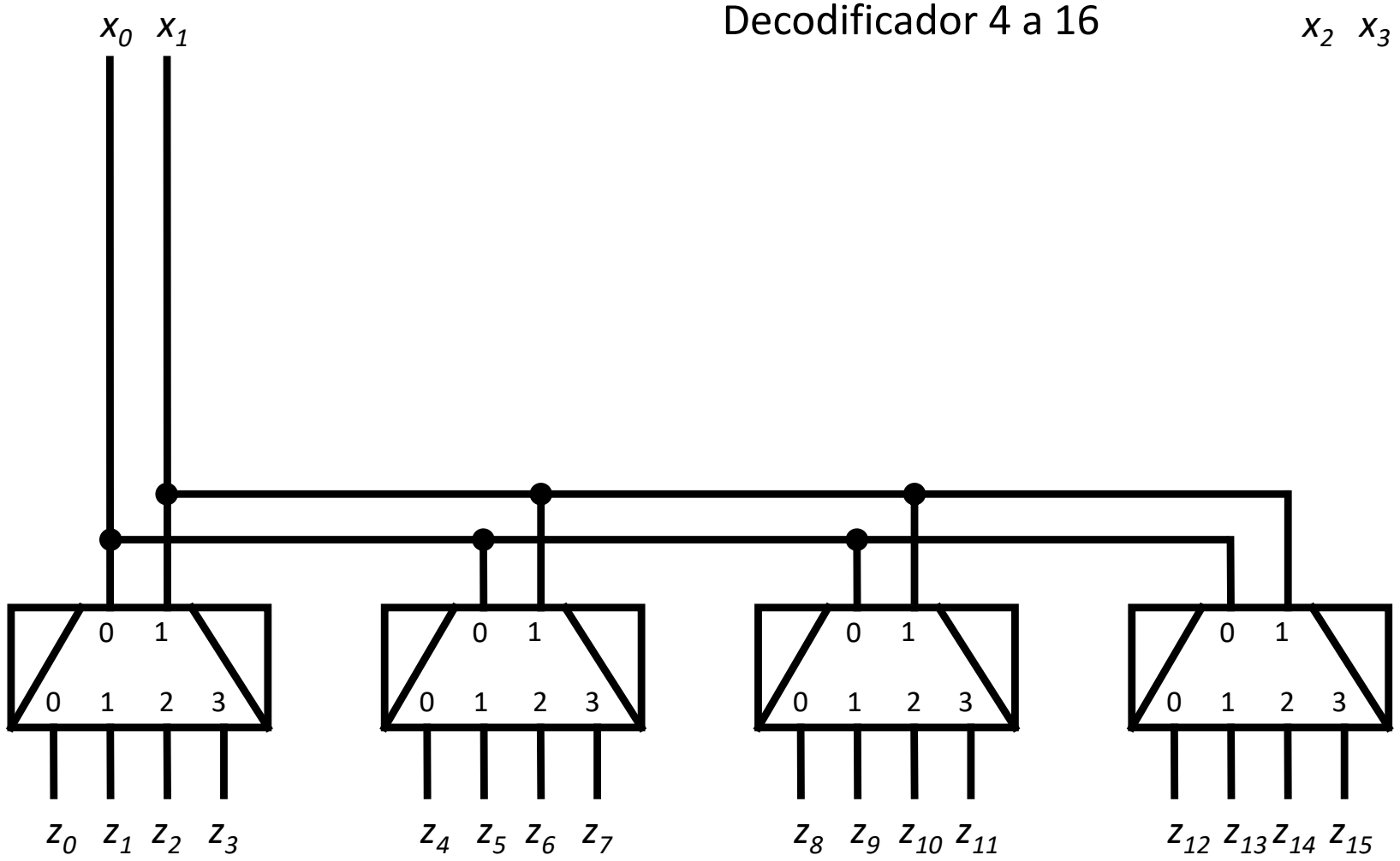
Implementación directa

Decodificador 2 a 4

Decodificador

Implementación en árbol

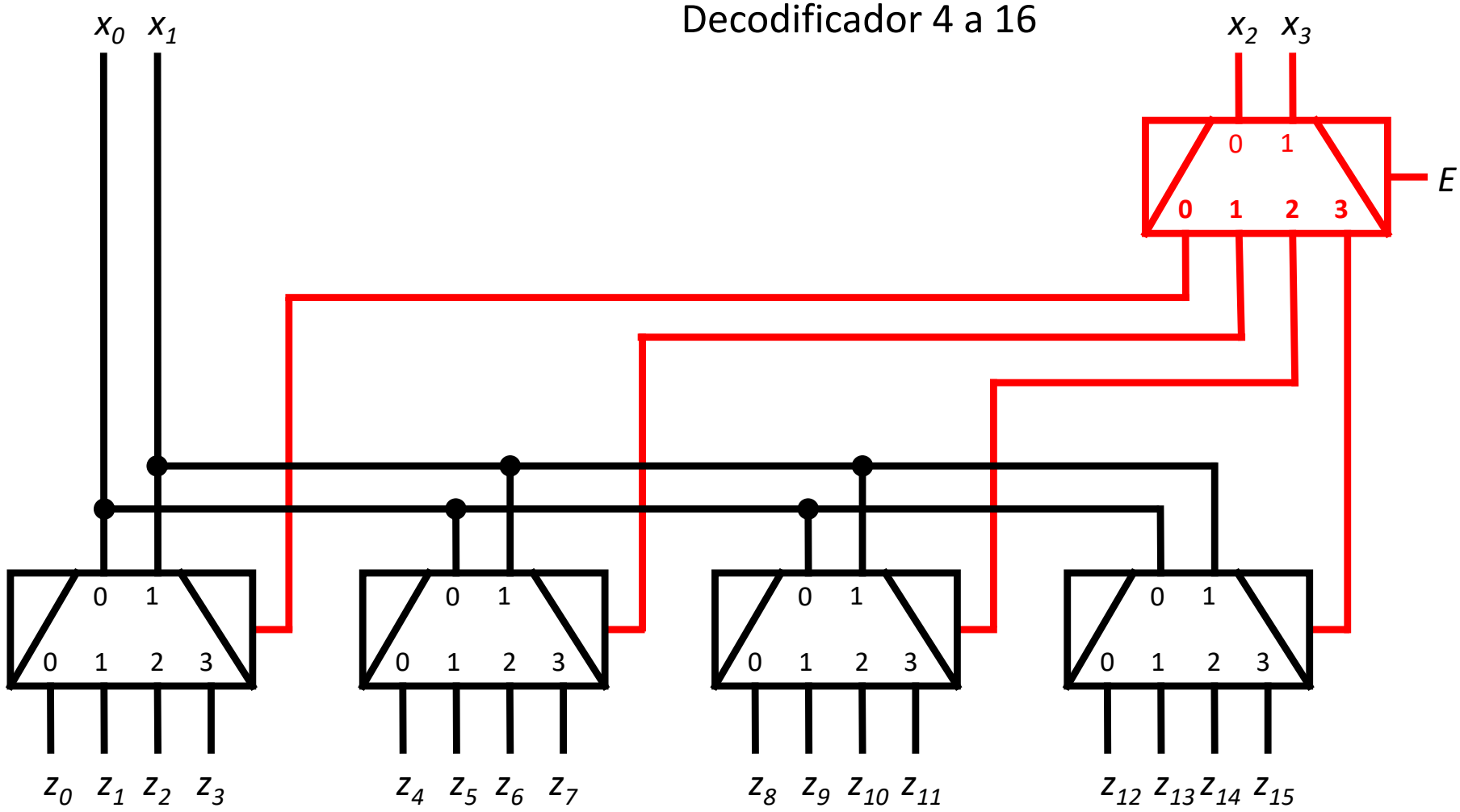
Decodificador 4 a 16



Decodificador

Implementación en árbol

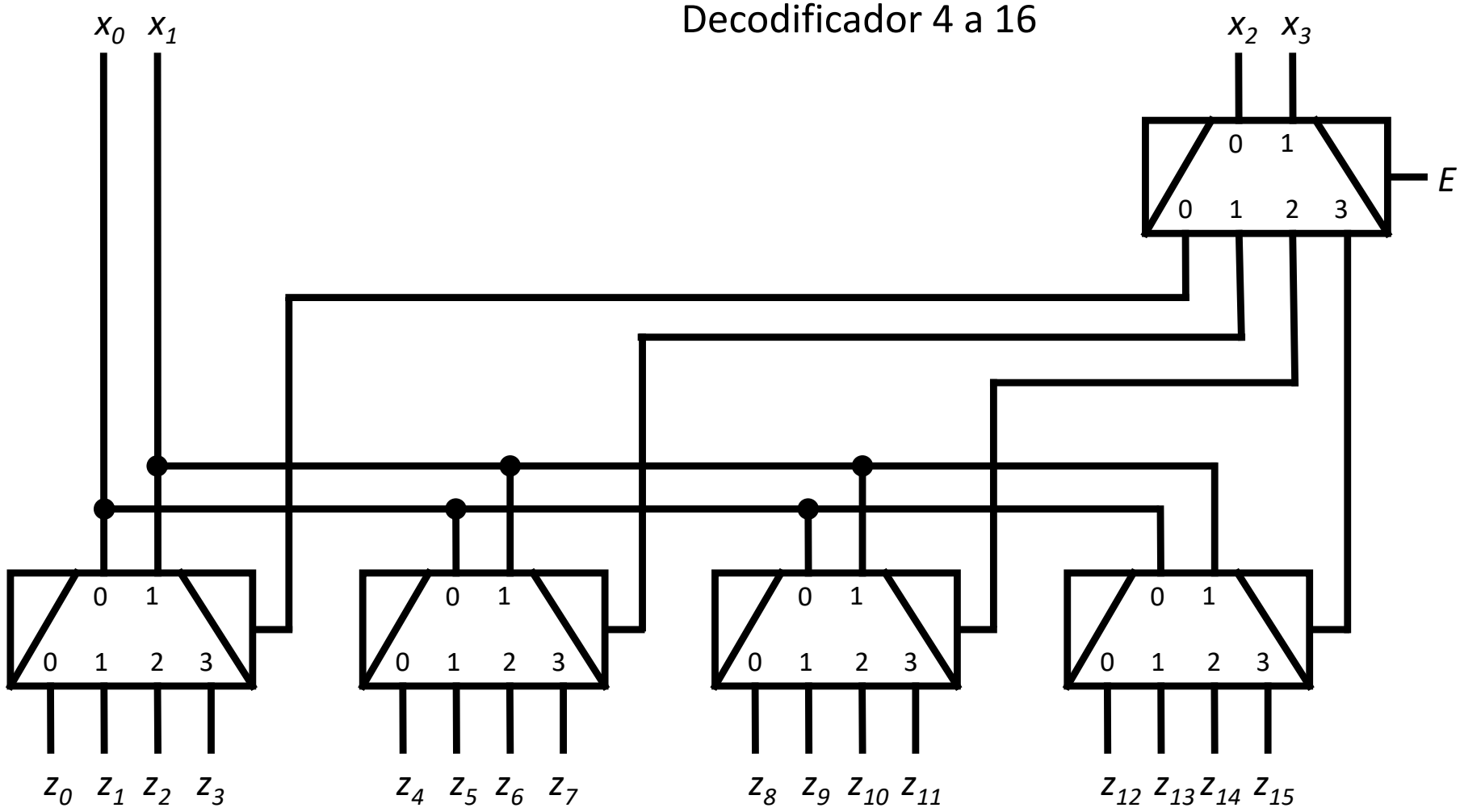
Decodificador 4 a 16



Decodificador

Implementación en árbol

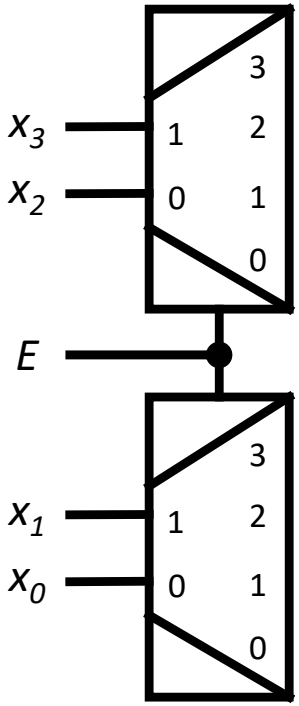
Decodificador 4 a 16

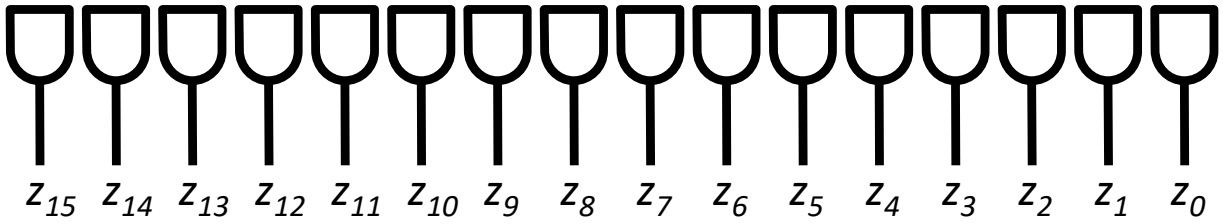


Decodificador

Implementación con predecodificación

Decodificador 4 a 16

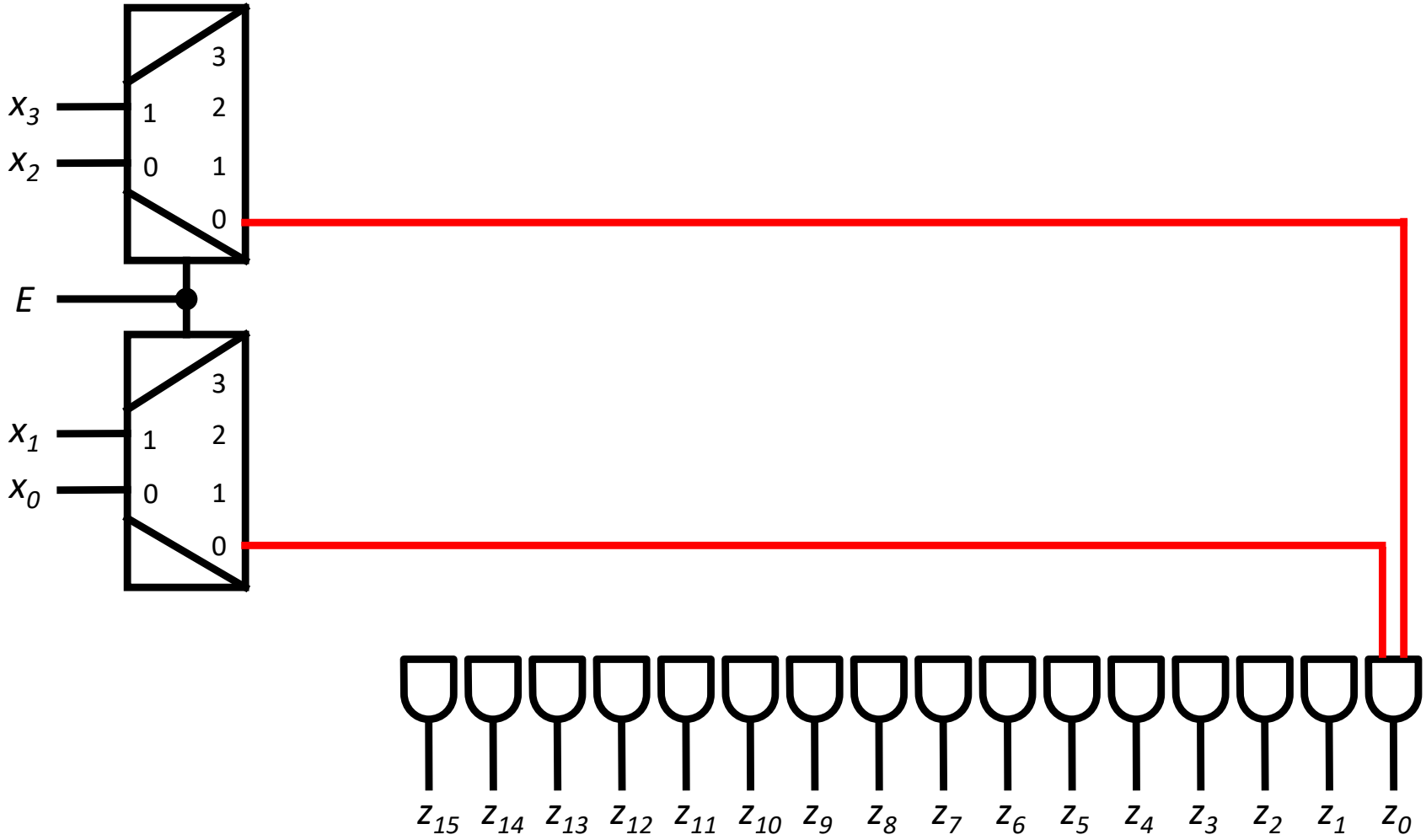




Decodificador

Implementación con predecodificación

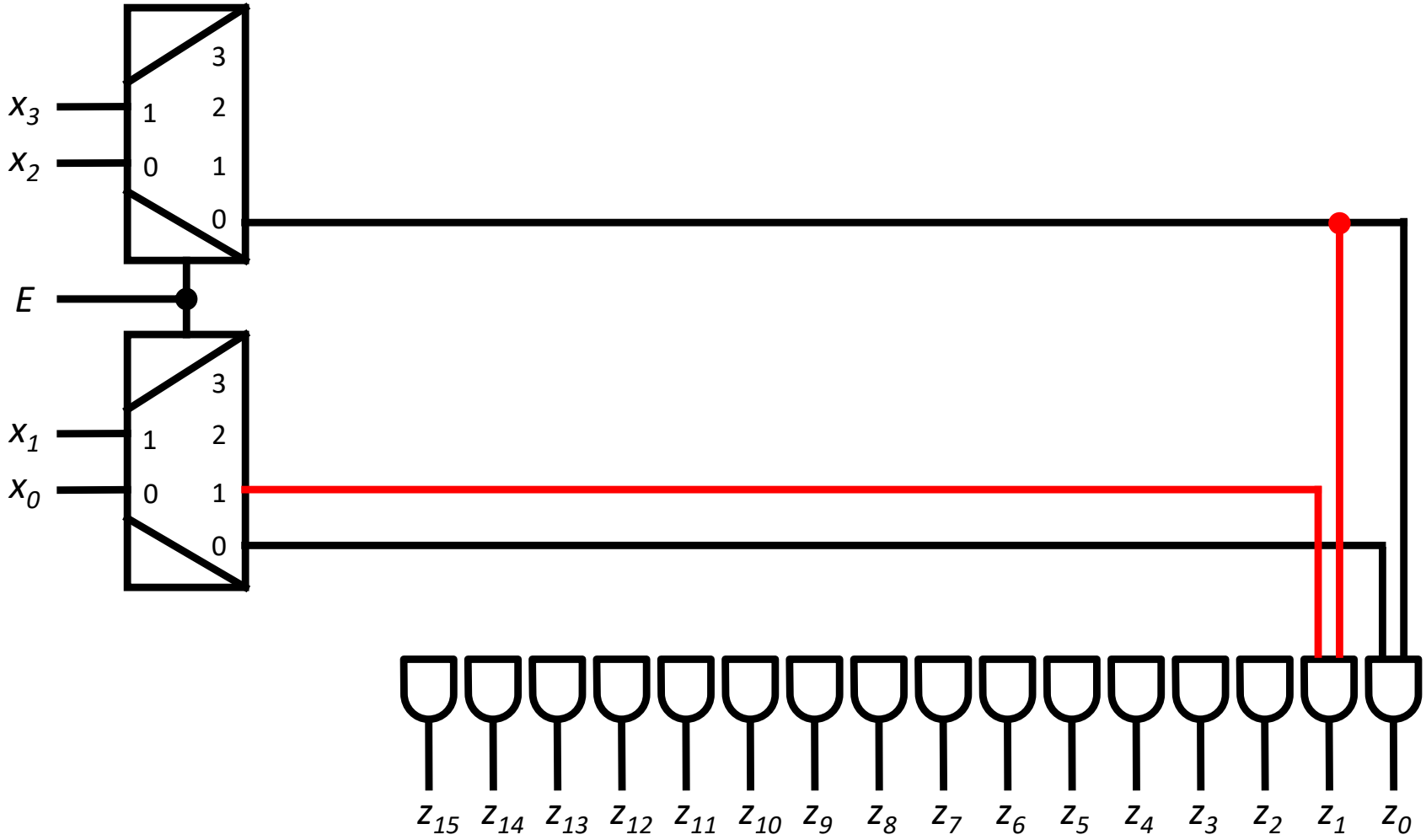
Decodificador 4 a 16



Decodificador

Implementación con predecodificación

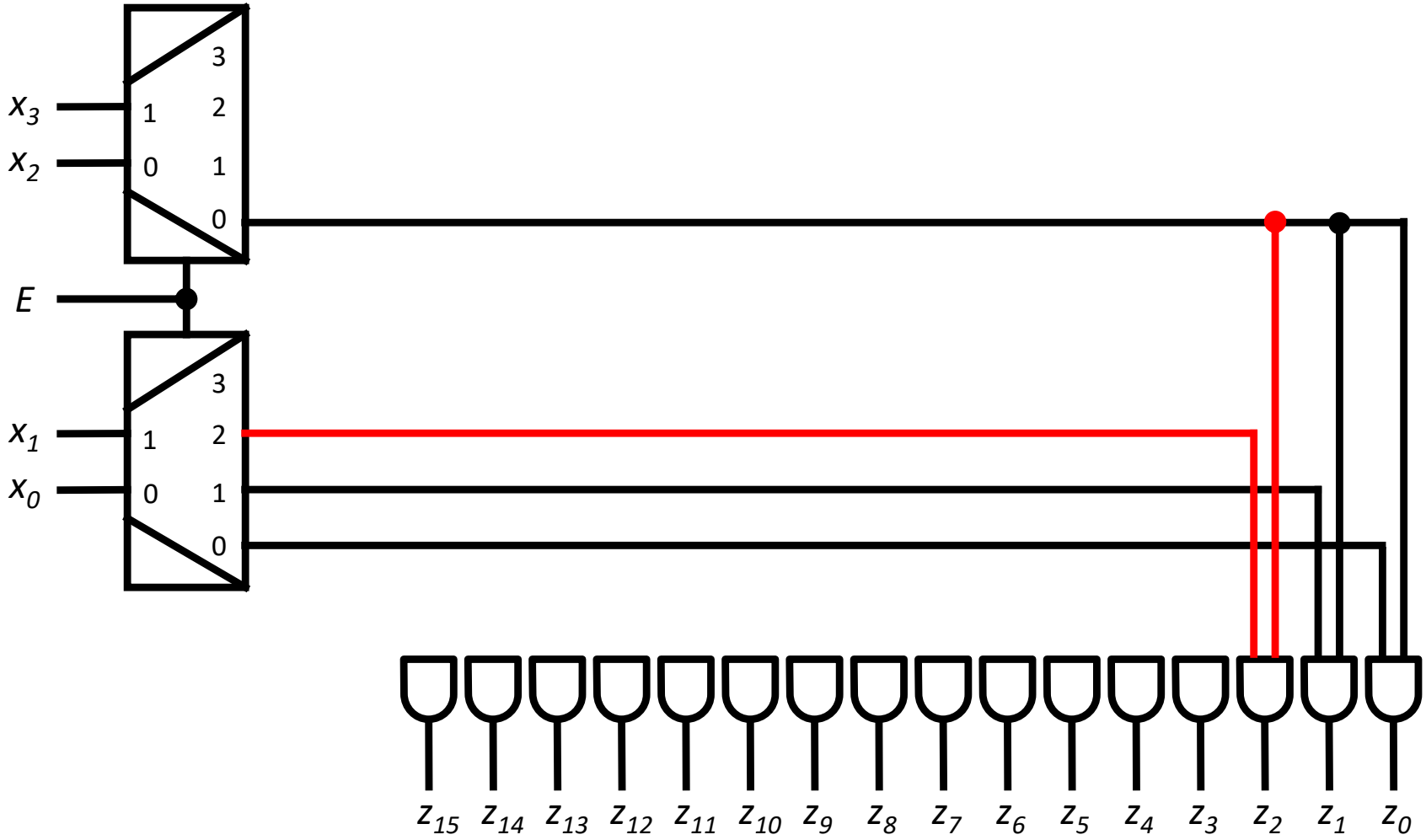
Decodificador 4 a 16



Decodificador

Implementación con predecodificación

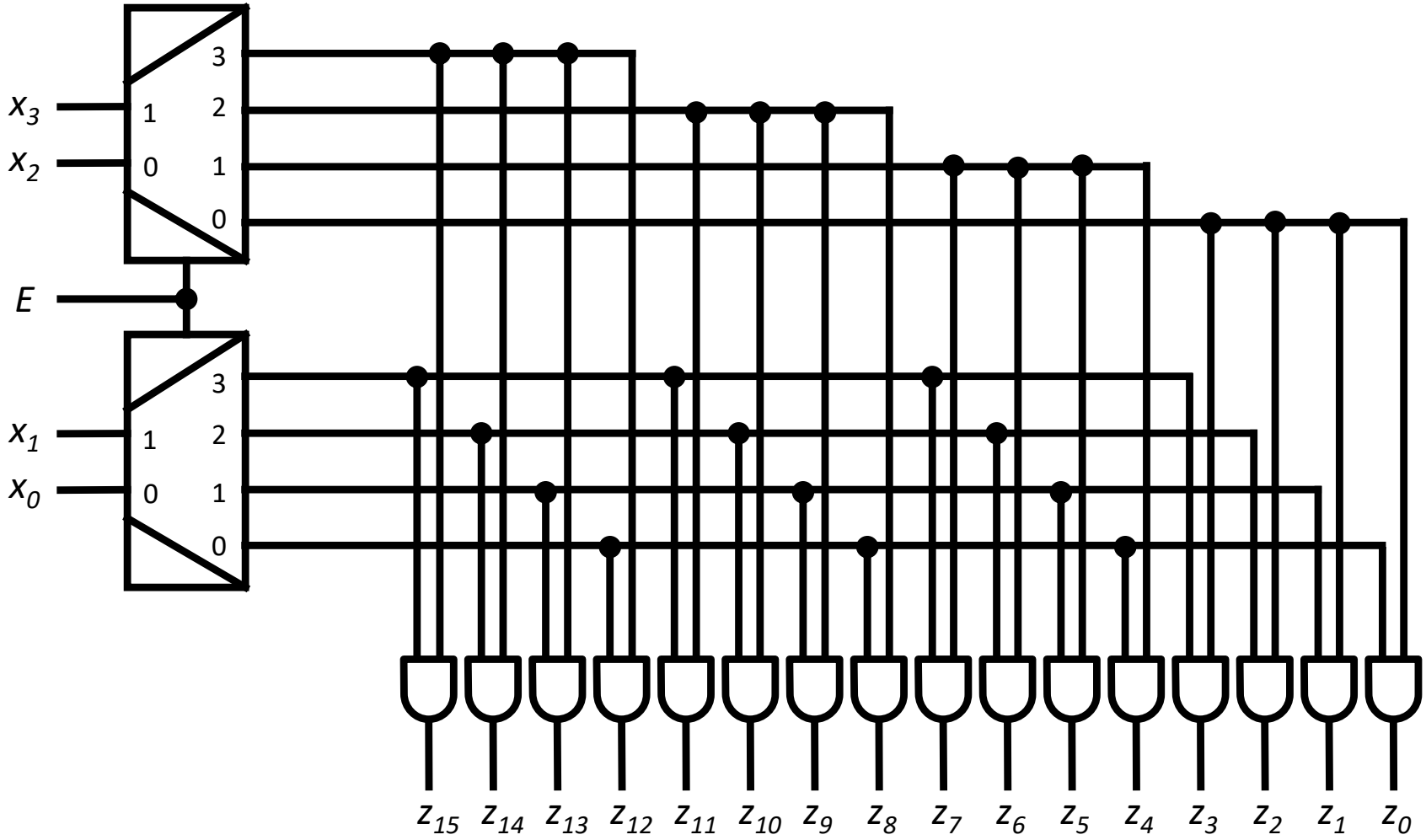
Decodificador 4 a 16



Decodificador

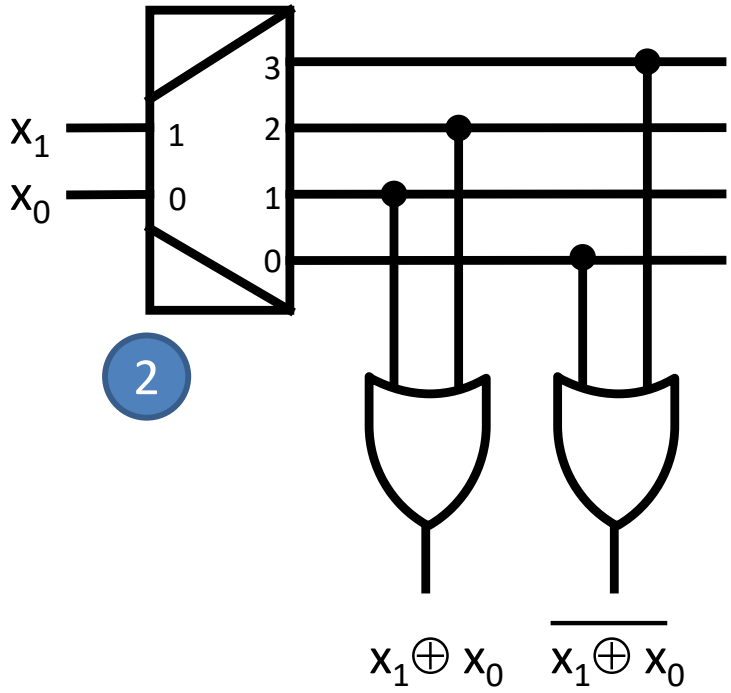
Implementación con predecodificación

Decodificador 4 a 16

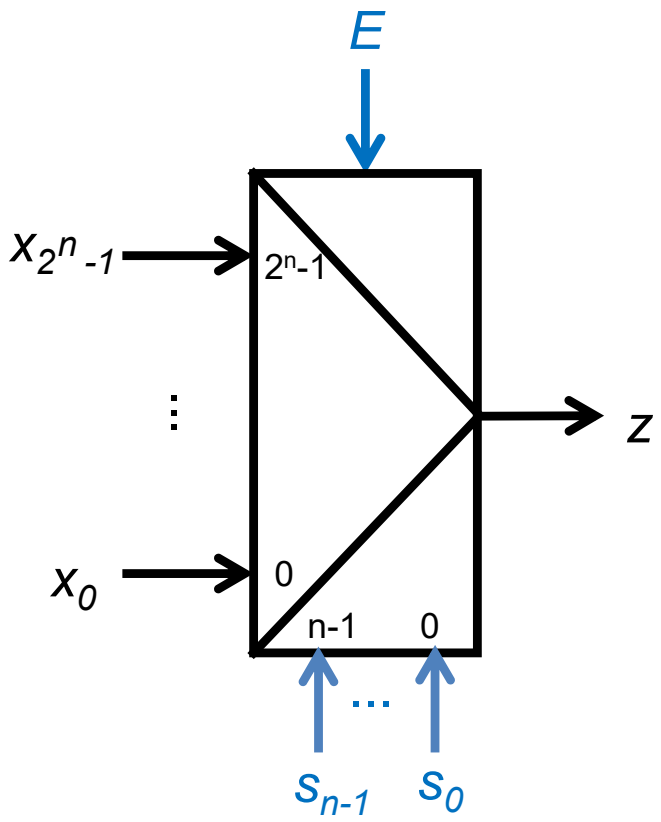


Decodificador

- Aplicaciones al diseño:
  1. Habilitar selectivamente 1 de n subcomponentes cada uno asociado a un índice (dirección) binaria.
  2. Implementar directamente SPC usando puertas OR adicionales (que sumen cada unos de los mintérminos de la FC).



Multiplexor



Multiplexor 2^n a 1

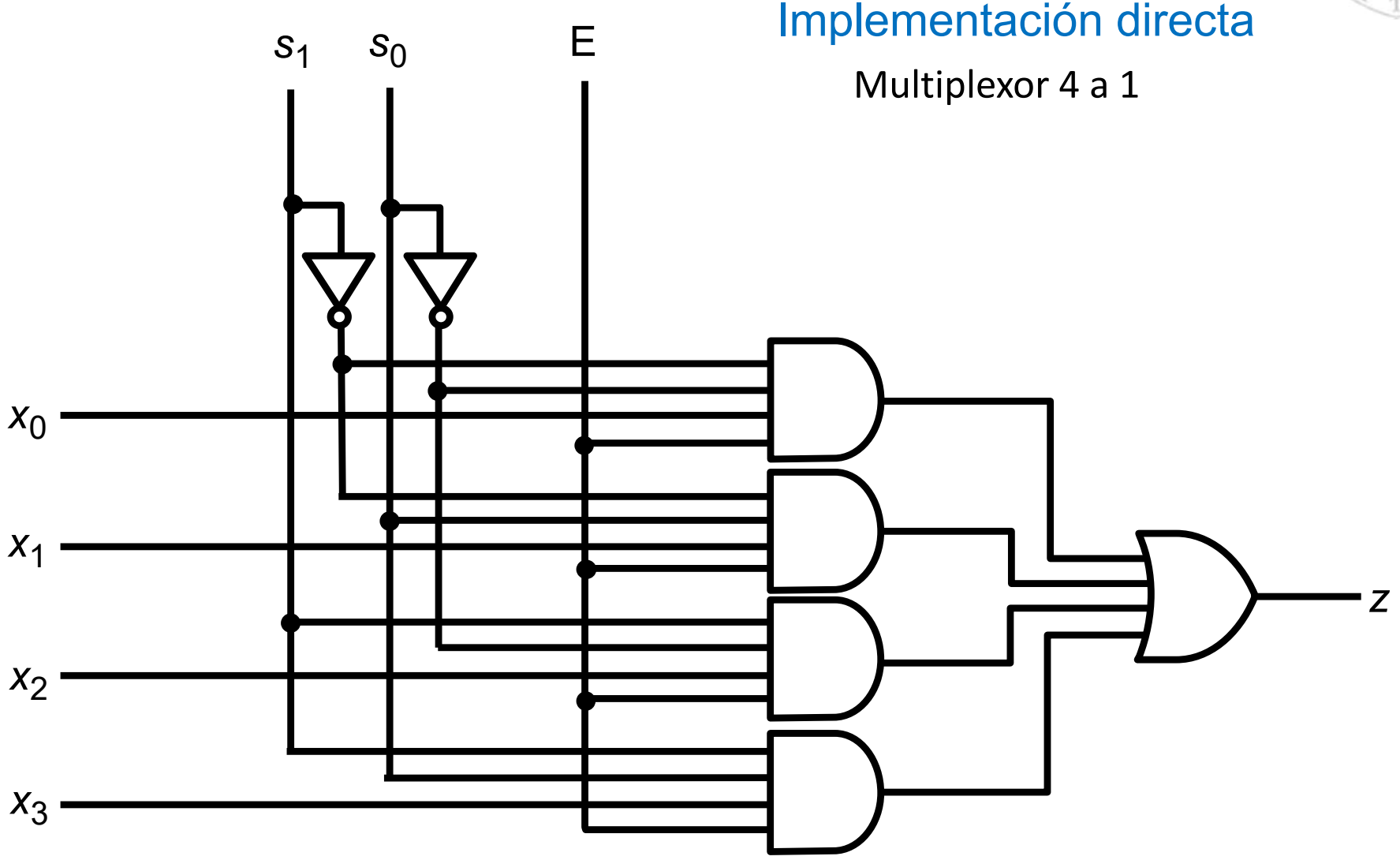
x	2^n entradas de datos
s	n entradas de control
E	1 entrada de capacitación (op)
z	1 salida de datos

si la entrada de control toma la configuración binaria p, la salida equivale a la entrada $(p)_{10}$-ésima

$$z = \begin{cases} x_i & \text{si } E=1 \text{ y } (\underline{s})_{10} = i \\ 0 & \text{en otro caso} \end{cases}$$

$$z = E \cdot \sum(x_i \cdot m_i(\underline{s}))$$

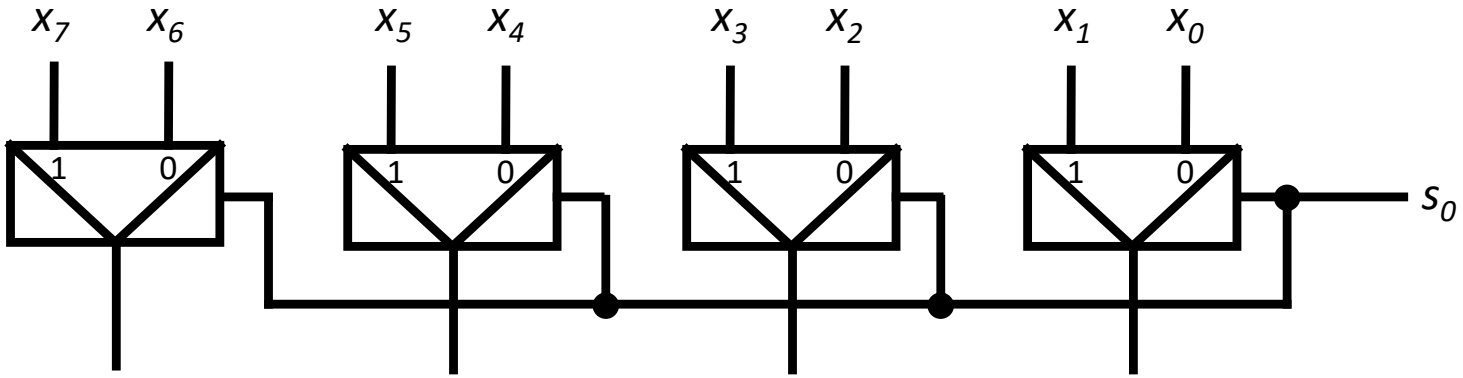
Multiplexor



Multiplexor

Implementación en árbol

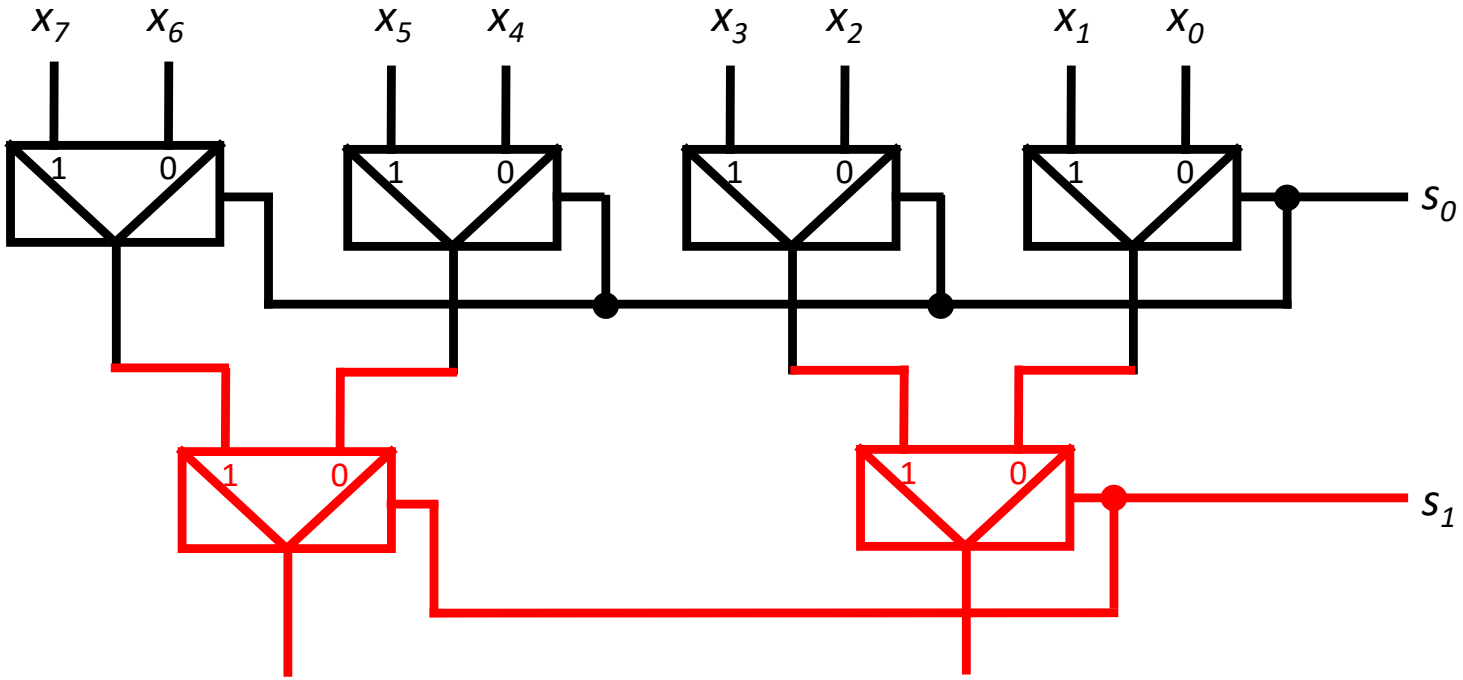
Multiplexor 8 a 1



Multiplexor

Implementación en árbol

Multiplexor 8 a 1

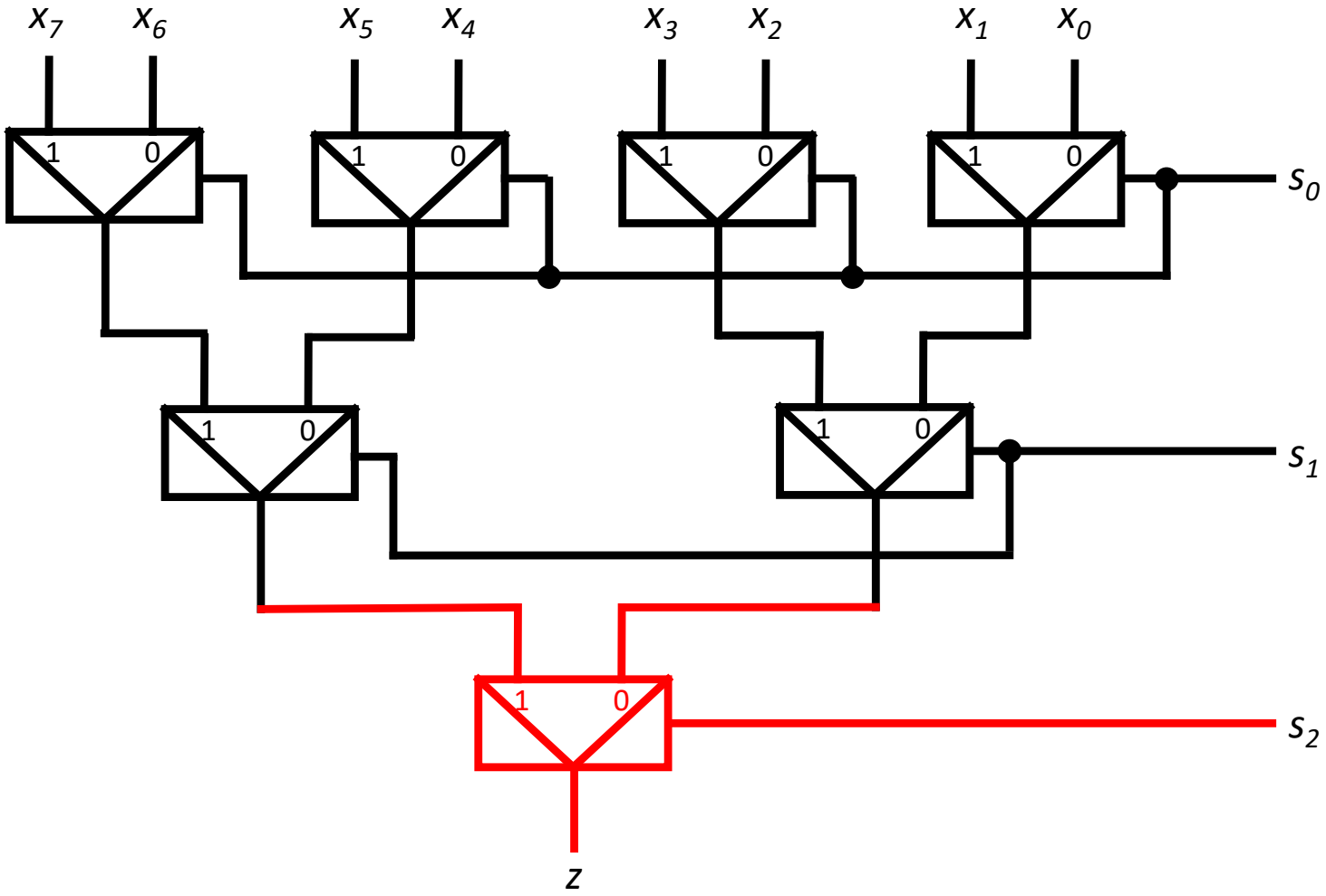


z

Multiplexor

Implementación en árbol

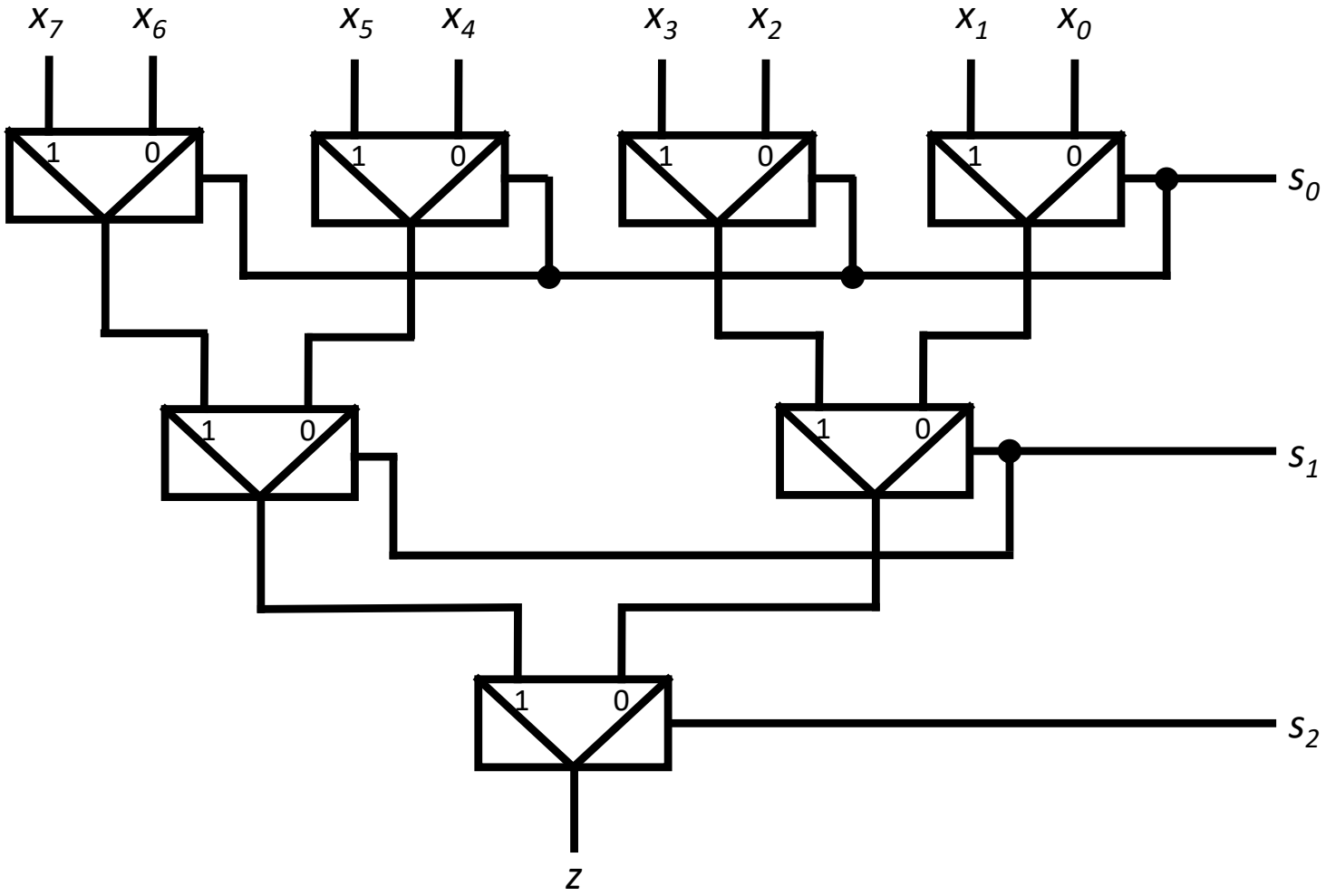
Multiplexor 8 a 1



Multiplexor

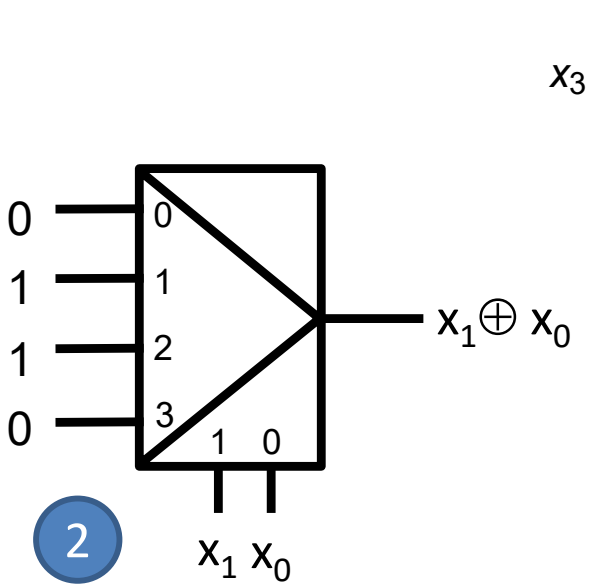
Implementación en árbol

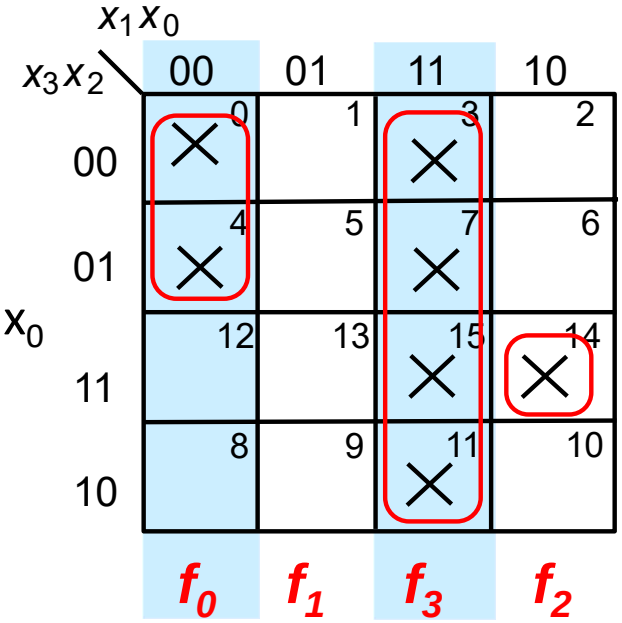
Multiplexor 8 a 1

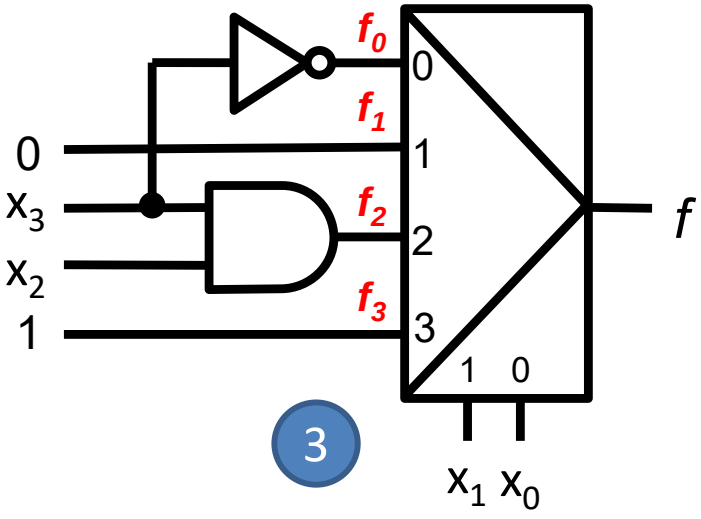


Multiplexor

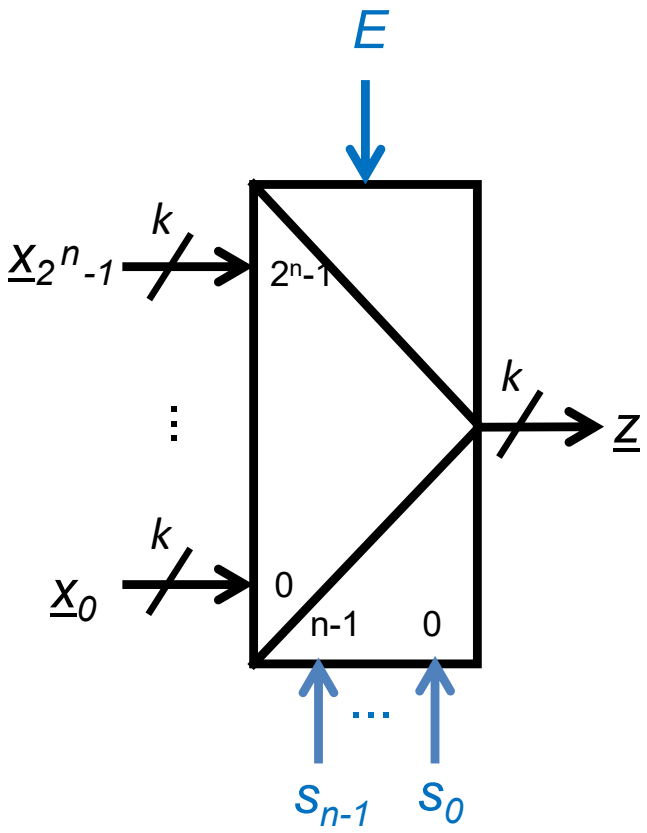
- Aplicaciones al diseño:
  1. Conectar selectivamente varias entradas una misma salida.
  2. Implementar directamente FC que tengan el mismo número de variables que entradas de control (transcribiendo su tabla de verdad).
  3. Implementar funciones de manera que las EC a simplificar tengan menos variables.







Multiplexor vectorial



Multiplexor 2^n a 1 de k bits

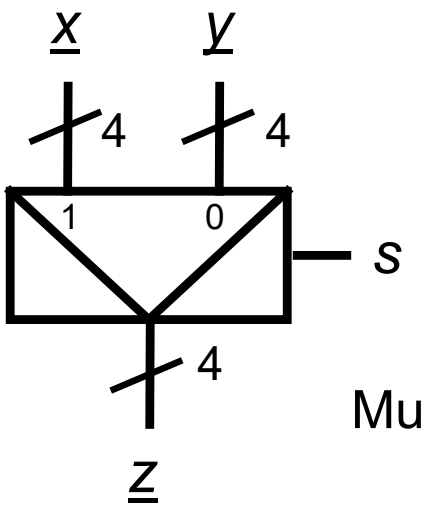
$\underline{x}$	2^n entradas de datos de k bits
$\underline{s}$	n entradas de control
E	1 entrada de capacitación (op)
z	1 salida de datos de k bits

si la entrada de control toma la configuración binaria p, la salida equivale a la entrada $(p)_{10}$-ésima

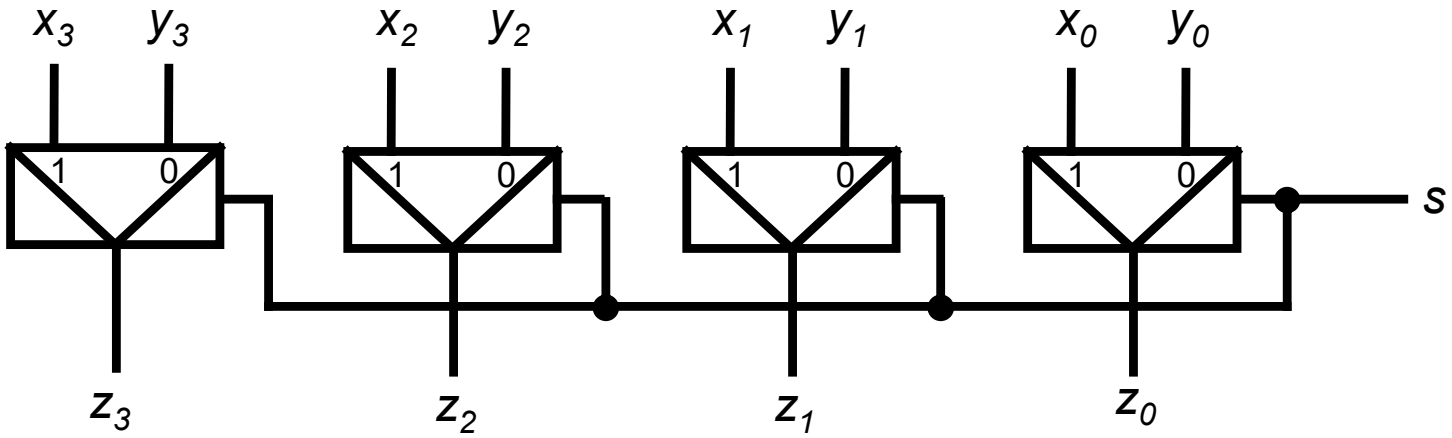
$$\underline{z} = \begin{cases} \underline{x}_i & \text{si } E=1 \text{ y } (\underline{s})_{10} = i \\ 0 & \text{en otro caso} \end{cases}$$

$$z_j = E \cdot \sum(x_{ij} \cdot m_i(\underline{s}))$$

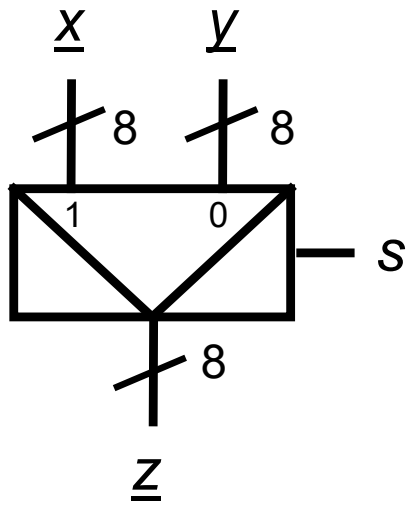
Multiplexor vectorial

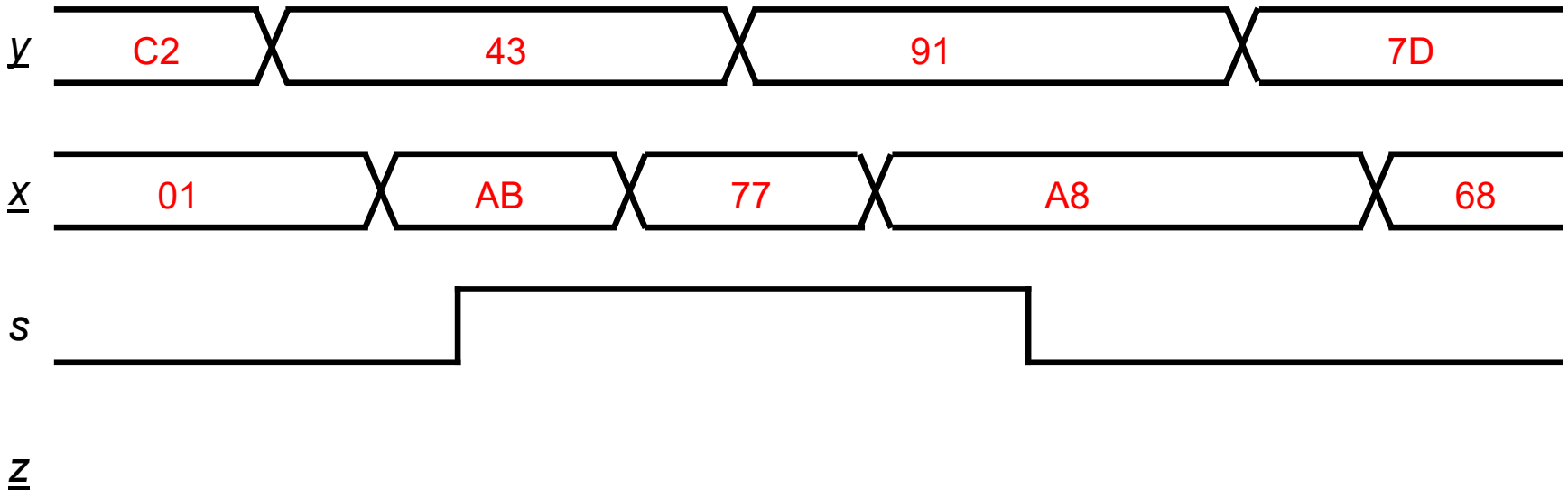


Multiplexor 2 a 1 de 4 bits

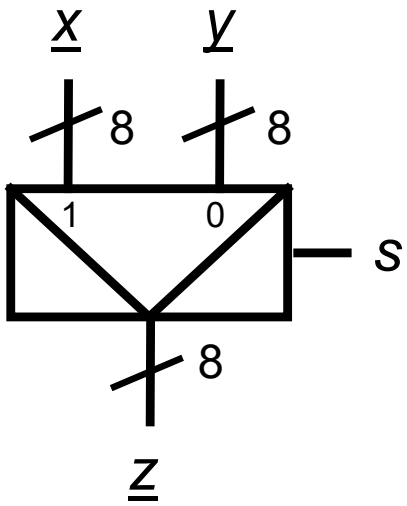


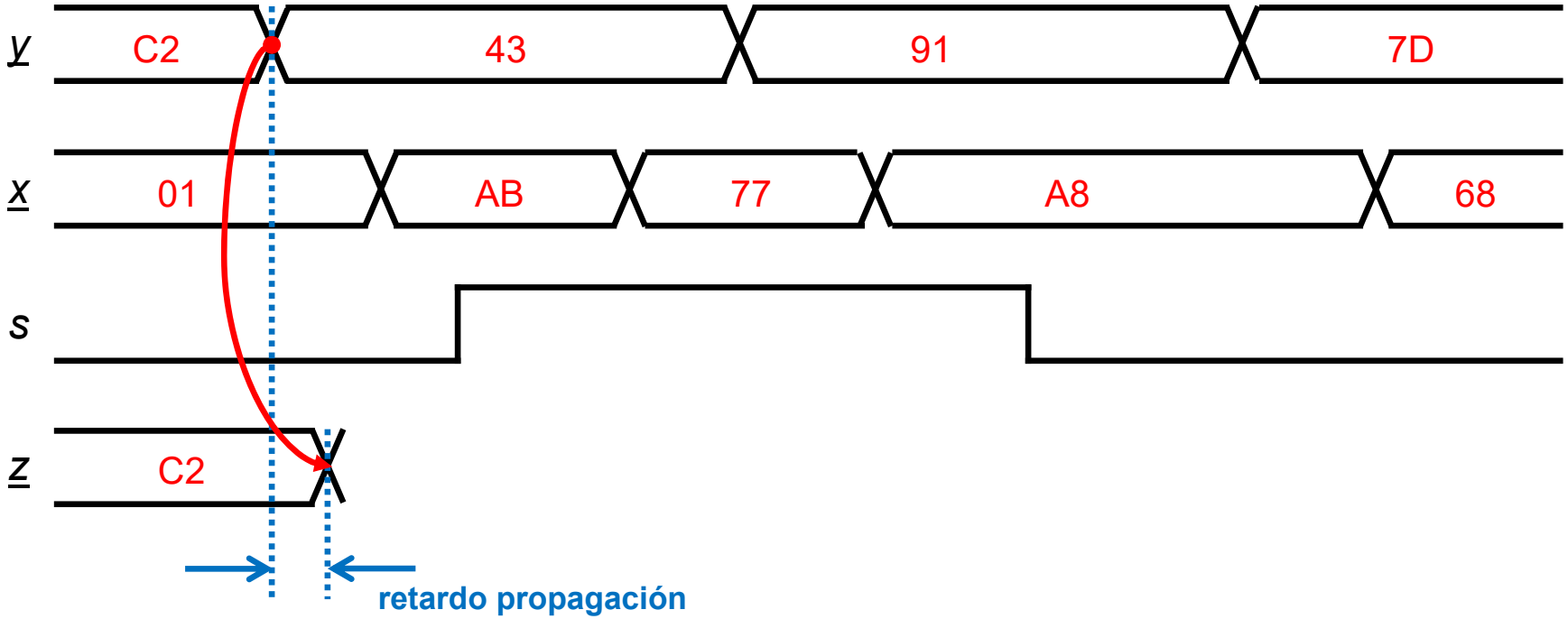
Multiplexor vectorial



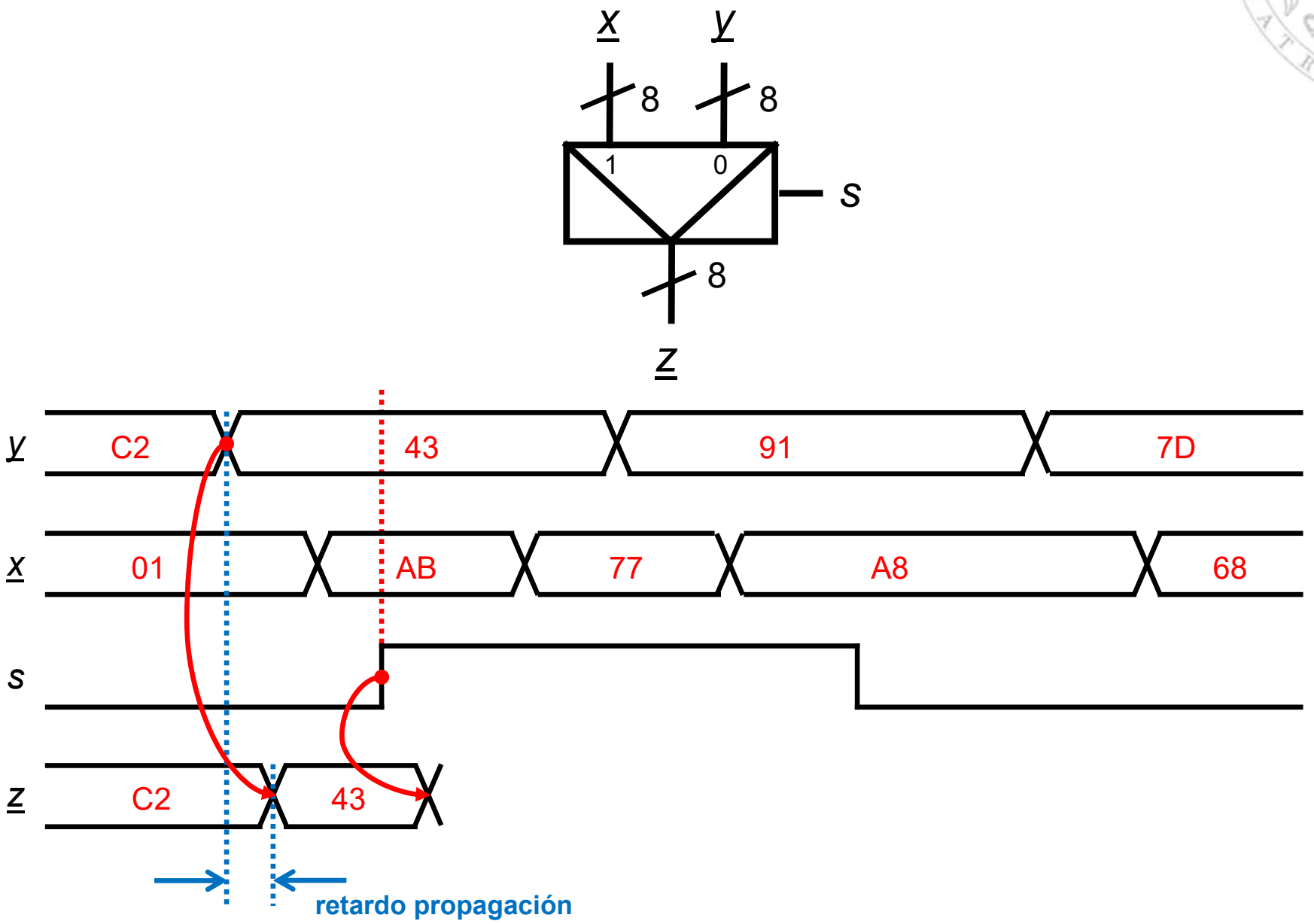


Multiplexor vectorial

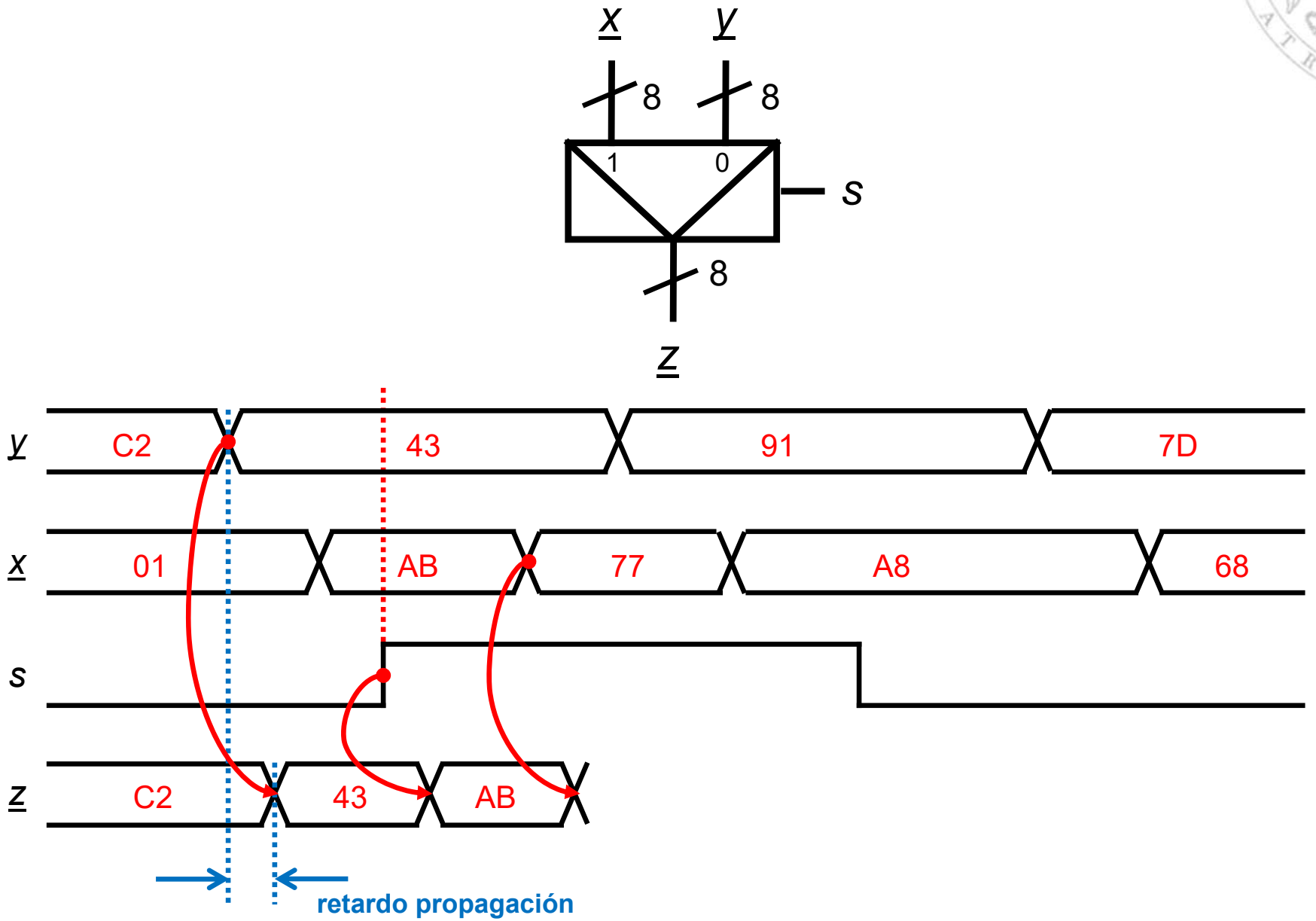




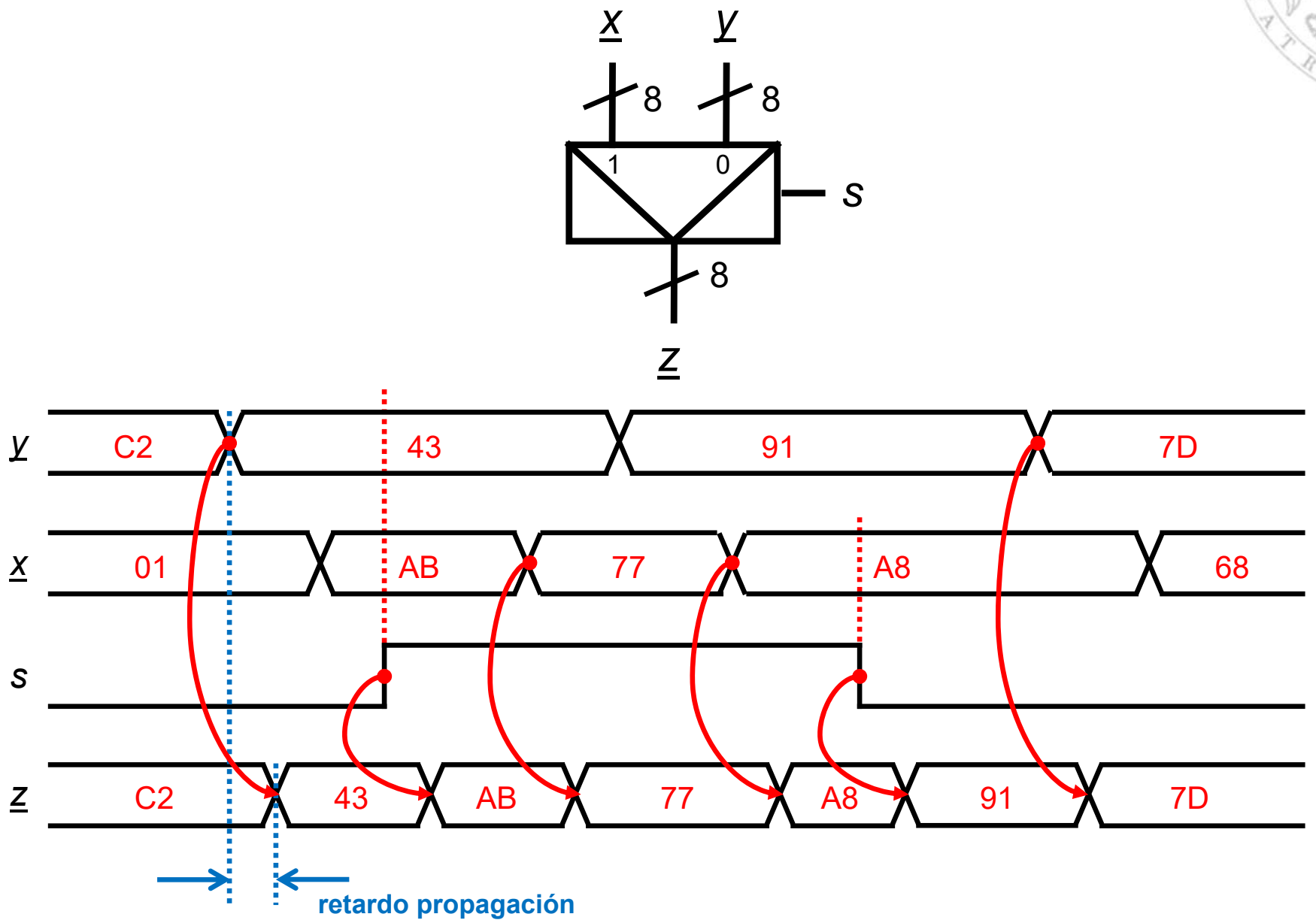
Multiplexor vectorial



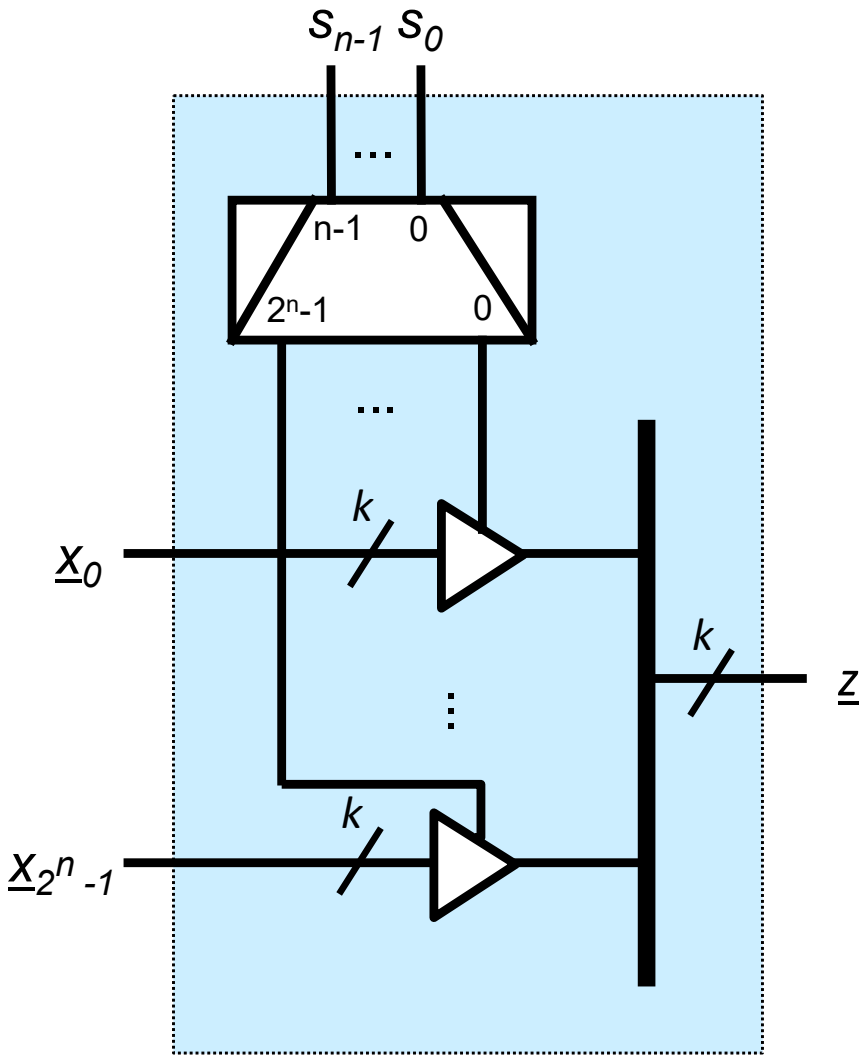
Multiplexor vectorial



Multiplexor vectorial



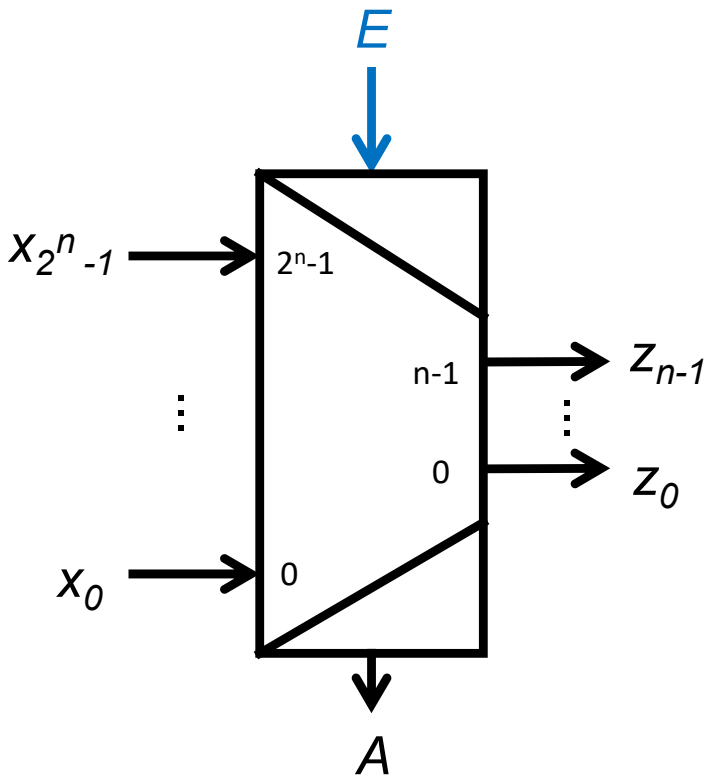
Bus



$\underline{x}$	2^n entradas de datos de k bits
$\underline{s}$	n entradas de control
z	1 salida de datos de k bits

si la entrada de control toma la configuración binaria p, la salida equivale a la entrada $(p)_{10}$-ésima

Codificador



Codificador 2^n a n

x	2^n entradas de datos
z	n salidas de datos
E	1 entrada de capacitación (op)
A	1 salida de actividad

si se activa la entrada p-ésima y solo esa, la salida codifica p en binario

$$z = \begin{cases} (i)_2 & \text{si } E=1 \text{ y } x_i = 1 \text{ y } \forall j, j\neq i,\ x_j=0 \\ 0 & \text{si } E=0 \text{ ó si } E=1 \text{ y } \forall i\ x_i=0 \\ - & \text{en otro caso} \end{cases}$$

$$A = \begin{cases} 1 & \text{si } E=1 \text{ y } \exists i,\ x_i=1 \\ 0 & \text{en otro caso} \end{cases}$$

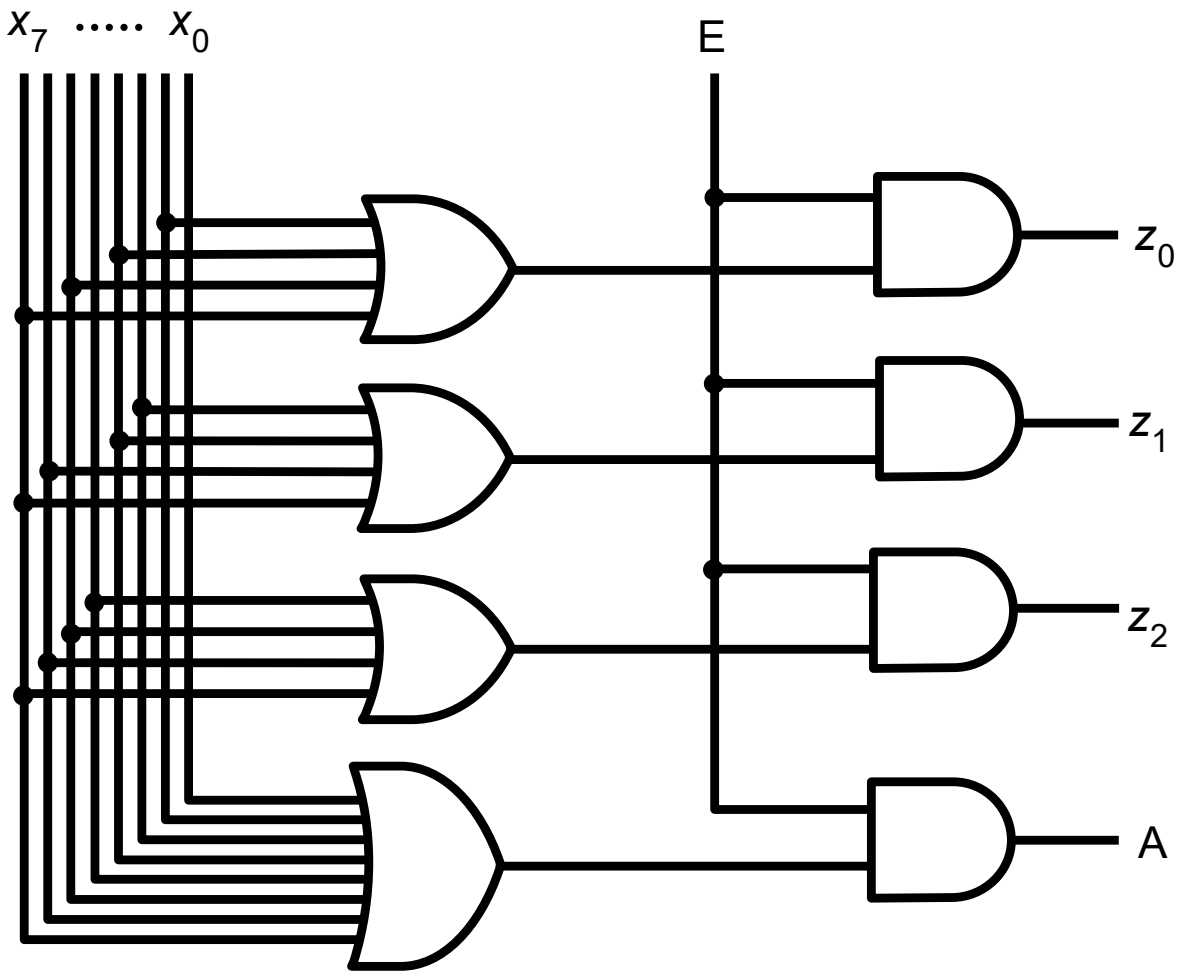
$$z_i = E \cdot \sum(x_j) \text{ con } j \in \{ (a_{n-1}...a_0)_2 / a_i = 1 \}$$
$$A = E \cdot \sum(x_i)$$

Codificador

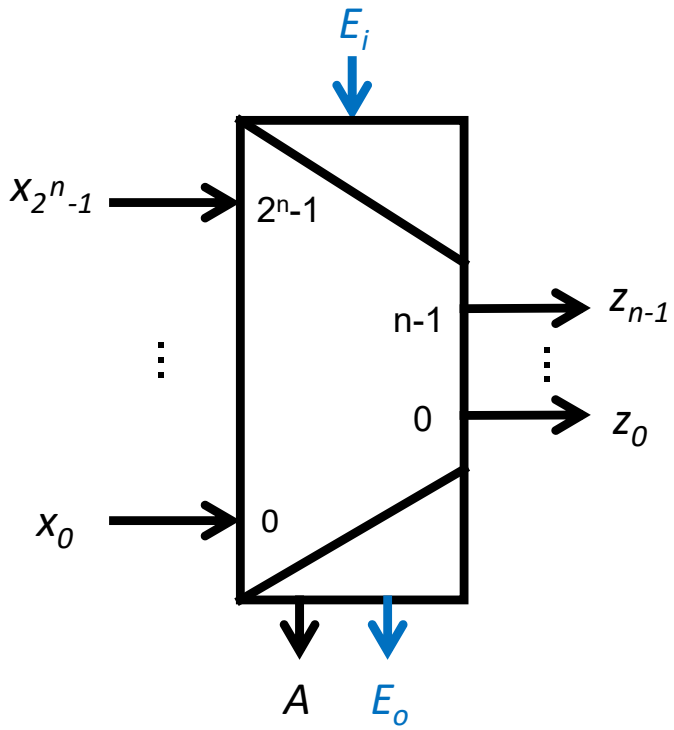
Implementación directa

Codificador 8 a 3

entrada activada	z_2	z_1	z_0
x_0	0	0	0
x_1	0	0	1
x_2	0	1	0
x_3	0	1	1
x_4	1	0	0
x_5	1	0	1
x_6	1	1	0
x_7	1	1	1



Codificador de prioridad



Codificador de prioridad 2^n a n

x	2^n entradas de datos
z	n salidas de datos
E_i	1 entrada de capacitación (op)
E_o	1 salida de capacitación (op)
A	1 salida de actividad

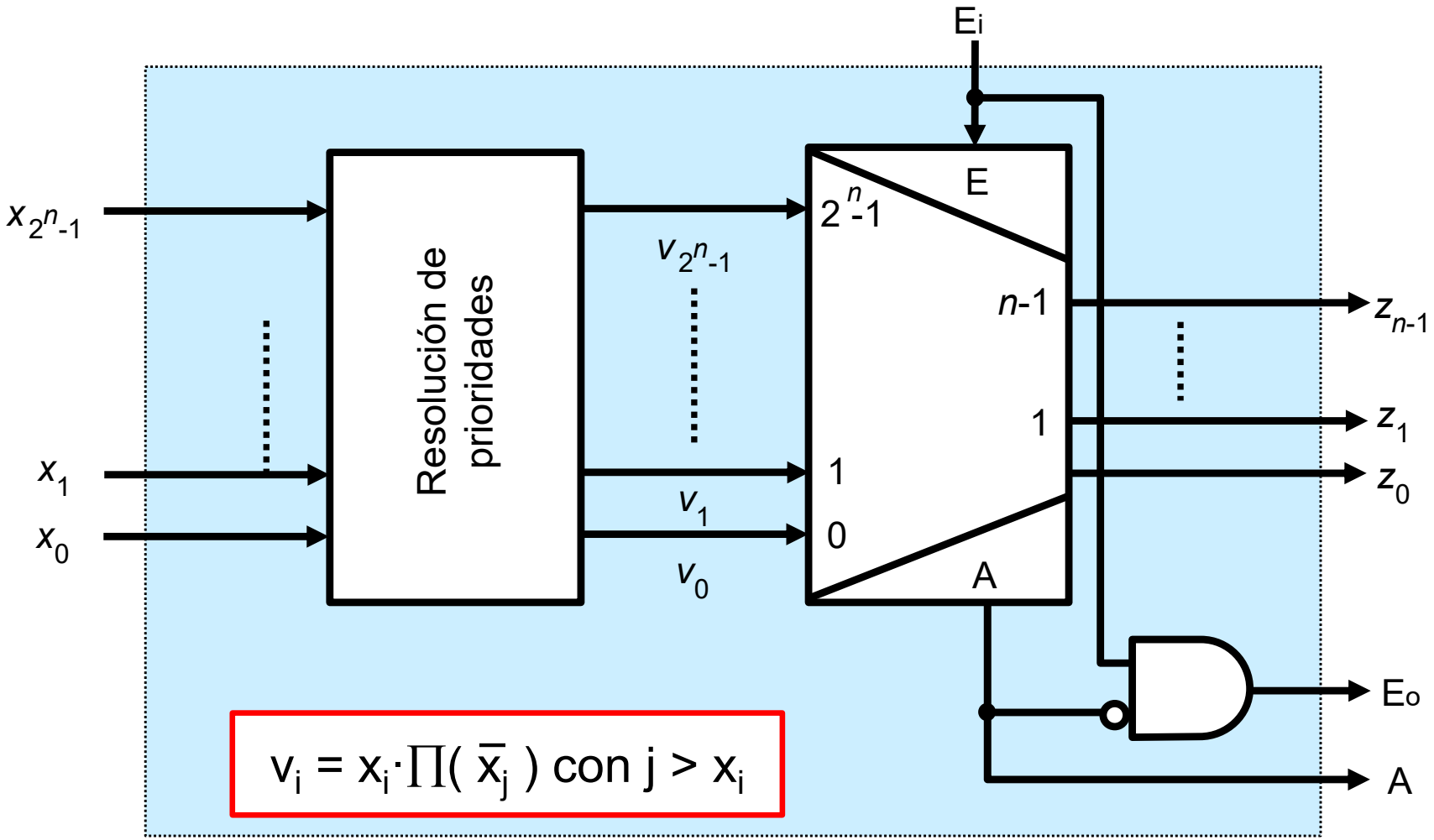
la salida codifica en binario la entrada activa de más peso

$$\underline{z} = \begin{cases} (i)_2 & \text{si } E_i=1 \text{ y } x_i = 1 \text{ y } \forall j,\ j>i,\ x_j=0 \\ 0 & \text{en otro caso} \end{cases}$$

$$A = \begin{cases} 1 & \text{si } E_i=1 \text{ y } \exists i,\ x_i=1 \\ 0 & \text{en otro caso} \end{cases}$$

$$E_o = \begin{cases} 1 & \text{si } E_i=1 \text{ y } \forall j,\ x_j = 0 \\ 0 & \text{en otro caso} \end{cases}$$

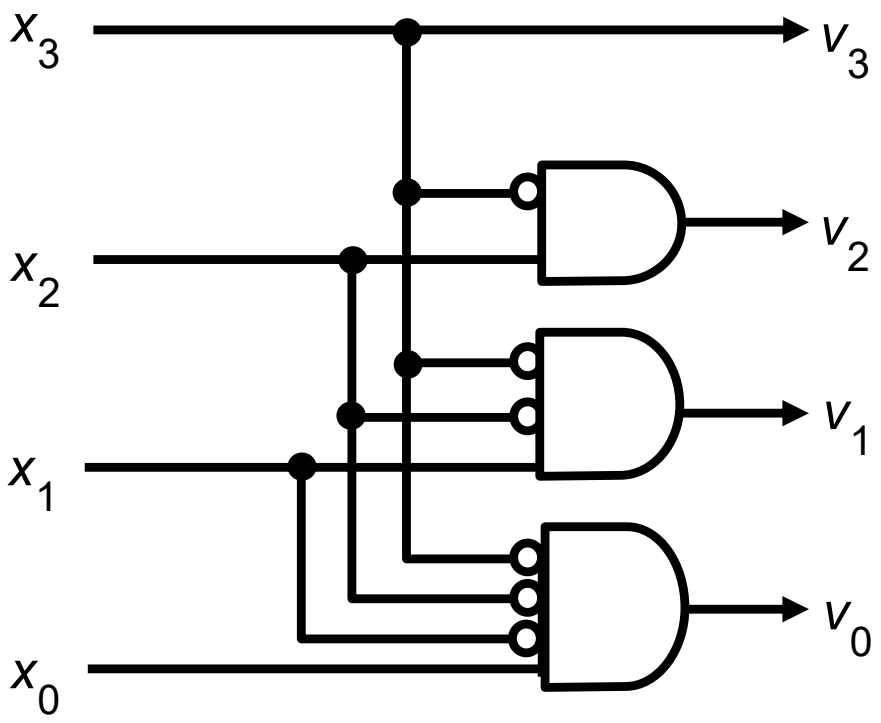
Codificador de prioridad



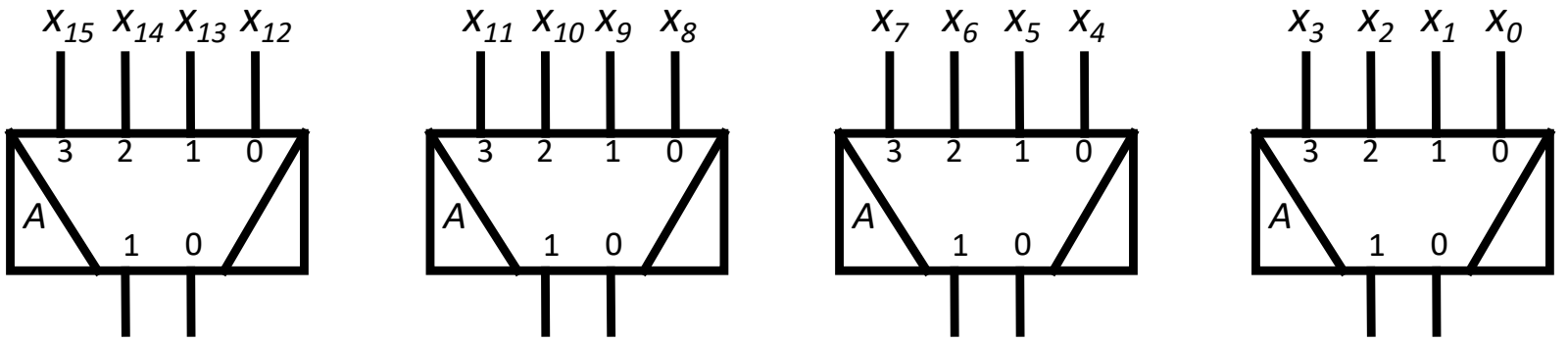
Codificador de prioridad

Implementación directa

Resolución de prioridades
Codificador 4 a 2



Codificador de prioridad



A

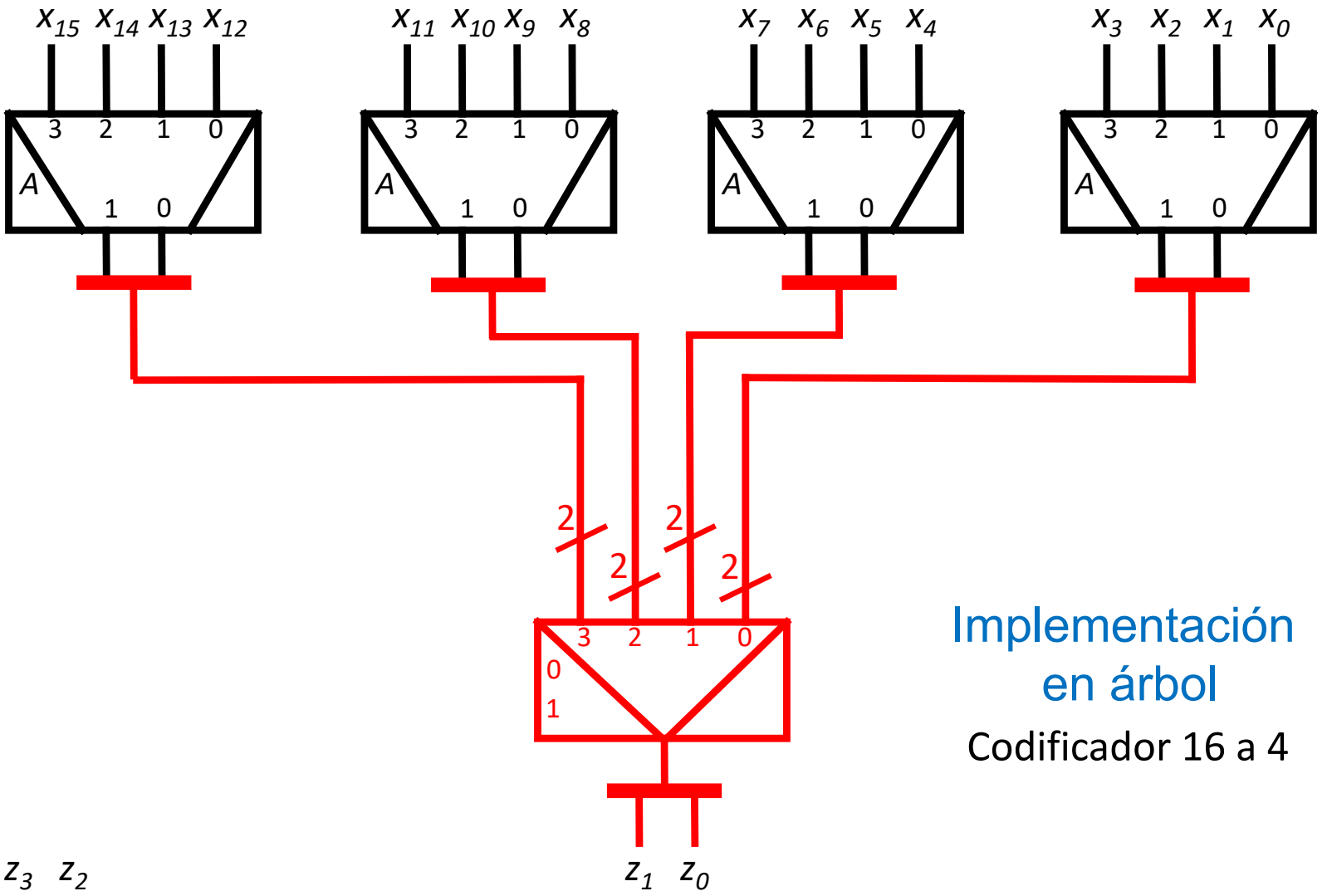
z_3 z_2

z_1 z_0

Implementación en árbol

Codificador 16 a 4

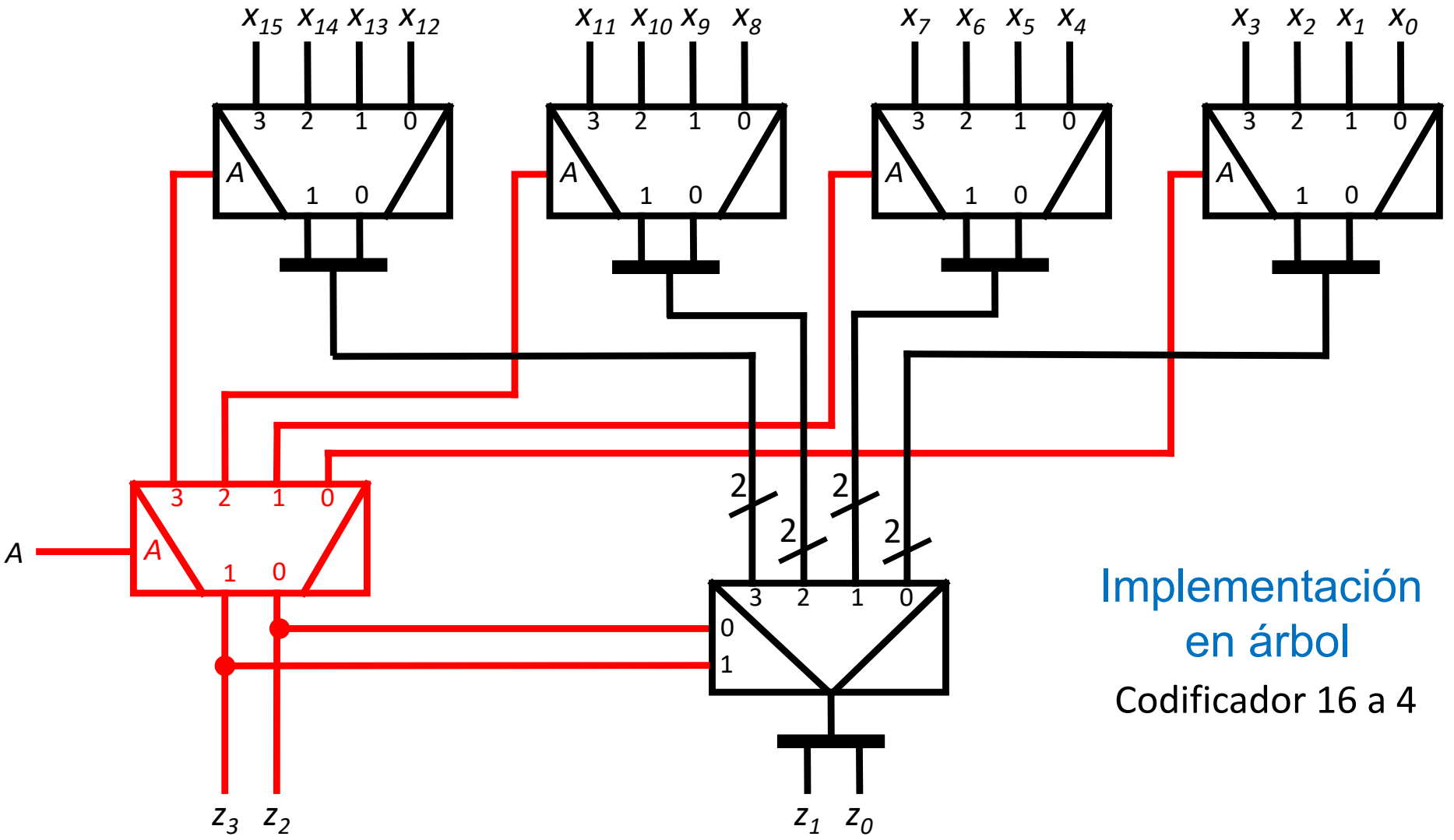
Codificador de prioridad



Implementación en árbol

Codificador 16 a 4

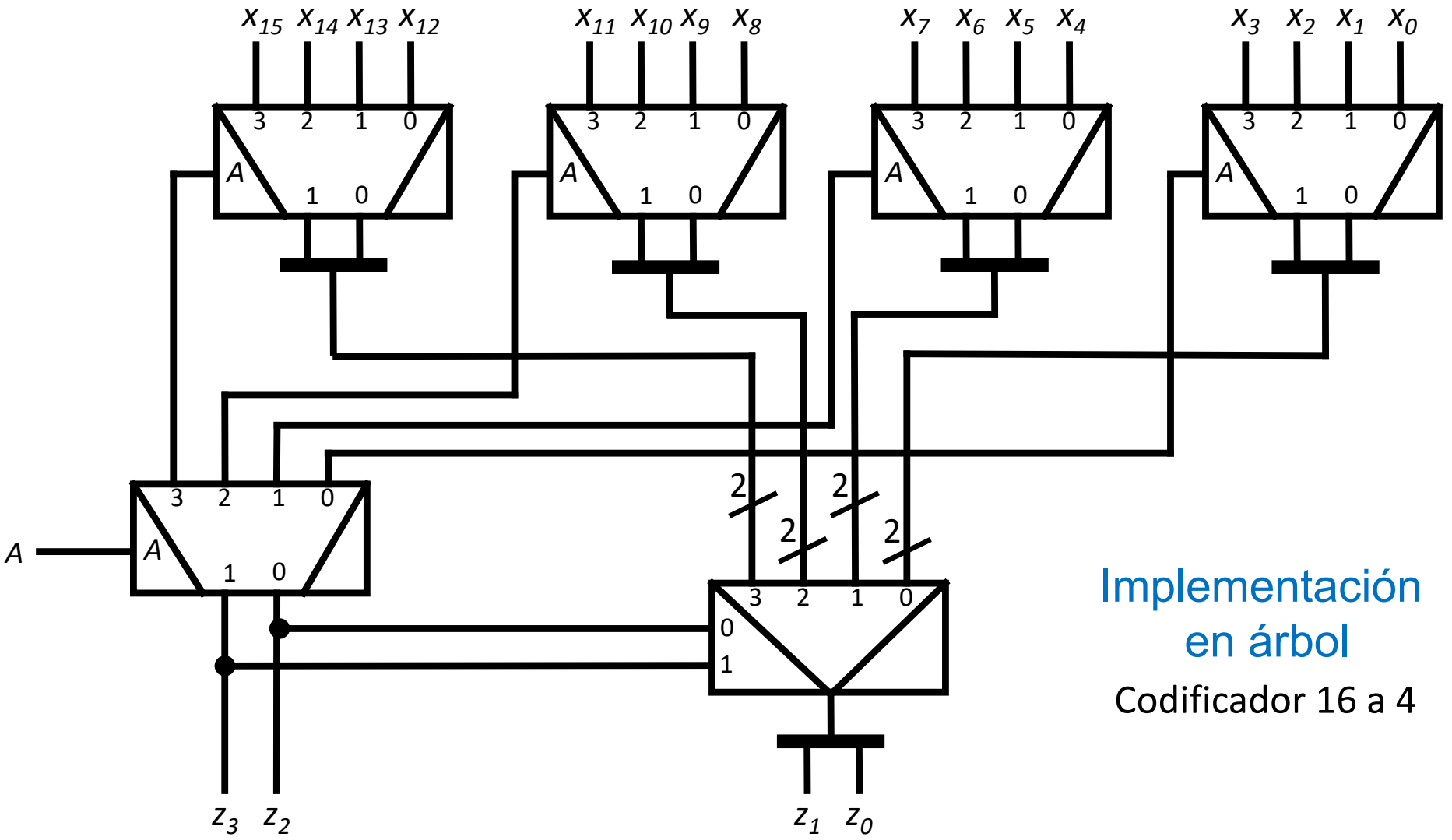
Codificador de prioridad



Implementación en árbol

Codificador 16 a 4

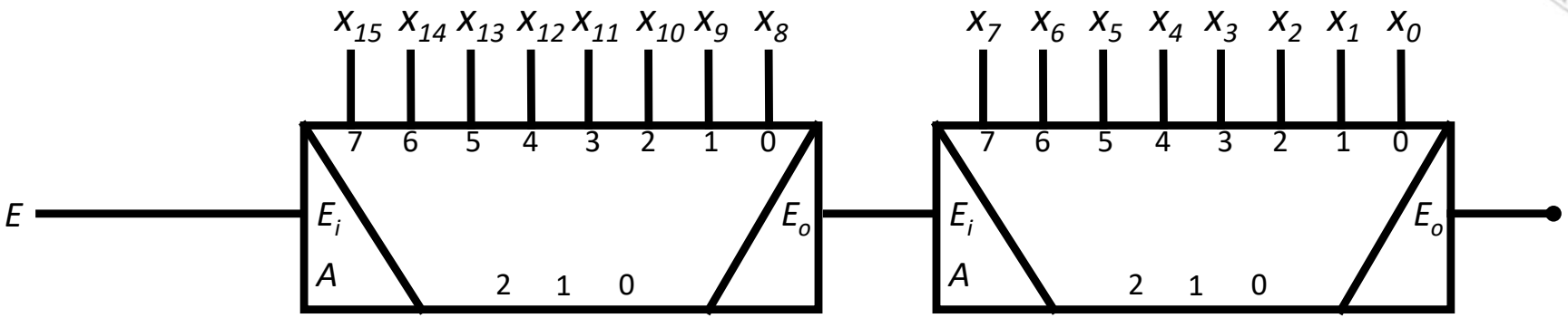
Codificador de prioridad



Implementación en árbol

Codificador 16 a 4

Codificador de prioridad

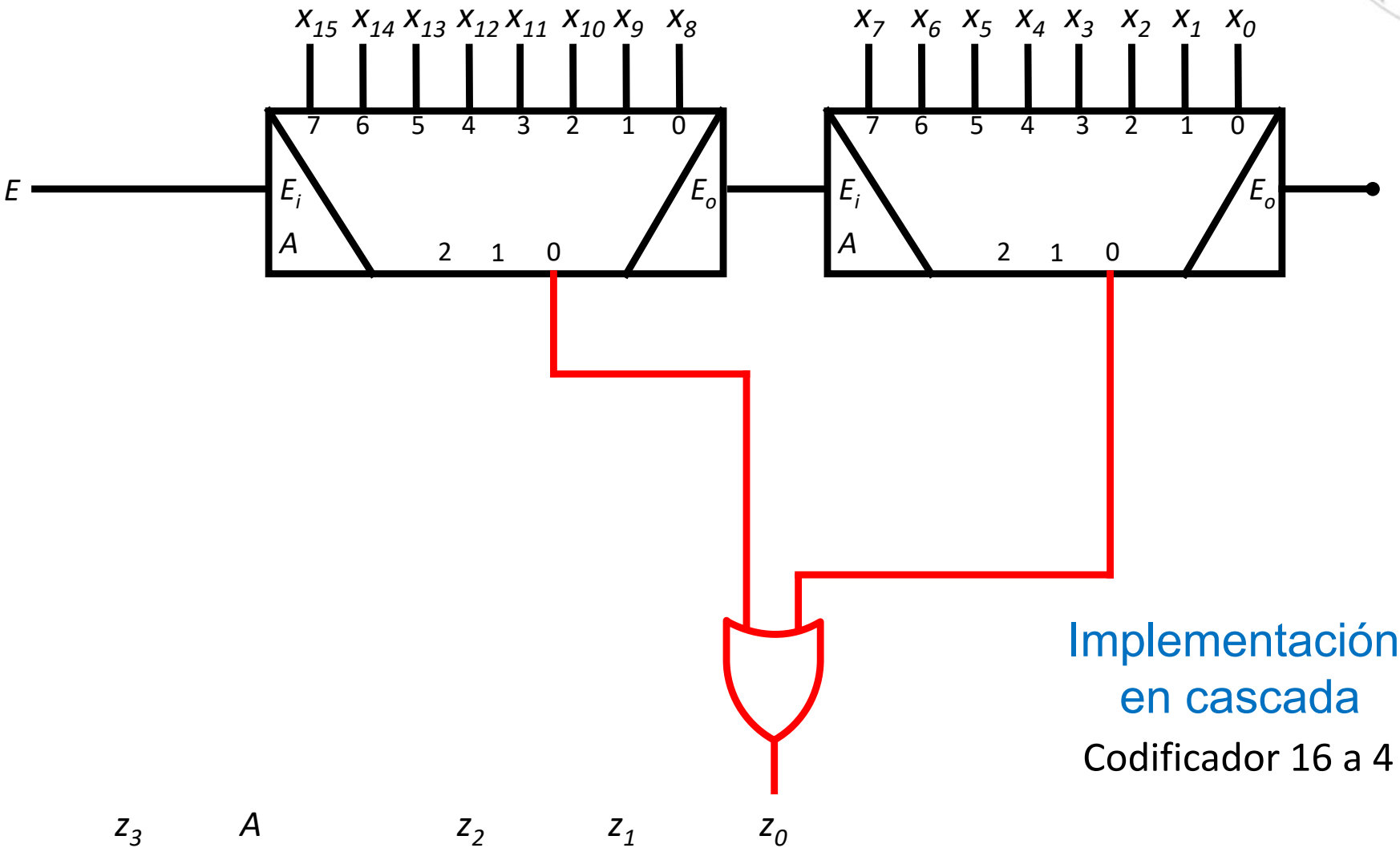


Implementación en cascada

Codificador 16 a 4

z_3 A z_2 z_1 z_0

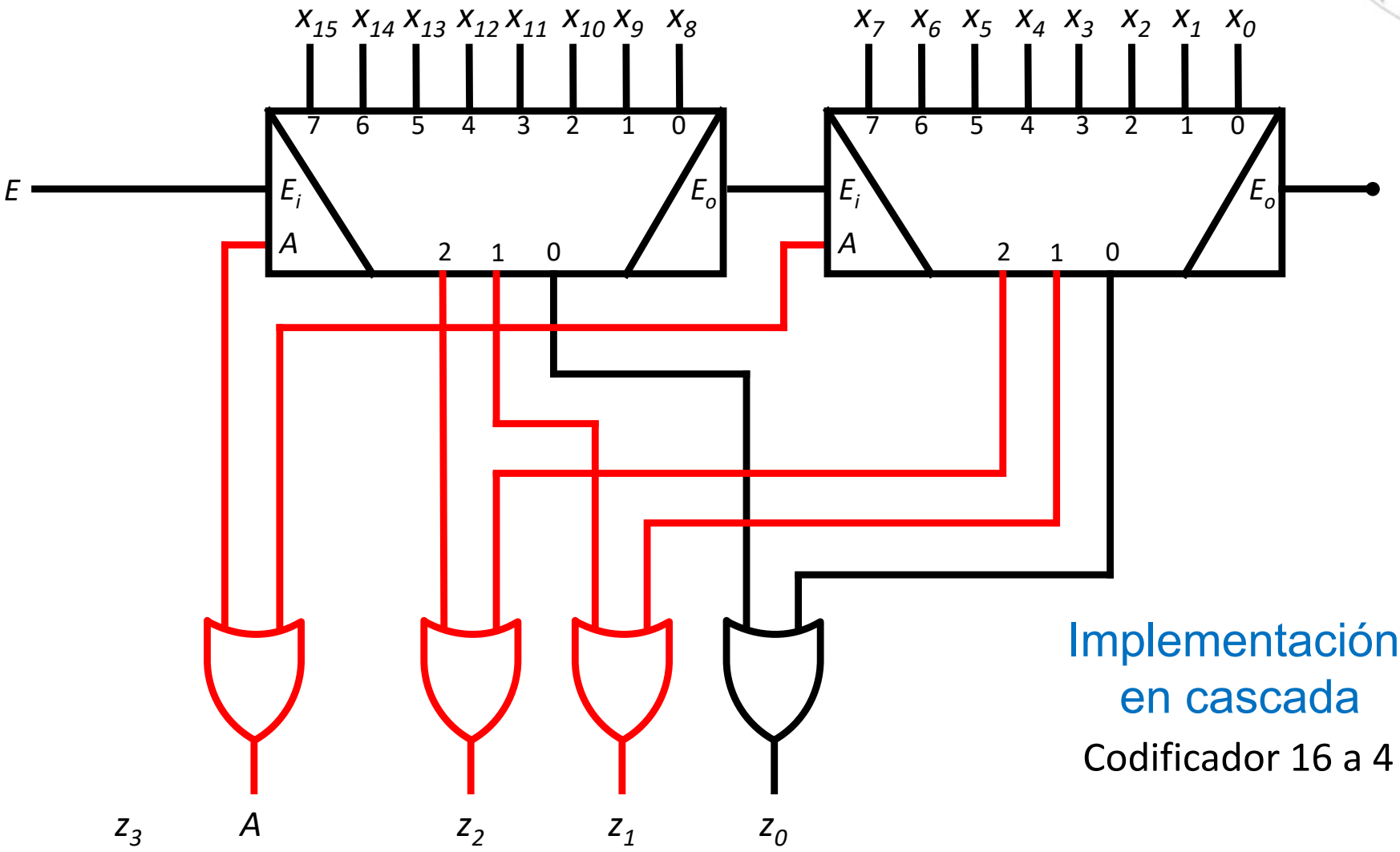
Codificador de prioridad



Implementación en cascada

Codificador 16 a 4

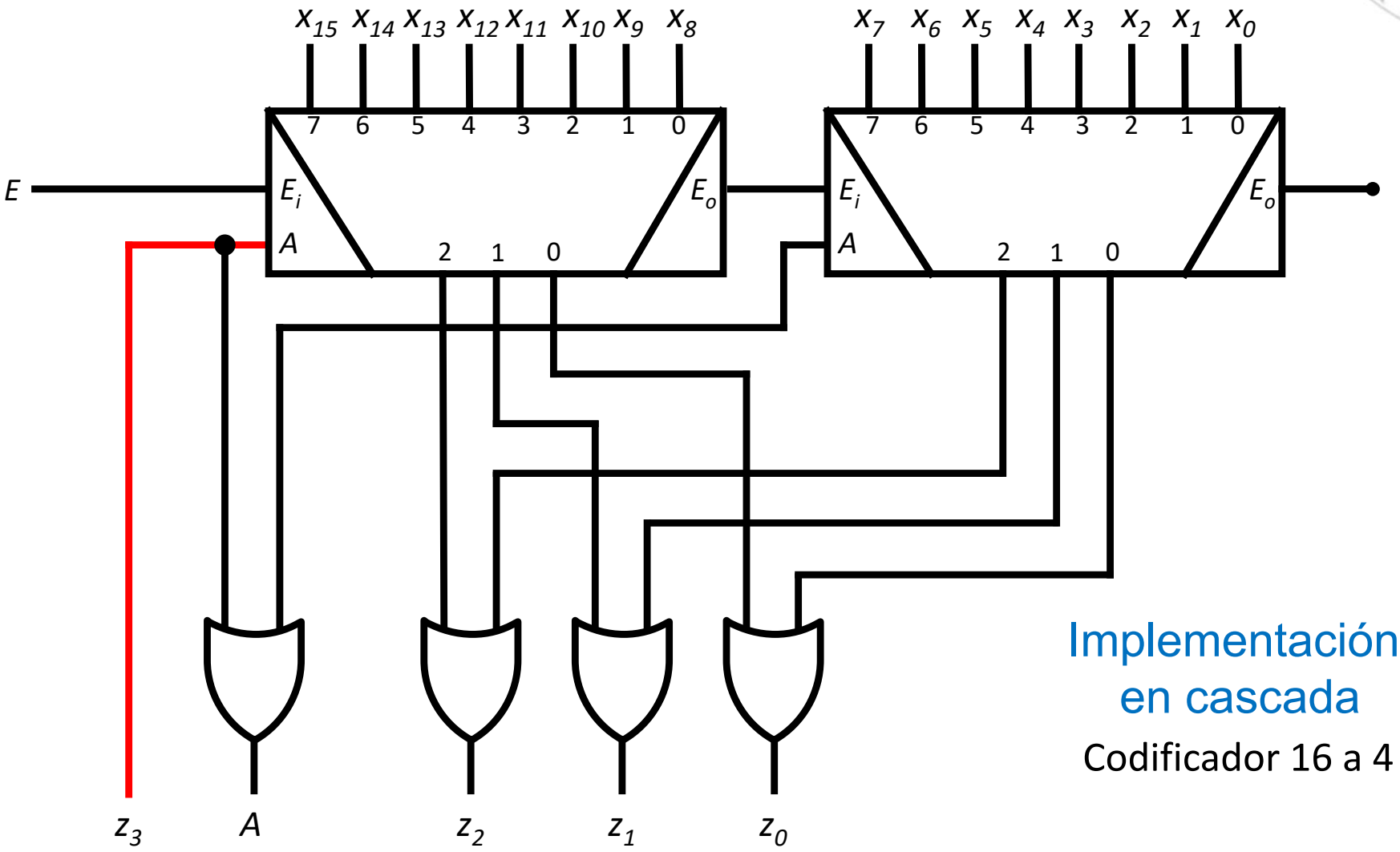
Codificador de prioridad



Implementación en cascada

Codificador 16 a 4

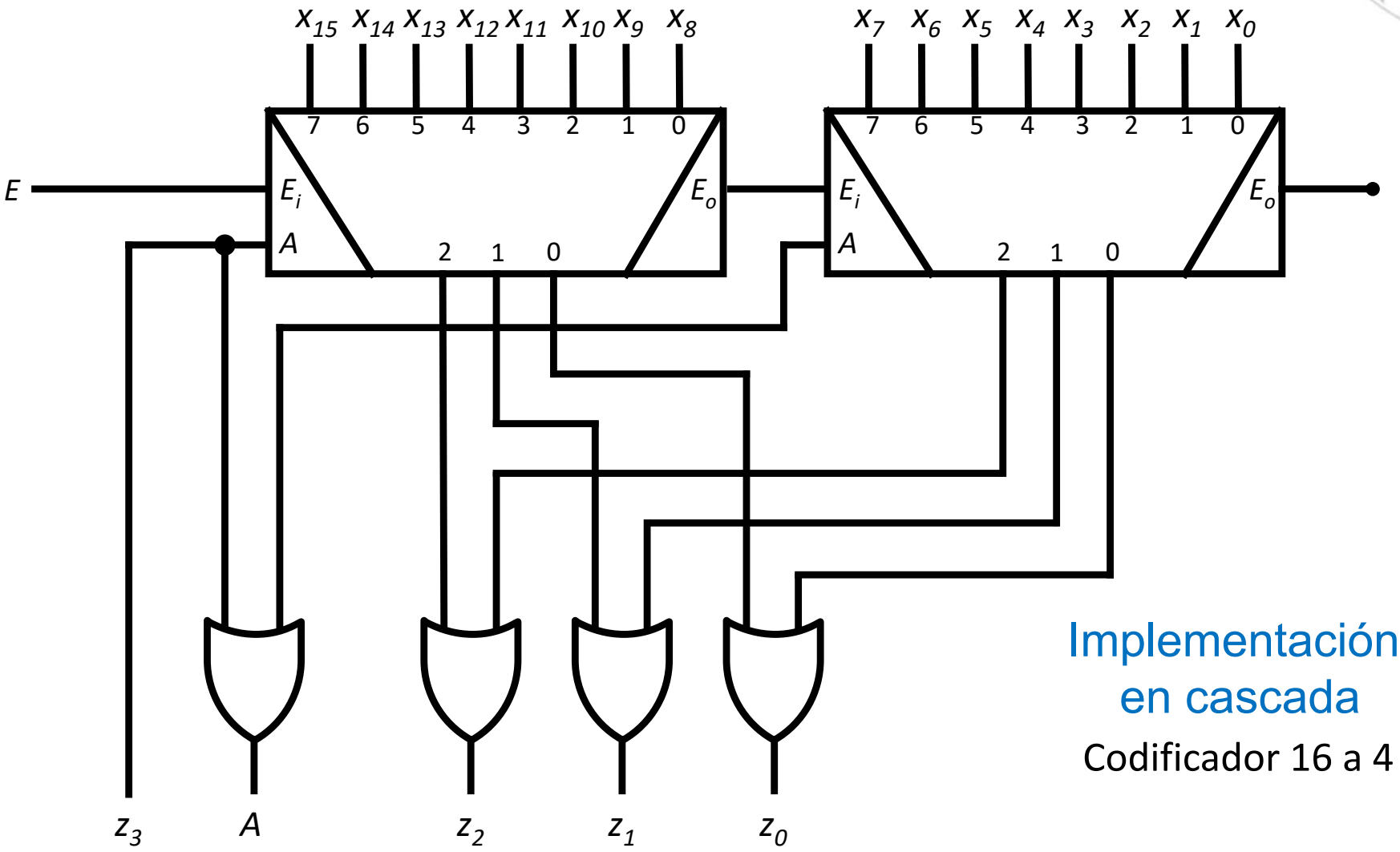
Codificador de prioridad



Implementación en cascada

Codificador 16 a 4

Codificador de prioridad

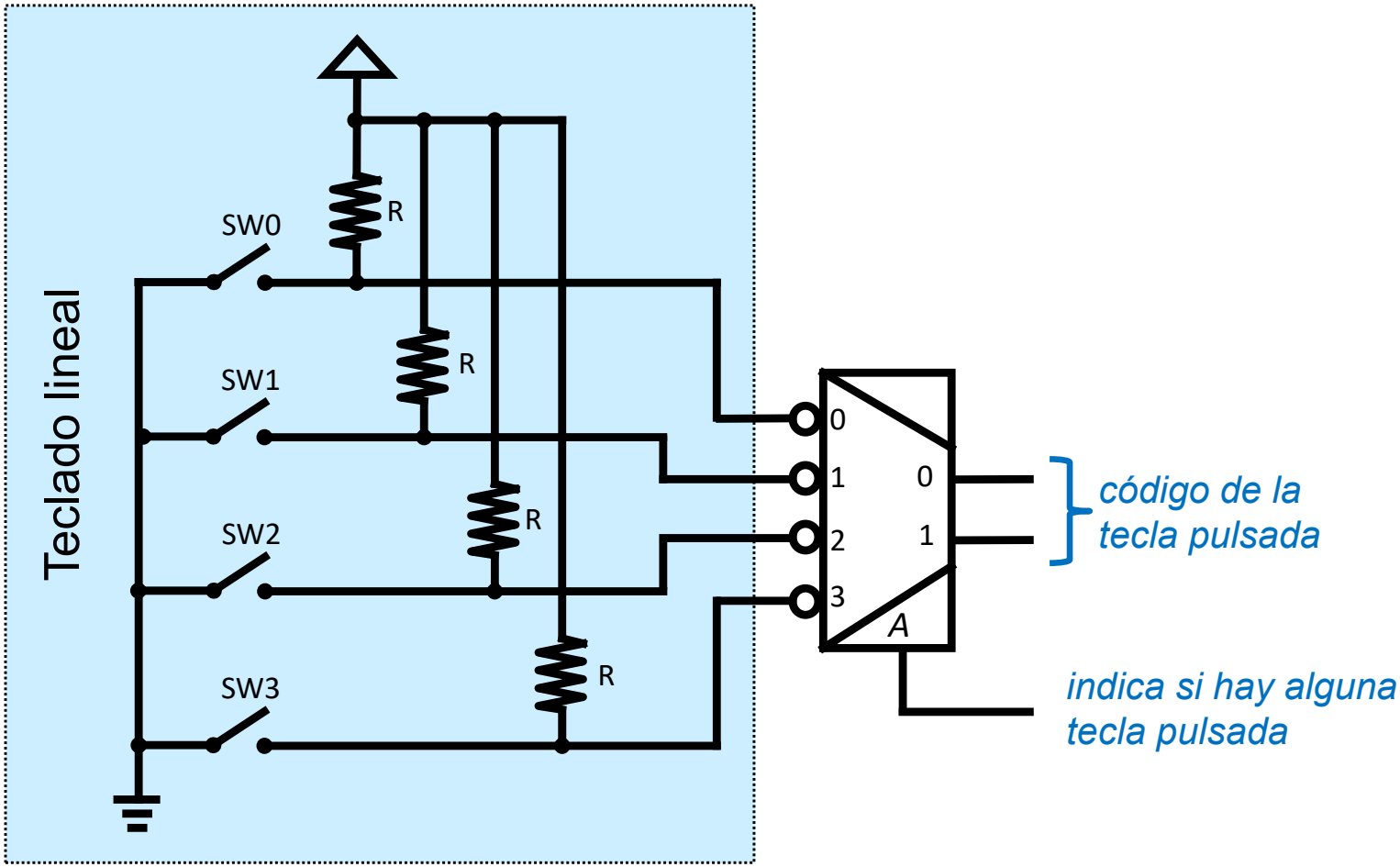


Implementación en cascada

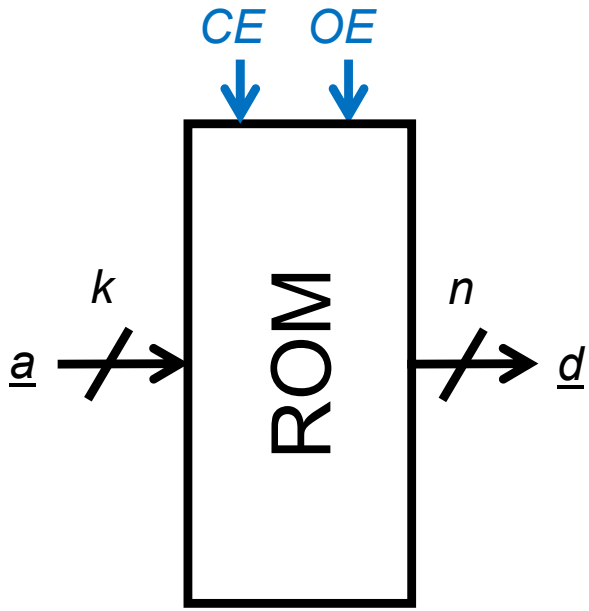
Codificador 16 a 4

Codificador de prioridad

- Aplicaciones al diseño:
  1. Asociar un código a cada componente de un vector de entrada.



ROM (Read Only Memory)



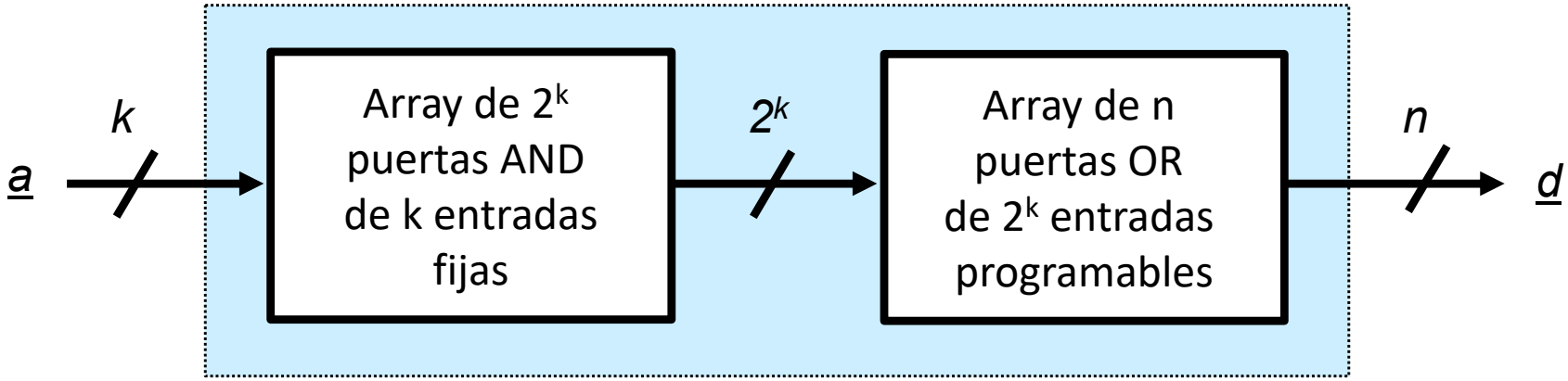
ROM $2^k \times n$
(2^k palabras de n bits)

a	1 entrada de dirección de k bits
d	1 salida de datos de n bits
CE	1 entrada de capacitación (op)
OE	1 entrada de capacitación de lectura (op)

dispositivo programable capaz de implementar n FC de k variables almacenando sus tablas de verdad

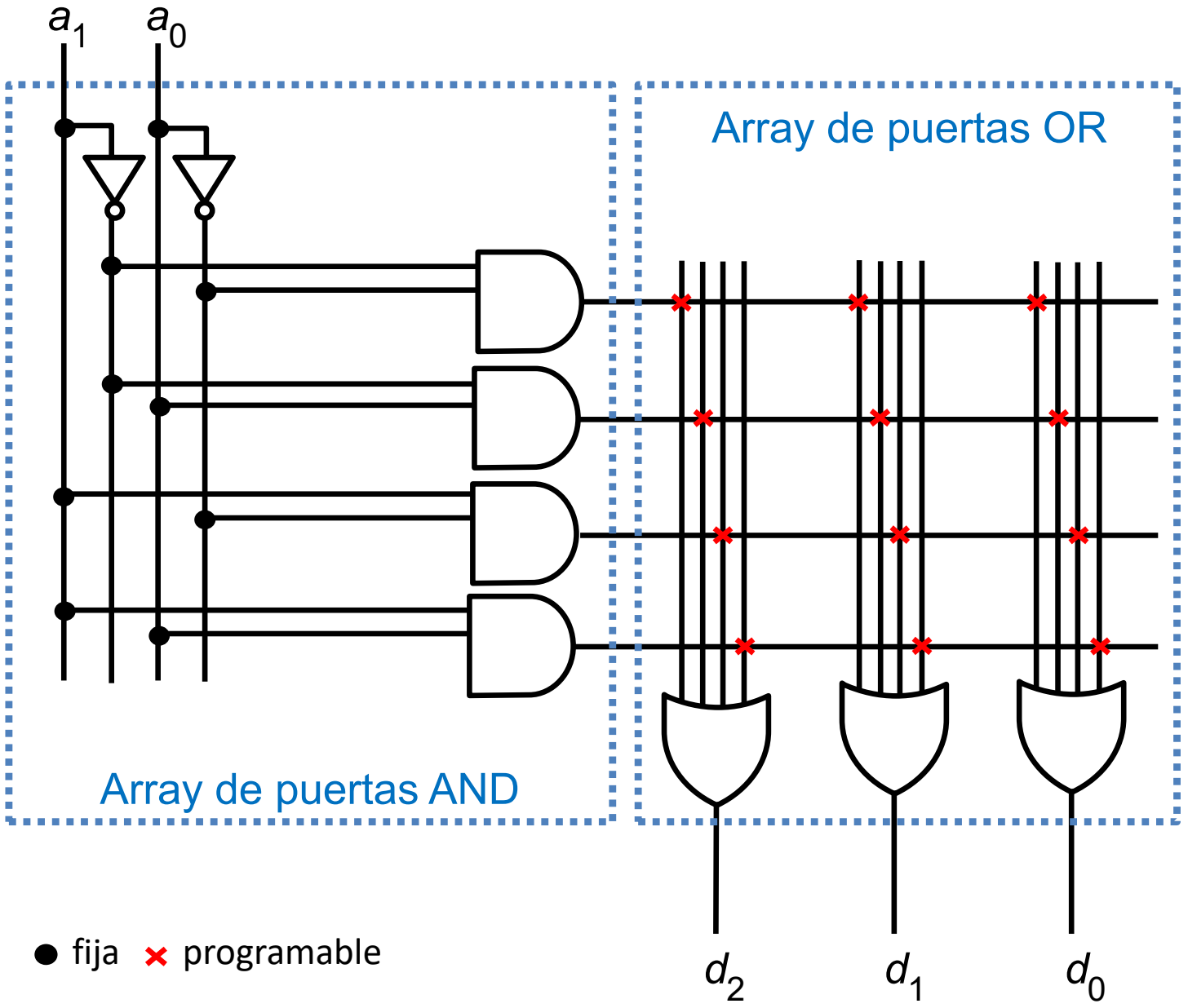
---

memoria no volátil de capaz de almacenar 2^k palabras de n bits cada una

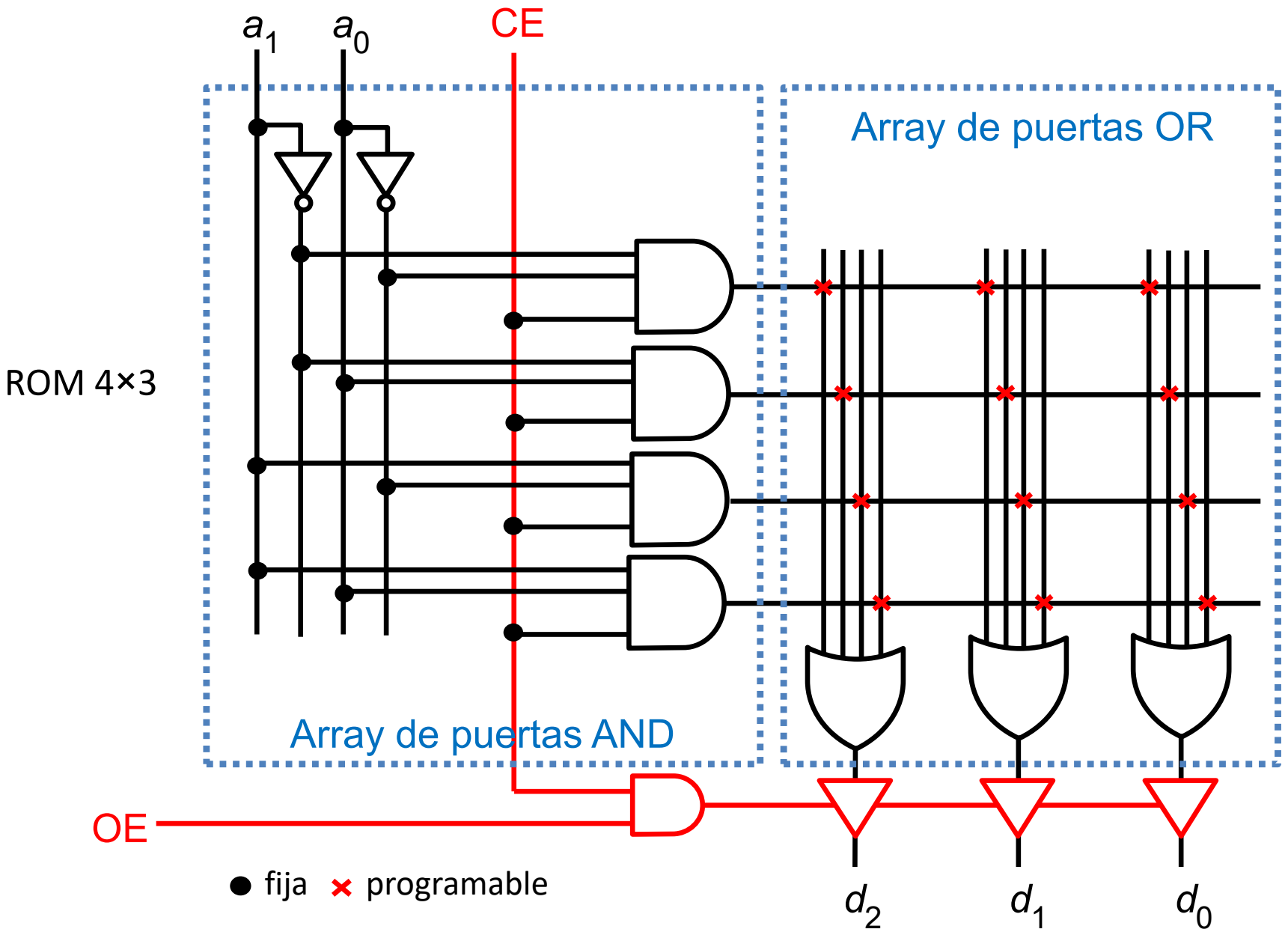


ROM (Read Only Memory)

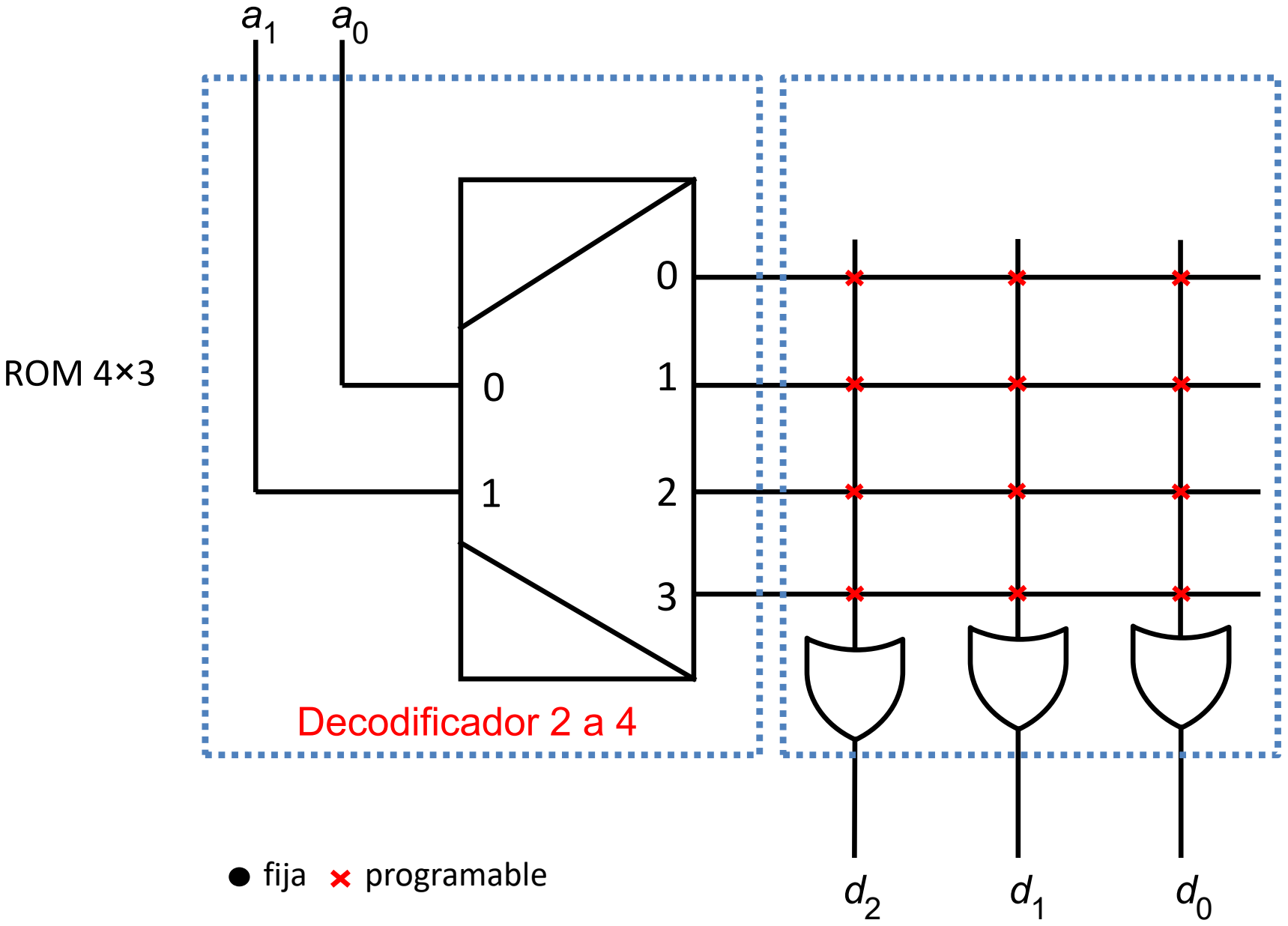
ROM 4×3



ROM (Read Only Memory)



ROM (Read Only Memory)



ROM (Read Only Memory)

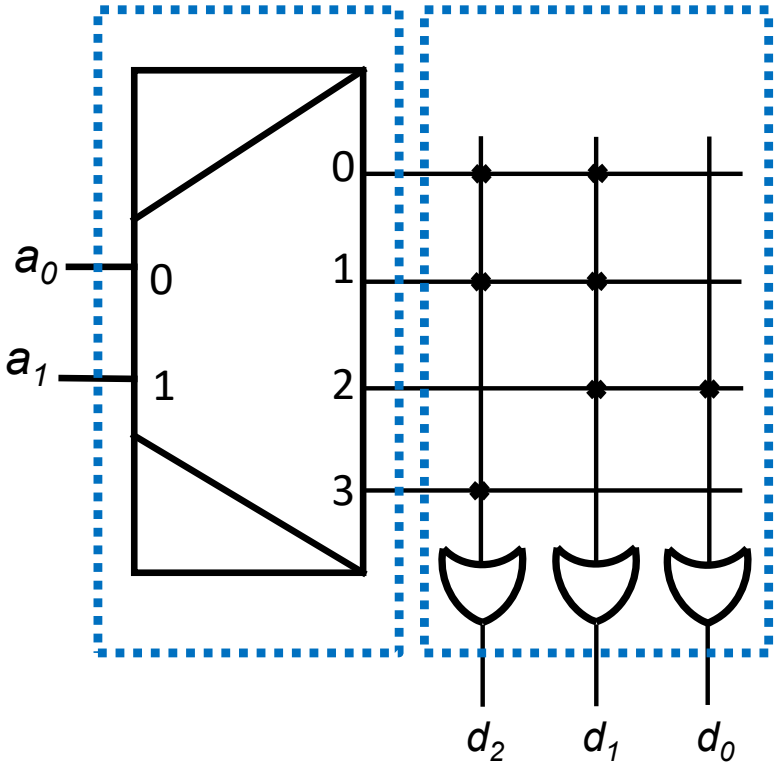
- Aplicaciones al diseño:
  - Implementar directamente FC almacenando su tabla de verdad.

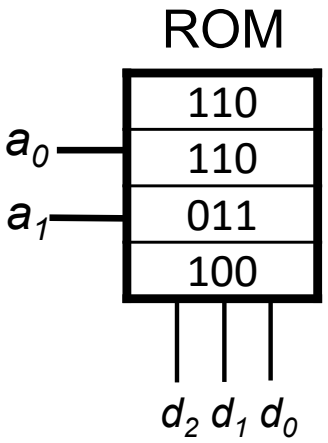
	a_1	a_0	d_2	d_1	d_0
0	0	0	1	1	0
1	0	1	1	1	0
2	1	0	0	1	1
3	1	1	1	0	0

$$d_2 = \overline{a}_1 + a_0$$

$$d_1 = \overline{a_1 \cdot a_0}$$

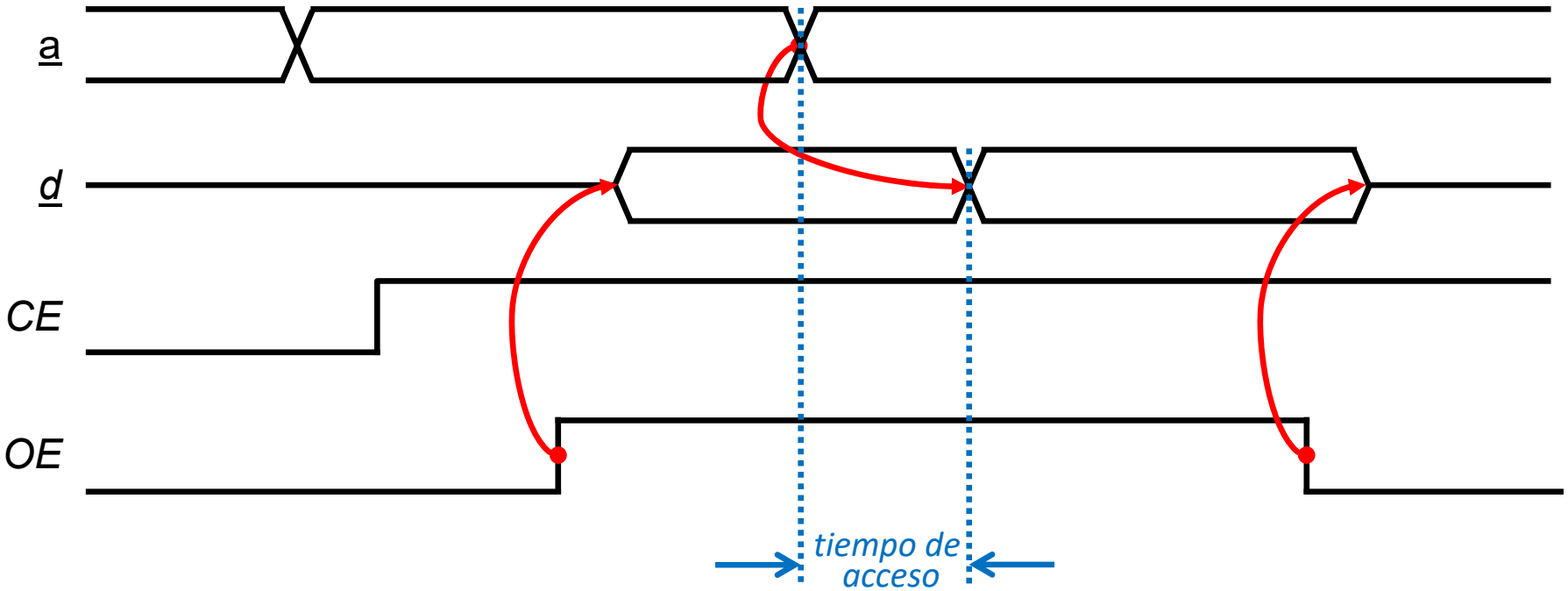
$$d_0 = a_1 \cdot \overline{a}_0$$





ROM (Read Only Memory)

- Cuando a cambia, la ROM recupera el dato correspondiente
  - El dato solo aparece en d si CE y OE están activadas



ROM (Read Only Memory)

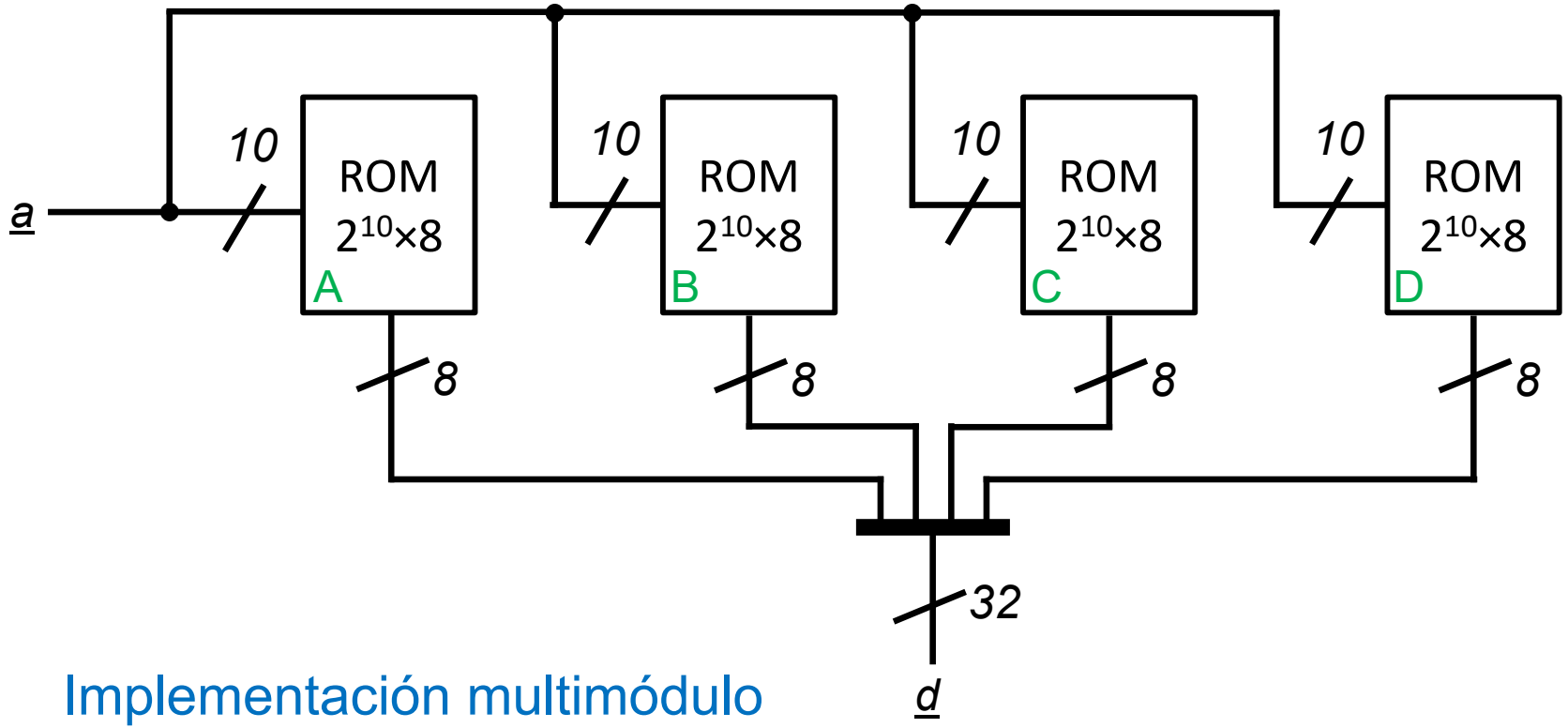
- Mask Programmable ROM
  - Se programa durante la fabricación del chip.
  - No puede borrarse/reprogramarse.
- PROM (Programmable ROM)
  - Se programa eléctricamente usando un programador.
  - No puede borrarse/reprogramarse.
- EPROM (Erasable Programmable ROM)
  - Se programa eléctricamente usando un programador.
  - Se borra (chip completo) exponiéndola a luz ultravioleta.
- EEPROM (Electrically Erasable Programmable ROM)
  - Se programa/borra (palabra) eléctricamente usando un programador.
- Flash memory
  - Se programa/borra (bloque) eléctricamente sin requerir programador.

ROM (Read Only Memory)

- La capacidad de las memorias se mide en bytes (8 bits)
  - Cuando el número de bytes es alto, se utilizan prefijos.
- Históricamente, los prefijos indican cantidades potencias de 2
  - Kilobyte (KB) = 2^{10} bytes = 1.024 bytes
  - Megabyte (MB) = 2^{20} bytes = 1.048.576 bytes
  - Gigabyte (GB) = 2^{30} bytes = 1.073.741.824 bytes
- Sin embargo, desde hace algunos años su significado se ha homogeneizado con el definido en el Sistema Internacional de unidades (potencias de 10)
  - Kilobyte (kB) = 10^3 bytes = 1.000 bytes
  - Megabyte (MB) = 10^6 bytes = 1.000.000 bytes
  - Gigabyte (GB) = 10^9 bytes = 1.000.000.000 bytes
  - Y se han definido nuevos prefijos para indicar las potencias de 2
    - Kibibyte (KiB) = 2^{10} bytes = 1.024 bytes
    - Mebibyte (MiB) = 2^{20} bytes = 1.048.576 bytes
    - Gibibyte (GiB) = 2^{30} bytes = 1.073.741.824 bytes
  - No obstante, todavía no está generalizado el uso de los nuevos prefijos .

ROM (Read Only Memory)

- Varias ROM se pueden componer para comportarse como una ROM de mayor anchura de palabra.

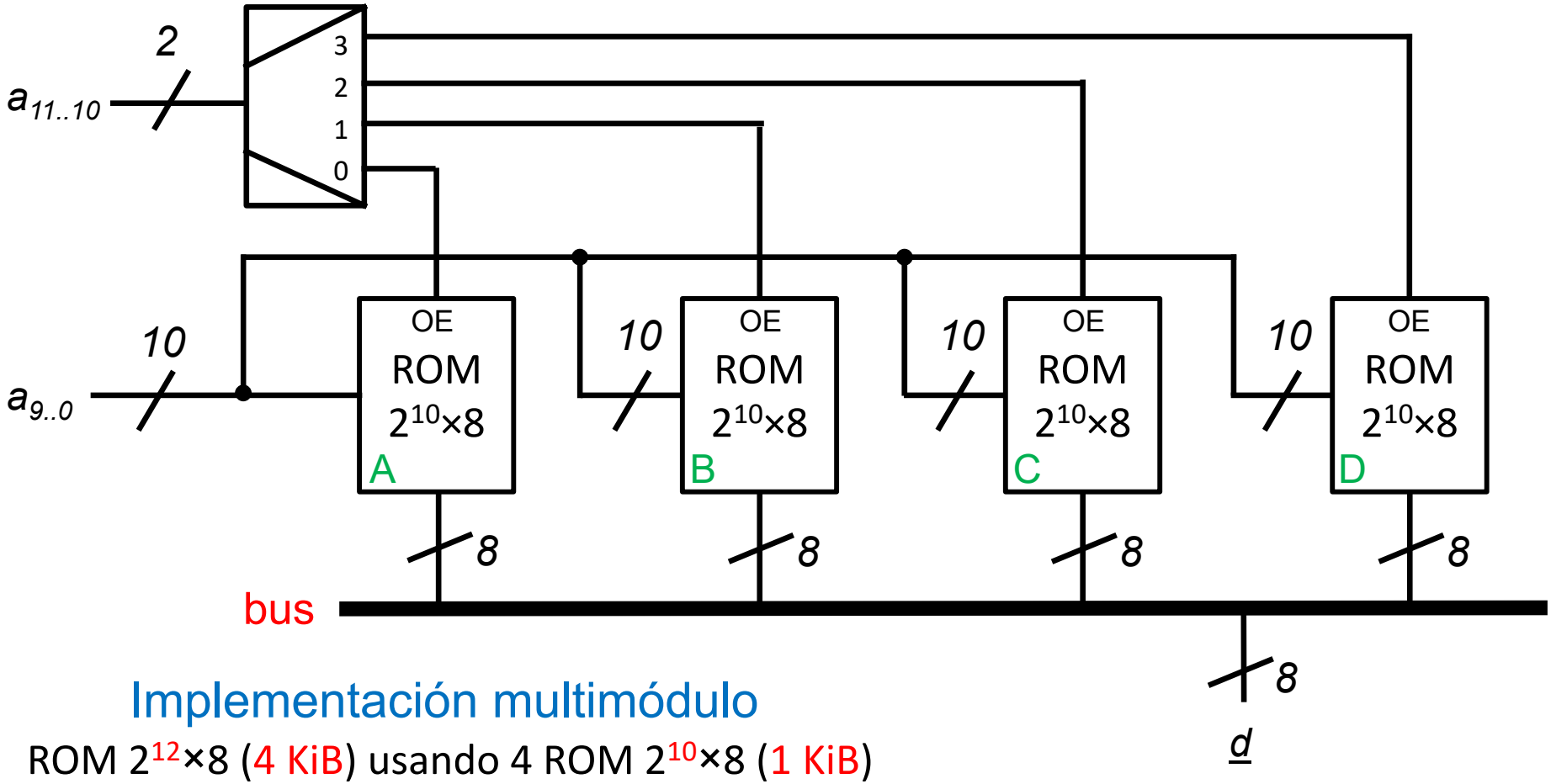


Implementación multimódulo

ROM 2^{10}×32 (4 KiB) usando 4 ROM 2^{10}×8 (1 KiB)

ROM (Read Only Memory)

- Varias ROM se pueden componer para comportarse como una ROM de mayor profundidad.

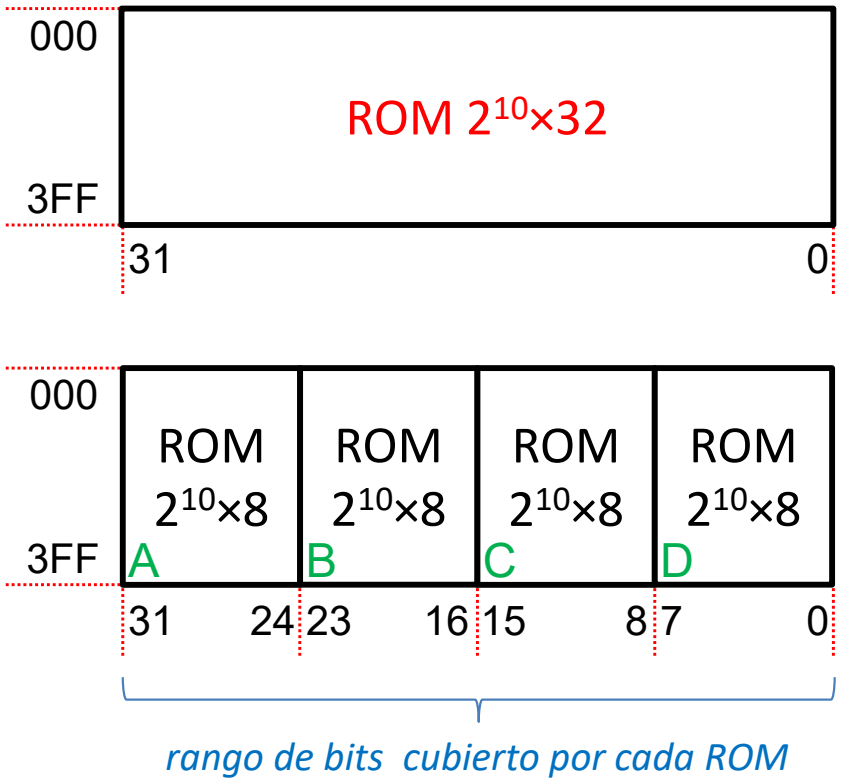


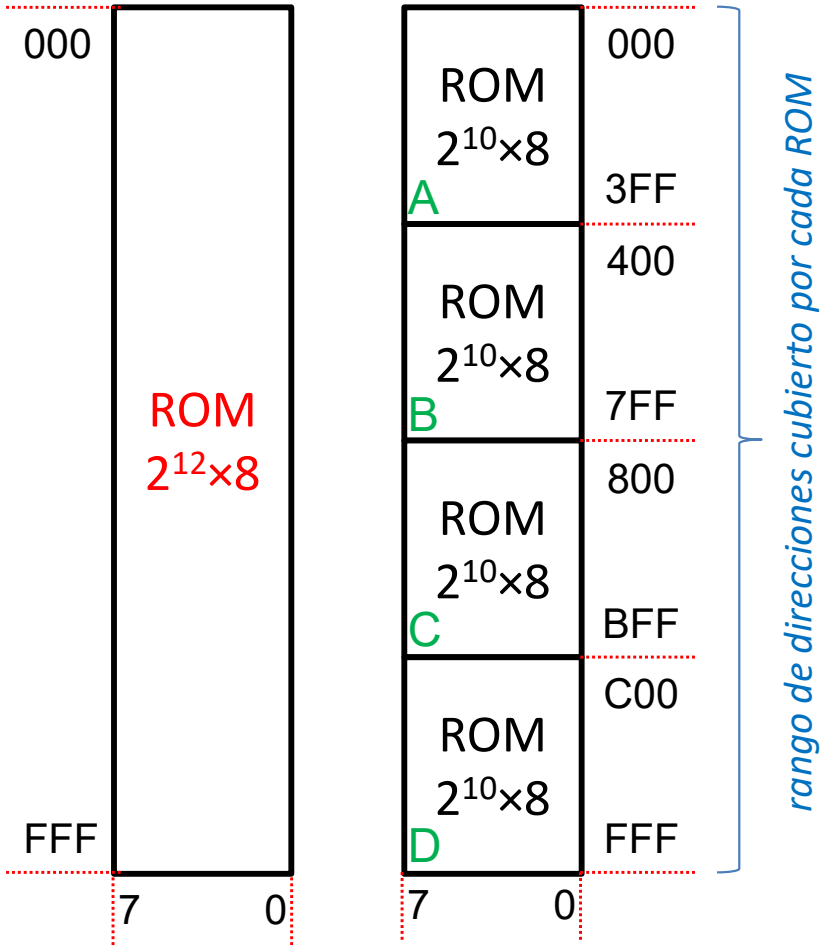
Implementación multimódulo

ROM $2^{12}\times8$ (4 KiB) usando 4 ROM $2^{10}\times8$ (1 KiB)

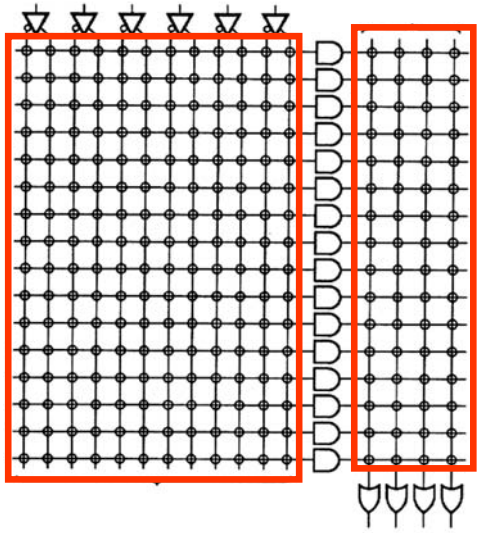
ROM (Read Only Memory)

- Distintas organizaciones de ROM dan lugar a distintos mapas de memoria:

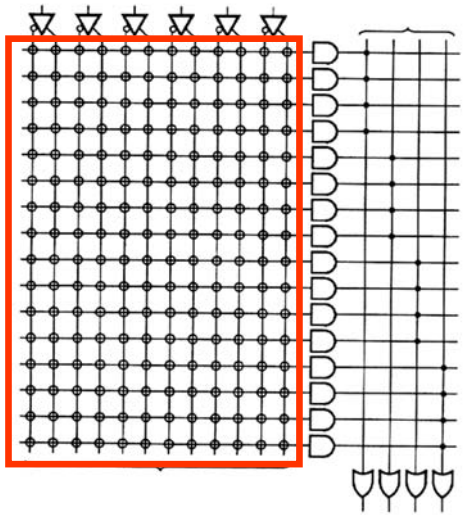




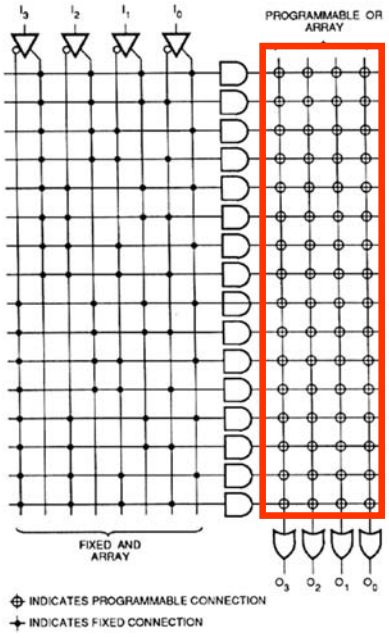
Otros dispositivos programables



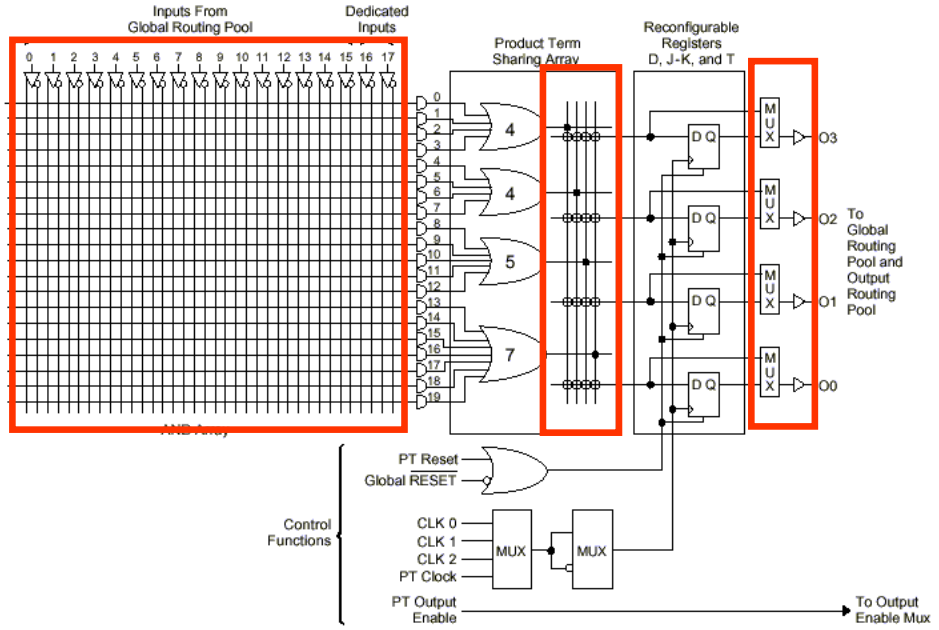
PLA



PAL



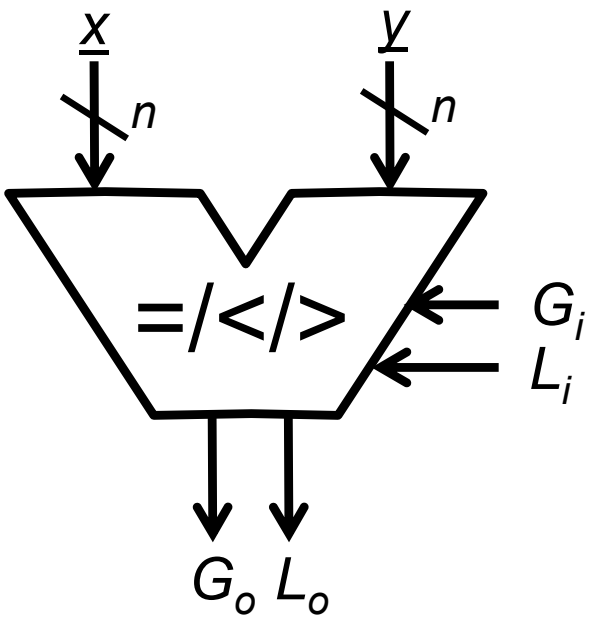
ROM



CPLD

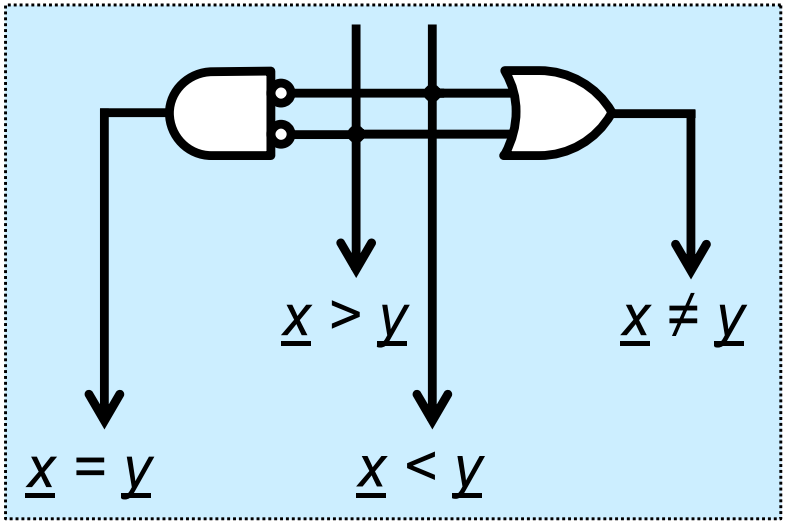
interconexiones (re)programables

Comparador de magnitud



$\underline{x}$, $\underline{y}$	2 entradas de datos de n bits
G_i, L_i	2 entrada de acarreo
G_o, L_o	2 salidas de comparación

compara las magnitudes de $\underline{x}$ e $\underline{y}$



$$G_o = \begin{cases} 1 & \text{si } (\underline{x}>\underline{y}) \text{ o } (\underline{x}=\underline{y} \text{ y } G_i>L_i) \\ 0 & \text{en otro caso} \end{cases}$$

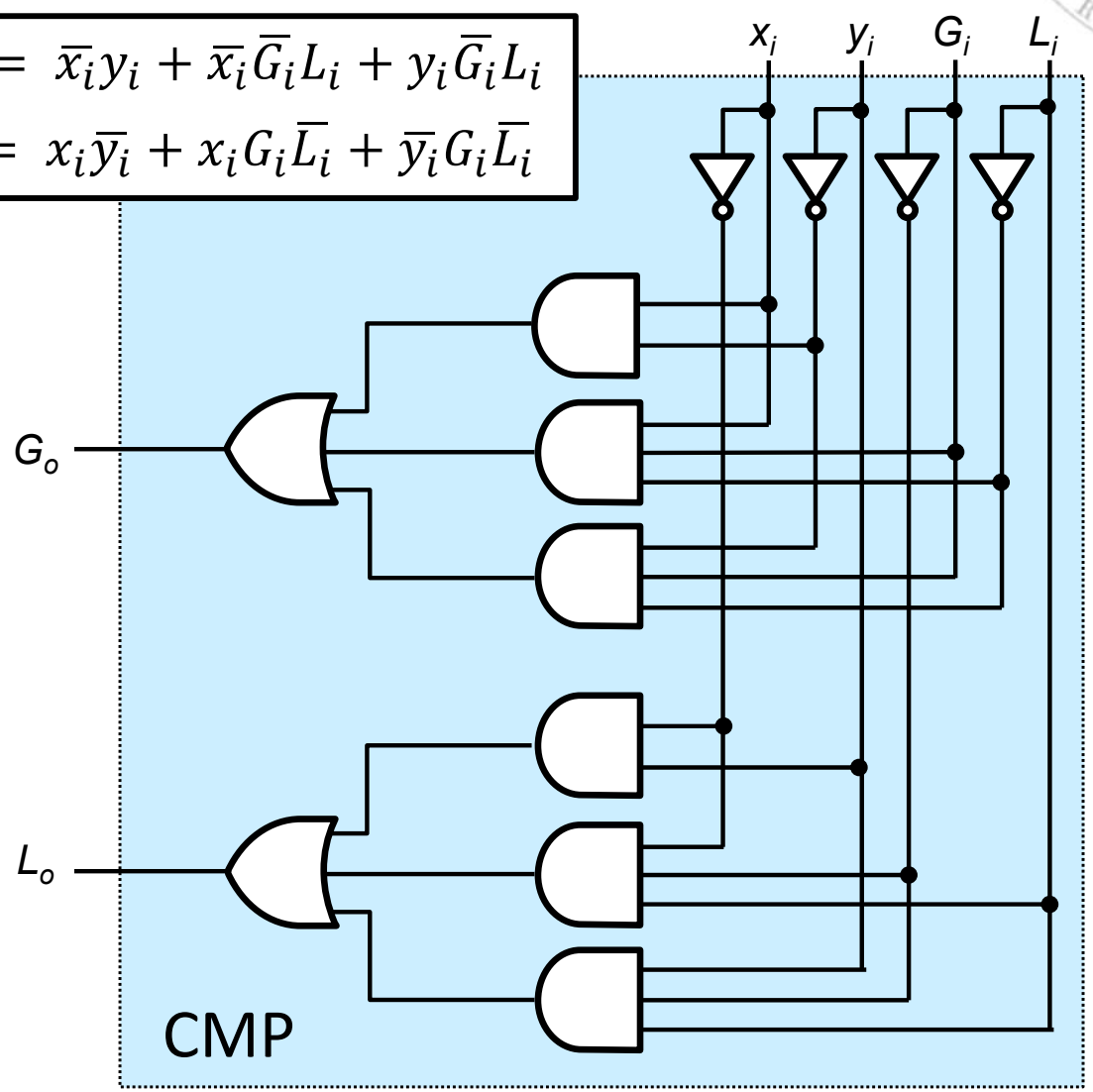
$$L_o = \begin{cases} 1 & \text{si } (\underline{x}<\underline{y}) \text{ o } (x=\underline{y} \text{ y } G_i<L_i) \\ 0 & \text{en otro caso} \end{cases}$$

Comparador de magnitud

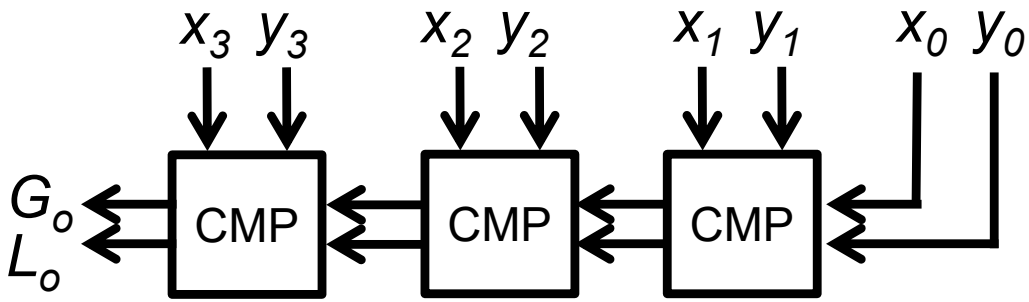
x_i	y_i	G_i	L_i	G_o	L_o
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	0	0

$$L_o = \overline{x_i} y_i + \overline{x_i}\,\overline{G_i} L_i + y_i \overline{G_i} L_i$$

$$G_o = x_i \overline{y_i} + x_i G_i \overline{L_i} + \overline{y_i} G_i \overline{L_i}$$

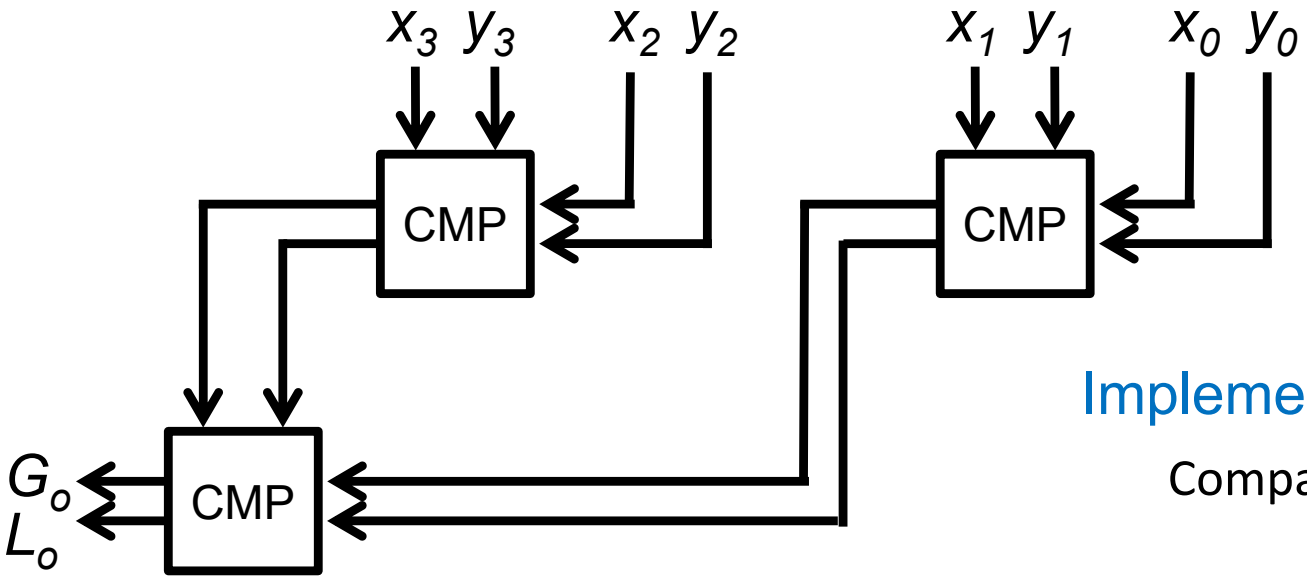


Comparador de magnitud



Implementación en serie

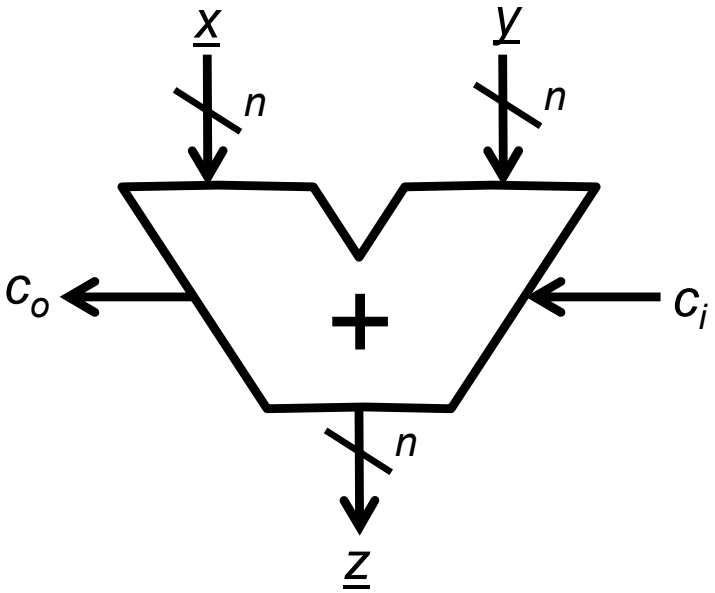
Comparador de 4 bits



Implementación en árbol

Comparador de 4 bits

Sumador



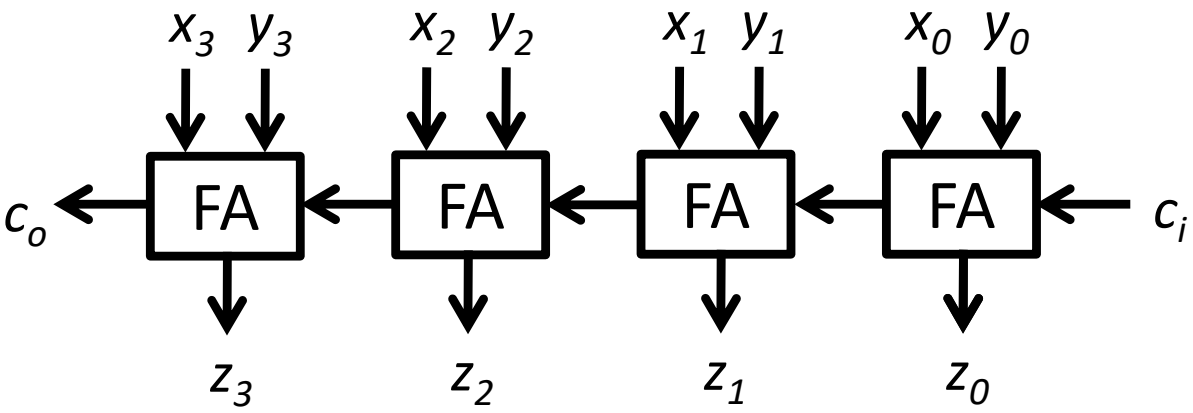
$\underline{x}$, $\underline{y}$	2 entradas de datos de n bits
c_i	1 entrada de acarreo (carry)
$\underline{z}$	1 salida de datos de n bits
c_o	1 salida de acarreo (carry)

realiza la suma binaria: $\underline{x} + \underline{y} + c_i$

$$\underline{z} = (\underline{x} + \underline{y} + c_i) \bmod 2^n$$

$$c_o = \begin{cases} 1 & \text{si } (\underline{x} + \underline{y} + c_i) \geq 2^n \\ 0 & \text{en otro caso} \end{cases}$$

Sumador



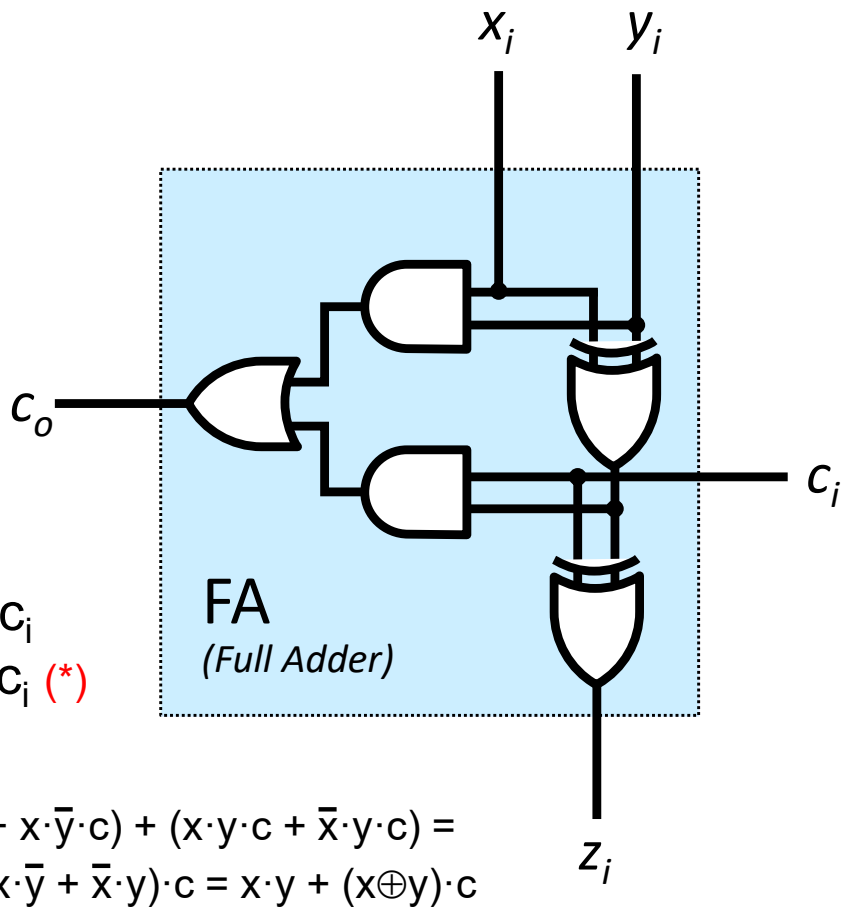
Implementación con propagación de acarreos

Sumador de 4 bits

c_i	x_i	y_i	c_o	z_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$z_i = (x_i \oplus y_i) \oplus c_i$$

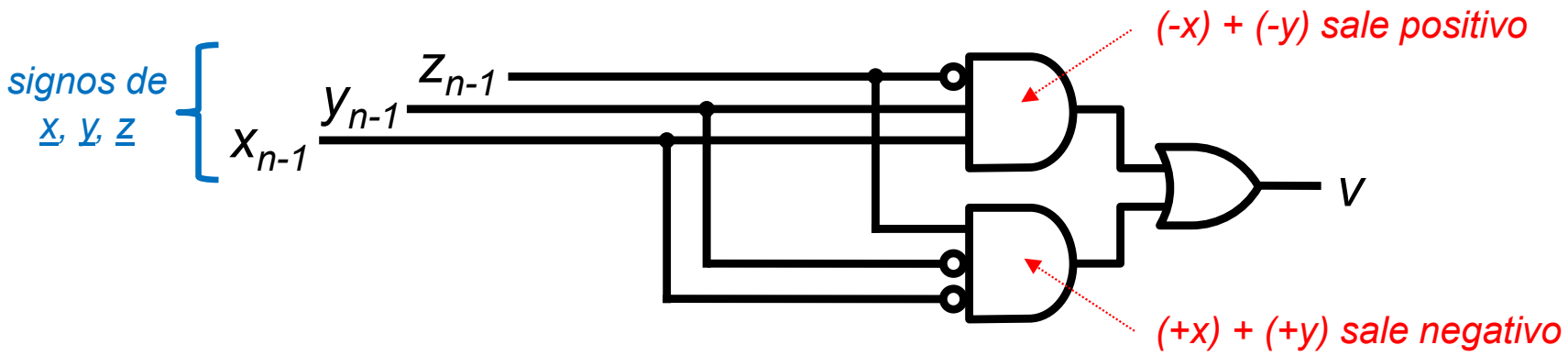
$$c_o = x_i \cdot y_i + x_i \cdot c_i + y_i \cdot c_i$$
$$= x_i \cdot y_i + (x_i \oplus y_i) \cdot c_i \quad (*)$$



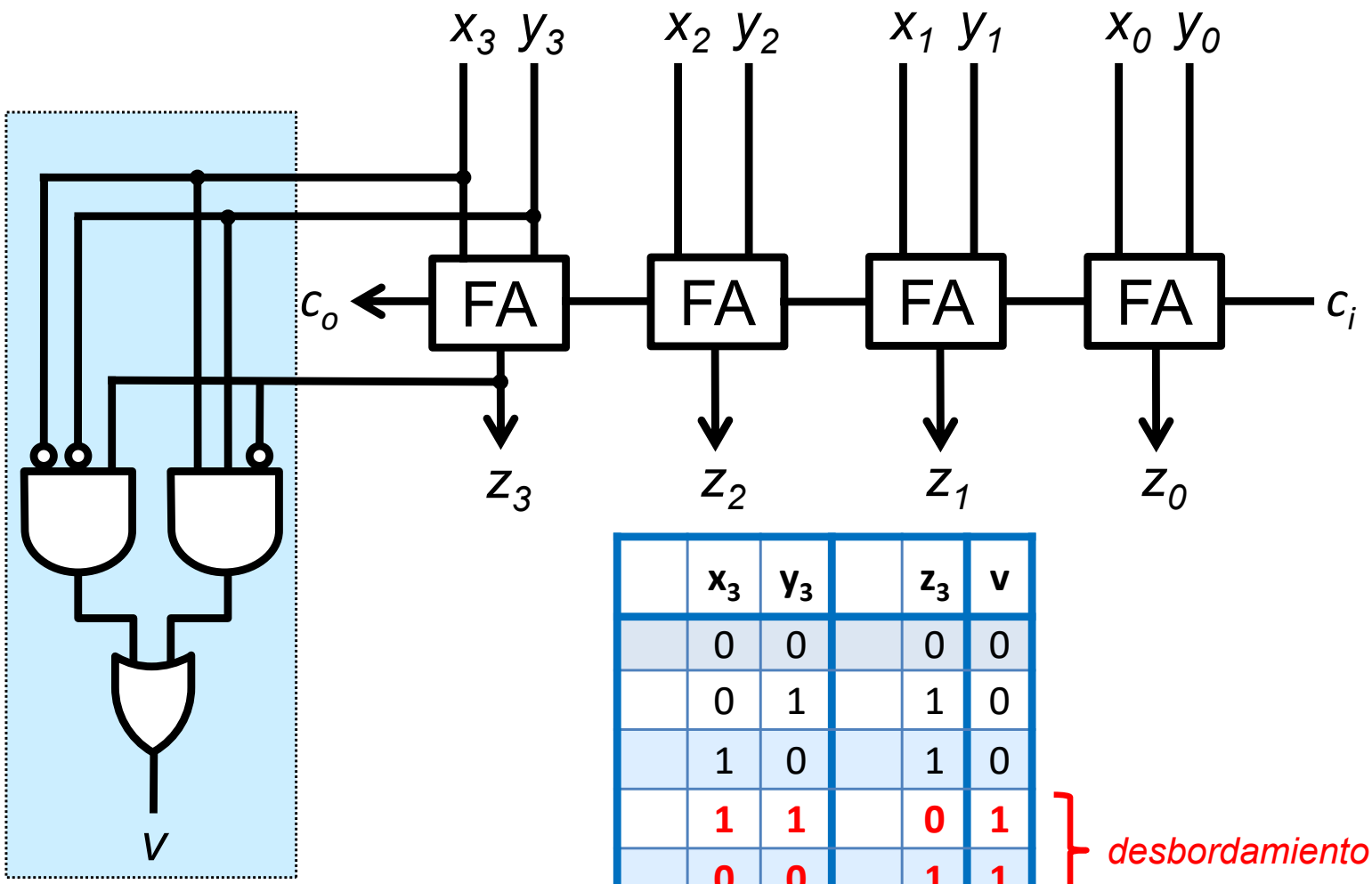
(*) $x \cdot y + x \cdot c + y \cdot c = x \cdot y + (x \cdot y \cdot c + x \cdot \bar{y} \cdot c) + (x \cdot y \cdot c + \bar{x} \cdot y \cdot c) =$
$= x \cdot y + x \cdot \bar{y} \cdot c + \bar{x} \cdot y \cdot c = x \cdot y + (x \cdot \bar{y} + \bar{x} \cdot y) \cdot c = x \cdot y + (x \oplus y) \cdot c$

Sumador

- El sumador permite sumar tanto 2 números codificados en binario puro como 2 números codificados en C2
  - El resultado será coherente con la codificación de las entradas.
- Si los números son sin signo (binario puro)
  - c_o indica si se produce desbordamiento.
- Si los números son con signo (en C2)
  - El desbordamiento se detecta chequeando si el signo del resultado es coherente con el signo de los operandos.



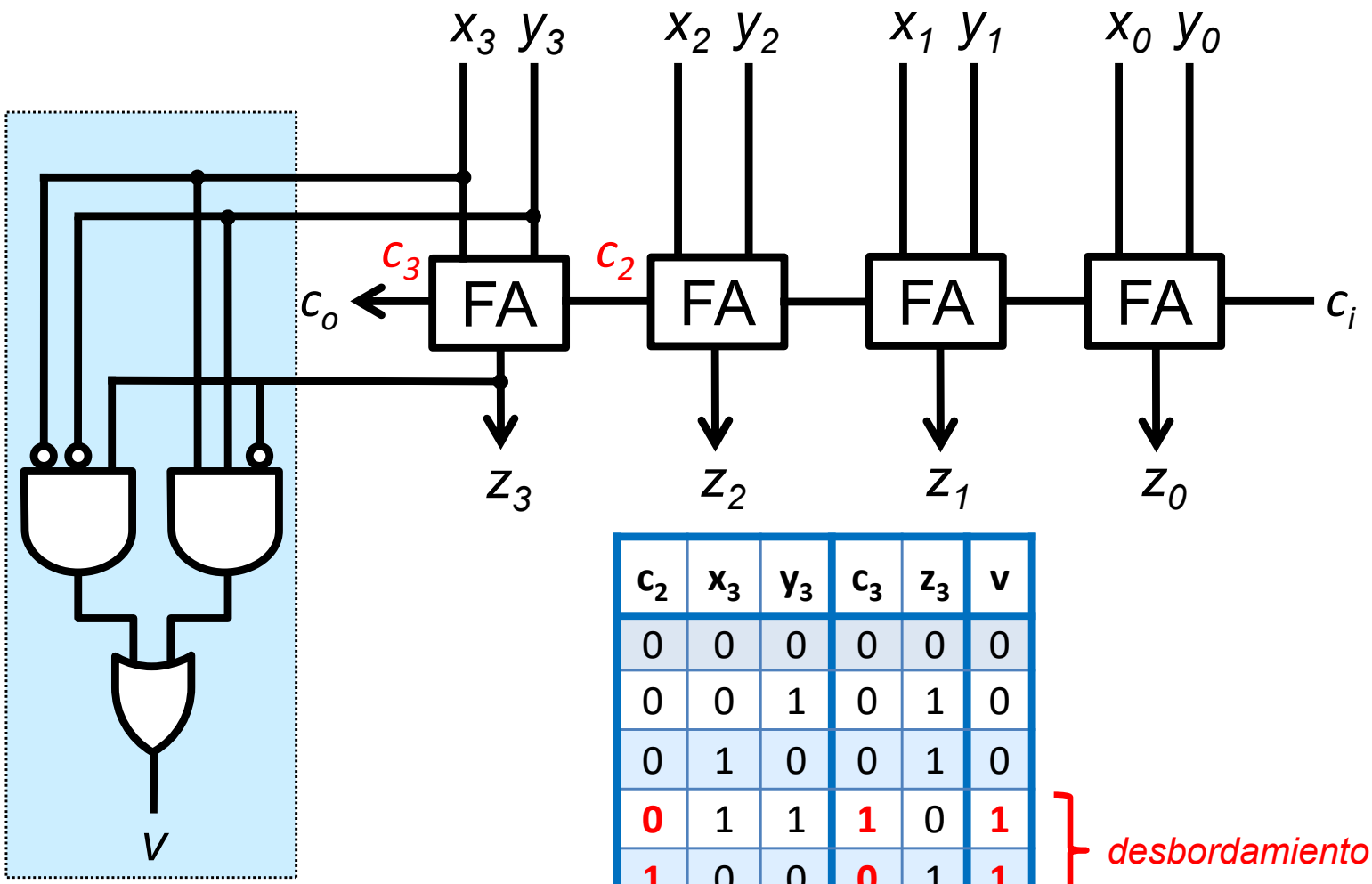
Sumador



	x_3	y_3		z_3	v
	0	0		0	0
	0	1		1	0
	1	0		1	0
	1	**1**		**0**	**1**
	0	**0**		**1**	**1**
	0	1		0	0
	1	0		0	0
	1	1		1	0

desbordamiento C2

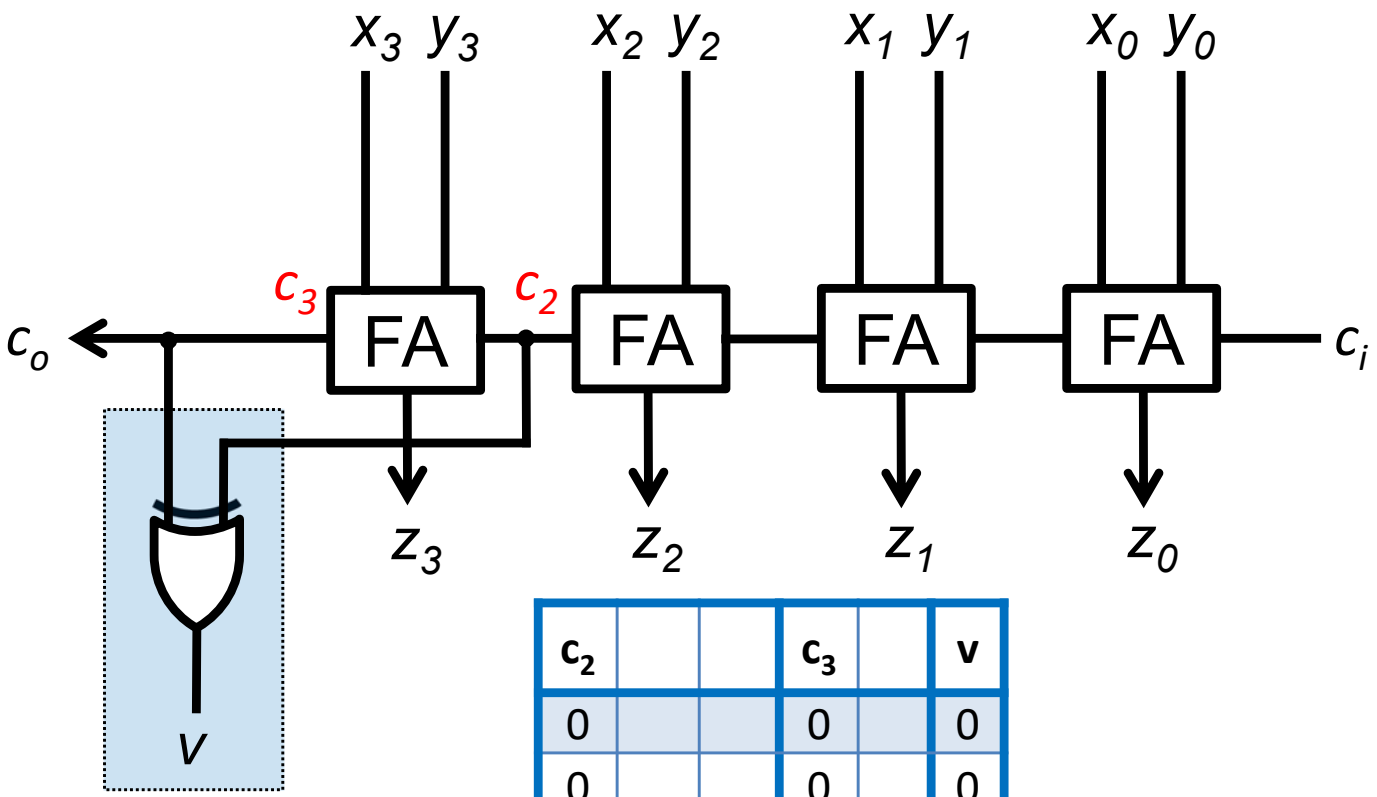
Sumador



c_2	x_3	y_3	c_3	z_3	v
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	**1**	0	**1**
1	0	0	**0**	1	**1**
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	0

desbordamiento C2

Sumador

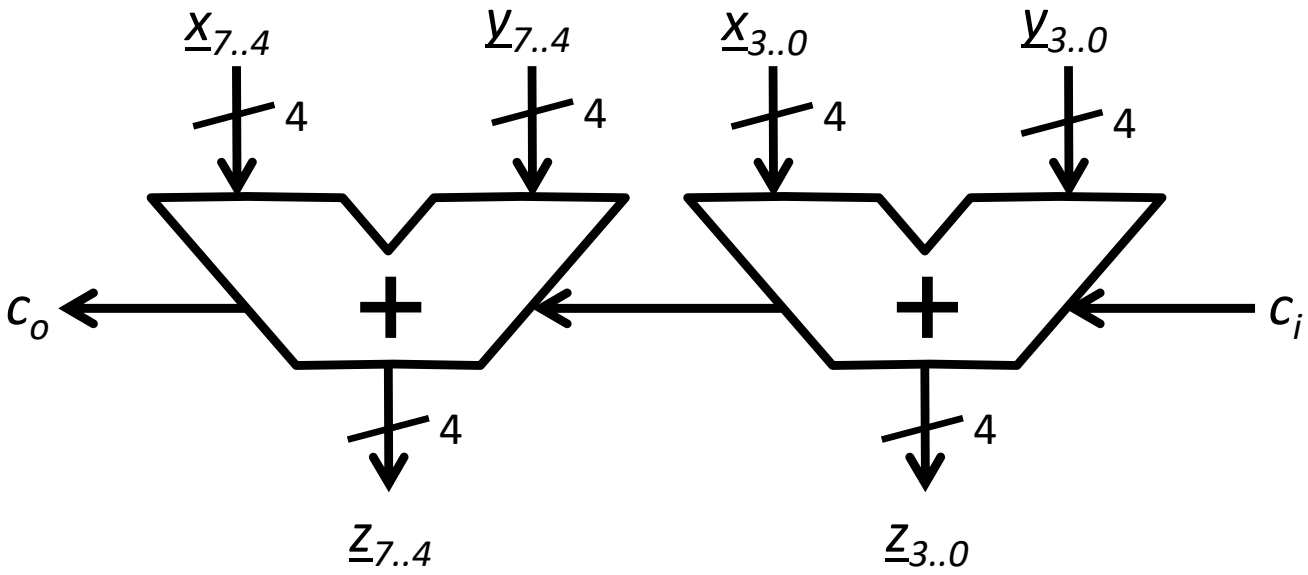


c_2			c_3		v
0			0		0
0			0		0
0			0		0
0			**1**		**1**
1			**0**		**1**
1			1		0
1			1		0
1			1		0

desbordamiento C2

Sumador

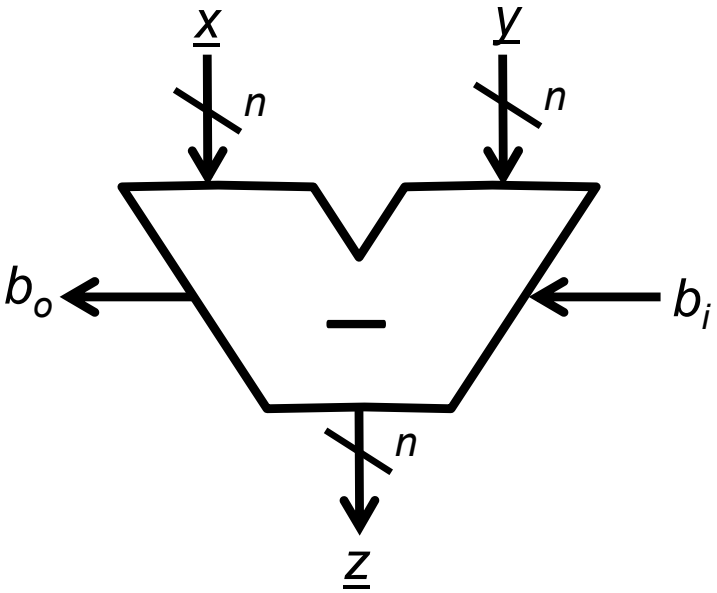
- Varios sumadores se pueden componer en serie para para comportarse como un sumador de mayor anchura.



Implementación serie

Sumador de 8 bits

Restador



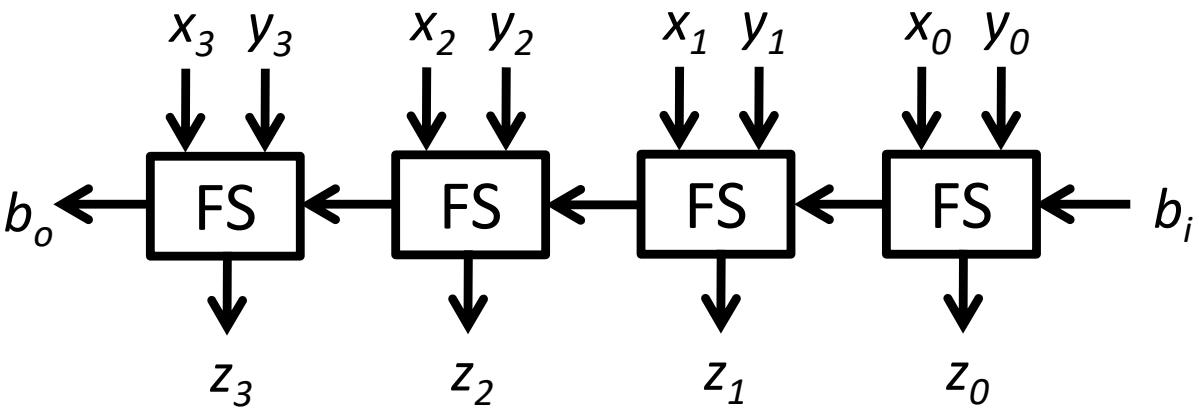
$\underline{x}$, $\underline{y}$	2 entradas de datos de n bits
b_i	1 entrada de acarreo (borrow)
$\underline{z}$	1 salida de datos de n bits
b_o	1 salida de acarreo (borrow)

realiza la resta binaria: $\underline{x} - \underline{y} - b_i$

$$\underline{z} = (\underline{x} - \underline{y} - b_i) \bmod 2^n$$

$$b_o = \begin{cases} 1 & \text{si } (\underline{x} - \underline{y} - b_i) < 0 \\ 0 & \text{en otro caso} \end{cases}$$

Restador



Implementación con propagación de acarreos

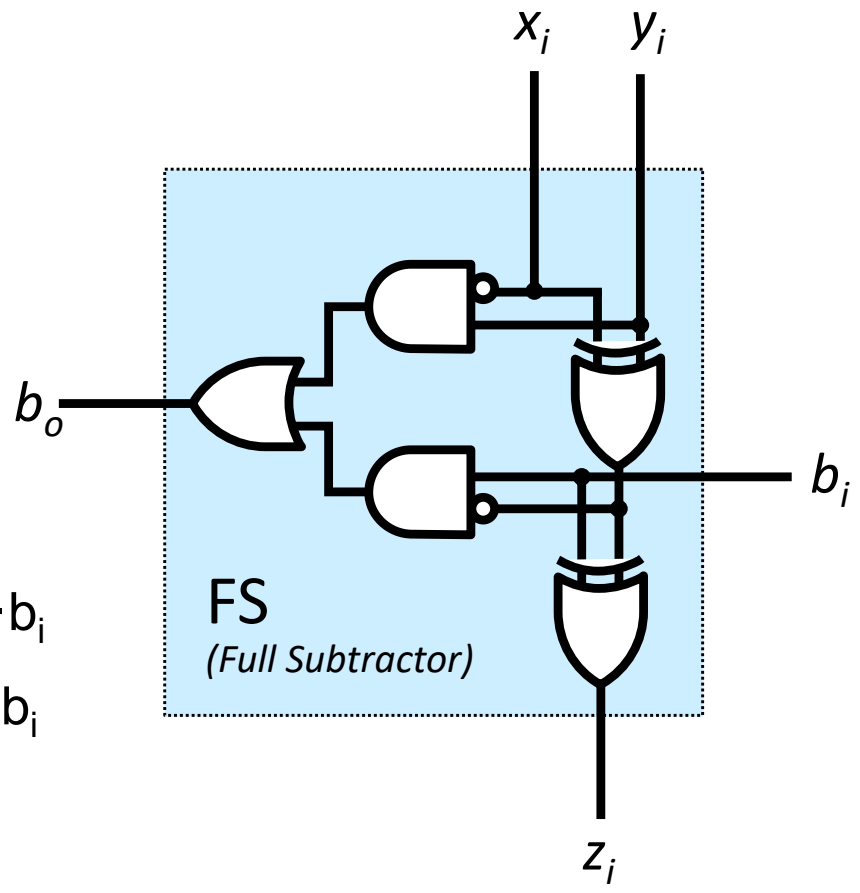
Restador de 4 bits

b_i	x_i	y_i	b_o	z_i
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

$$z_i = (x_i \oplus y_i) \oplus b_i$$

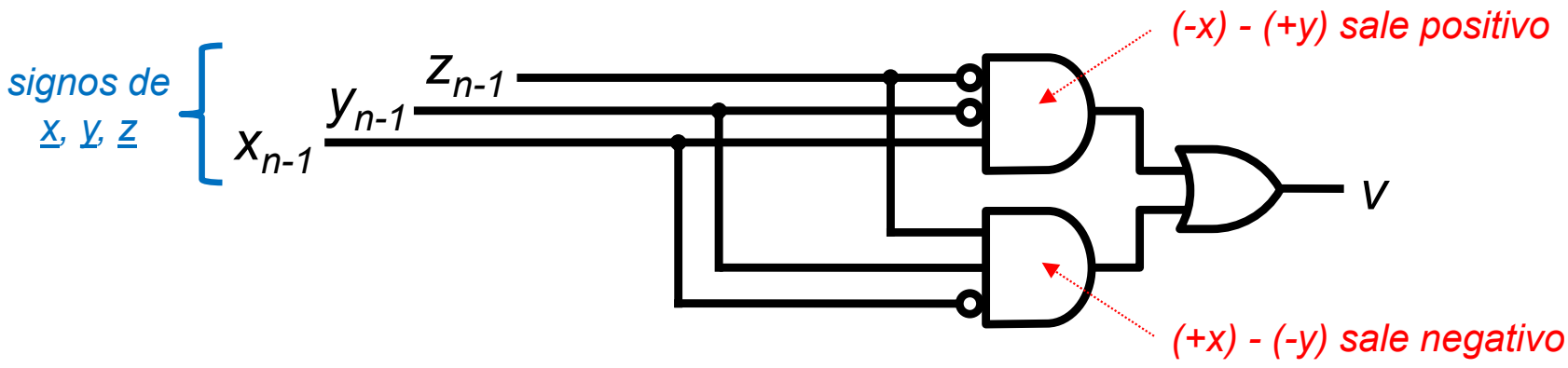
$$b_o = \bar{x}_i \cdot y_i + \bar{x}_i \cdot b_i + y_i \cdot b_i$$

$$= \bar{x}_i \cdot y_i + \overline{(x_i \oplus y_i)} \cdot b_i$$

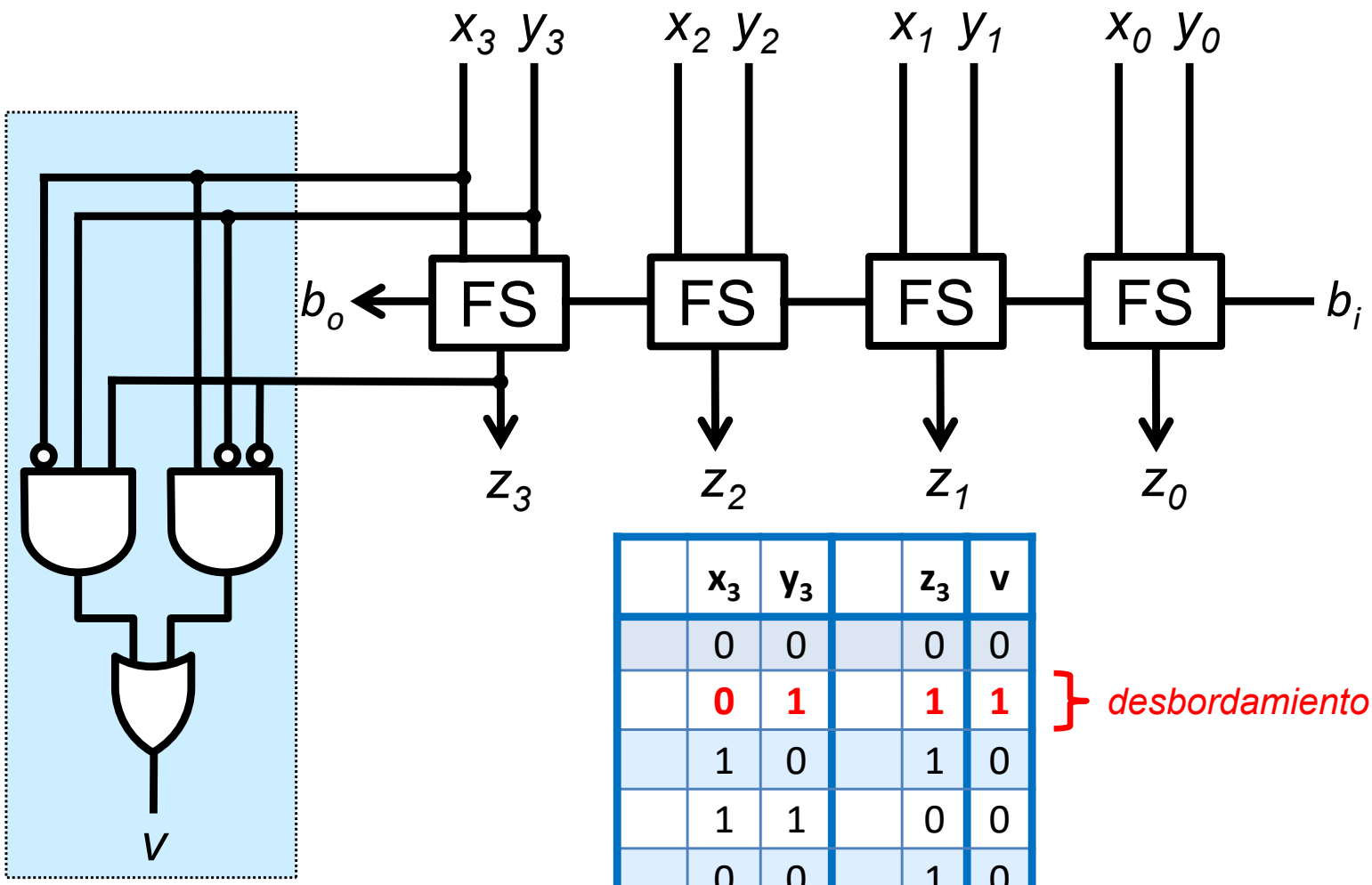


Restador

- El restador permite restar tanto 2 números codificados en binario puro como 2 números codificados en C2
  - El resultado será coherente con la codificación de las entradas.
- Si los números son sin signo (binario puro)
  - b_o indica si se produce desbordamiento.
- Si los números son con signo (en C2)
  - El desbordamiento se detecta chequeando si el signo del resultado es coherente con el signo de los operandos.

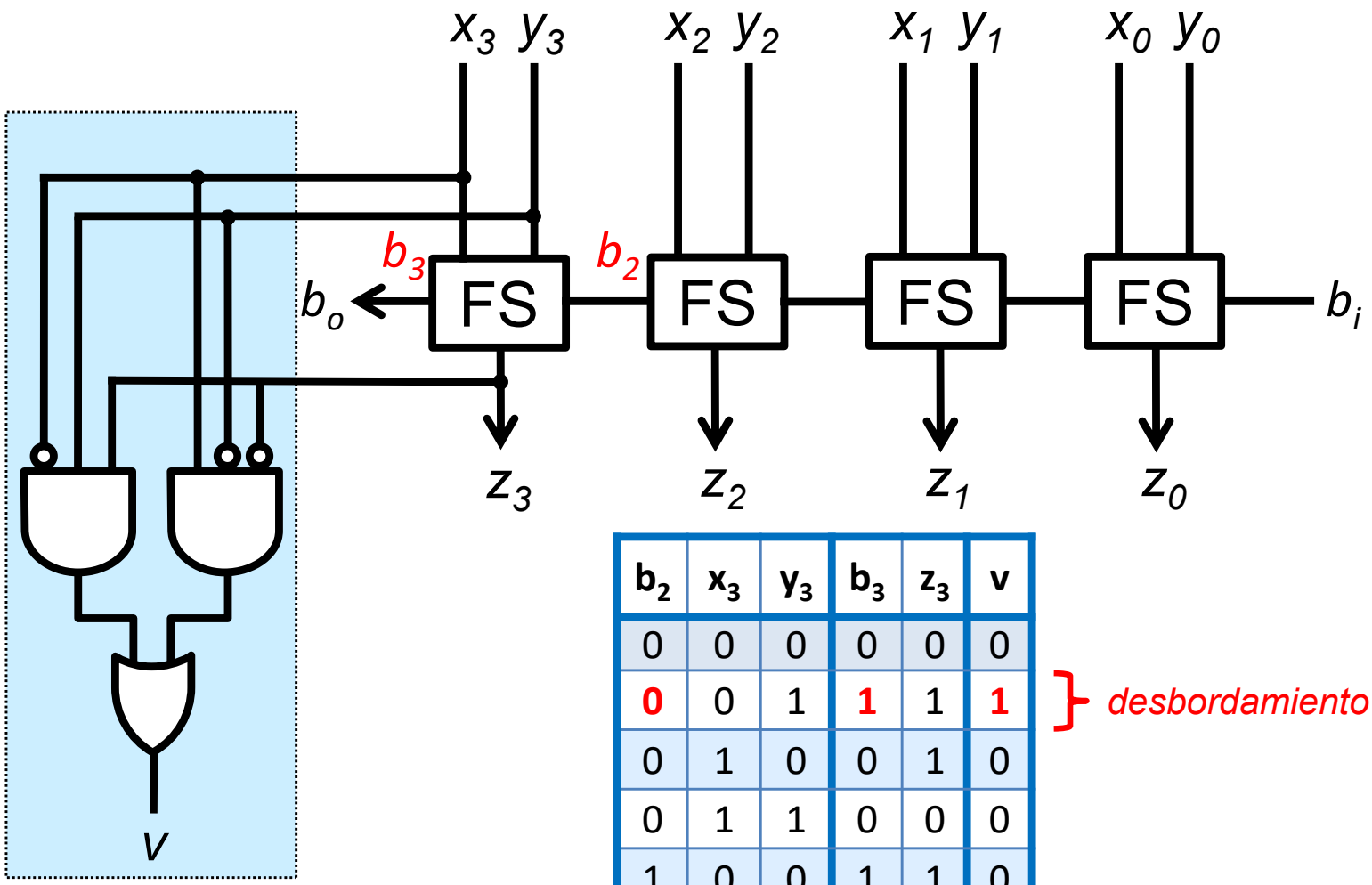


Restador



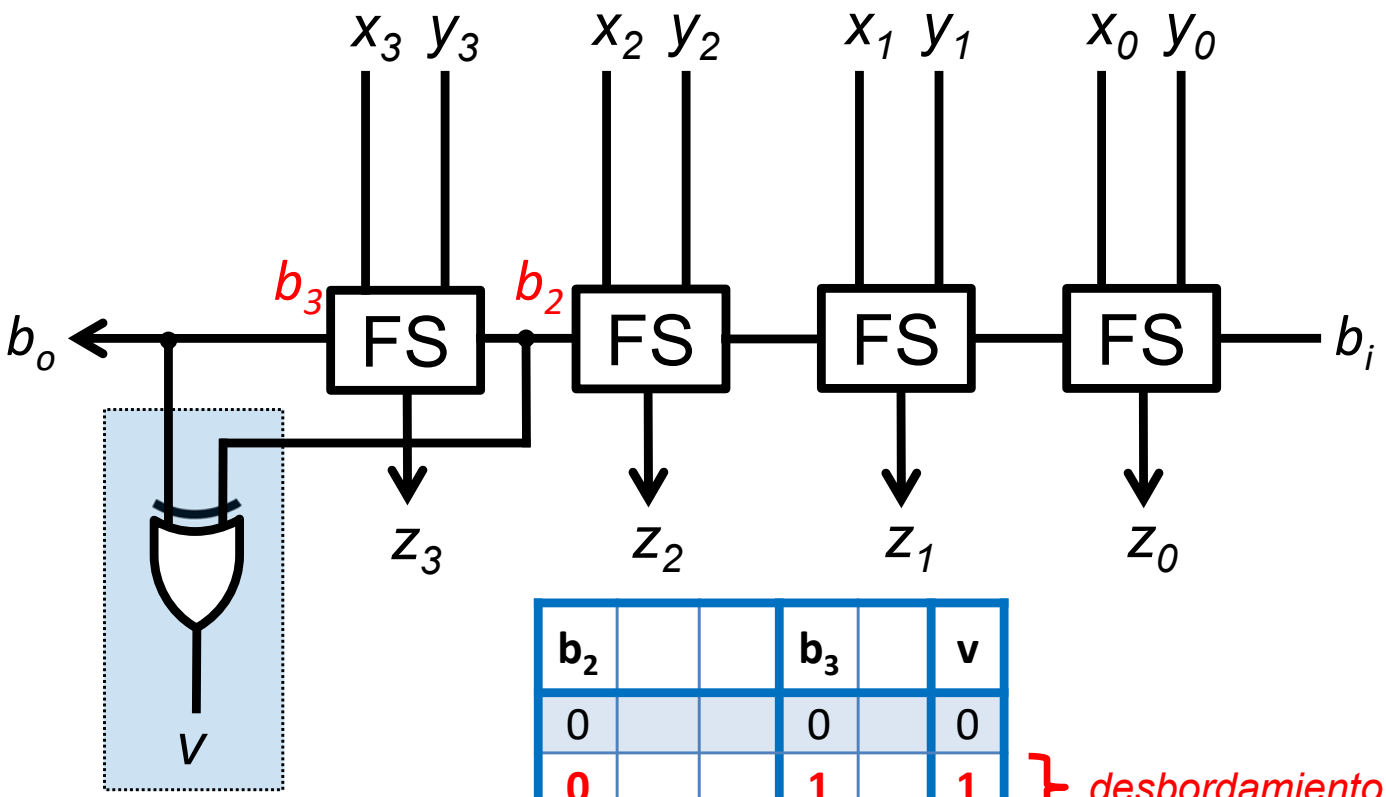
	x_3	y_3		z_3	v	
	0	0		0	0	
	0	**1**		**1**	**1**	} desbordamiento C2
	1	0		1	0	
	1	1		0	0	
	0	0		1	0	
	0	1		0	0	
	1	**0**		**0**	**1**	} desbordamiento C2
	1	1		1	0	

Restador



b_2	x_3	y_3	b_3	z_3	v	
0	0	0	0	0	0	
0	0	1	**1**	1	**1**	} desbordamiento C2
0	1	0	0	1	0	
0	1	1	0	0	0	
1	0	0	1	1	0	
1	0	1	1	0	0	
1	1	0	**0**	0	**1**	} desbordamiento C2
1	1	1	1	1	0	

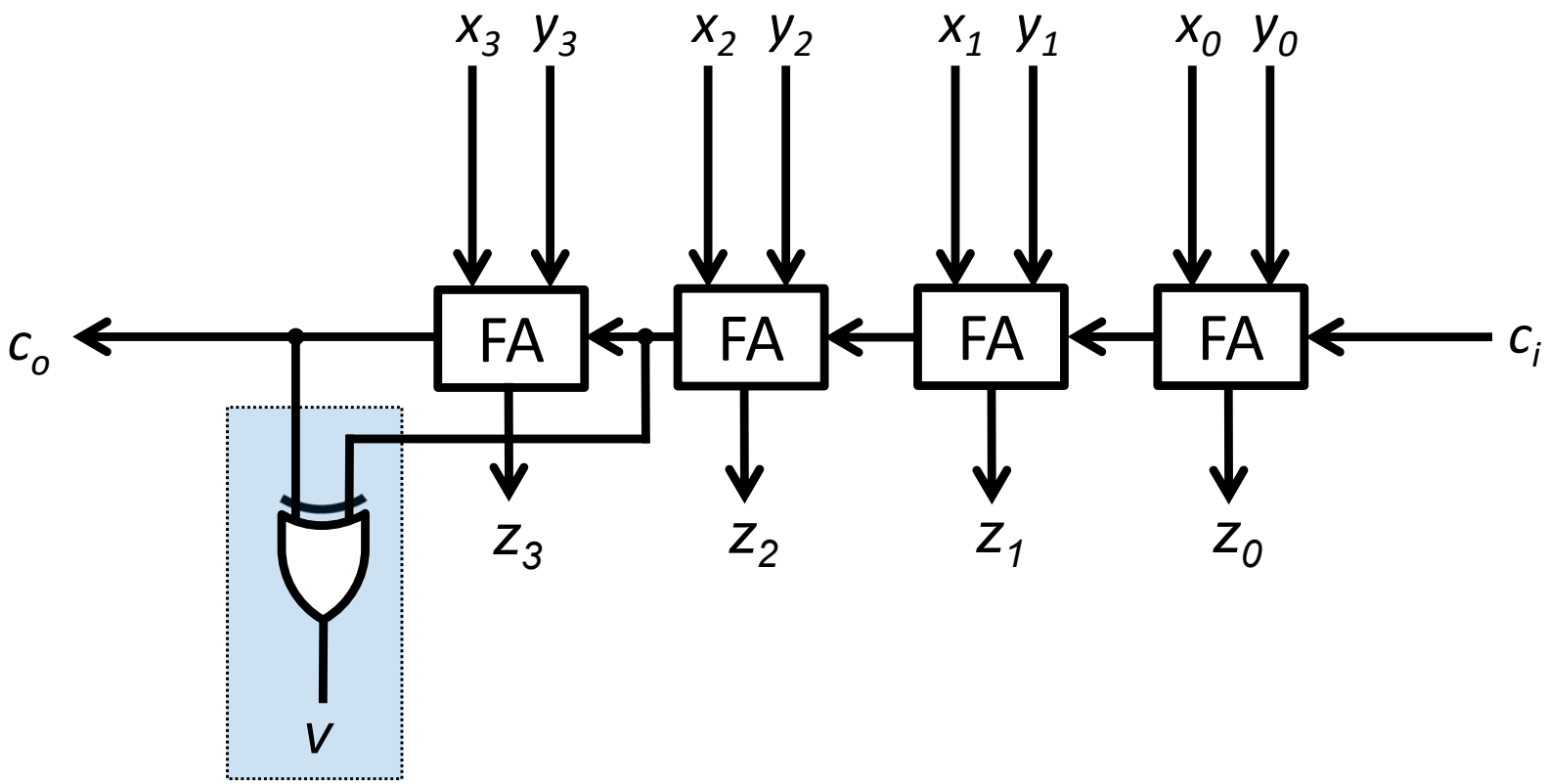
Restador



b_2			b_3		v	
0			0		0	
0			**1**		**1**	} desbordamiento C2
0			0		0	
0			0		0	
1			1		0	
1			1		0	
1			**0**		**1**	} desbordamiento C2
1			1		0	

Restador

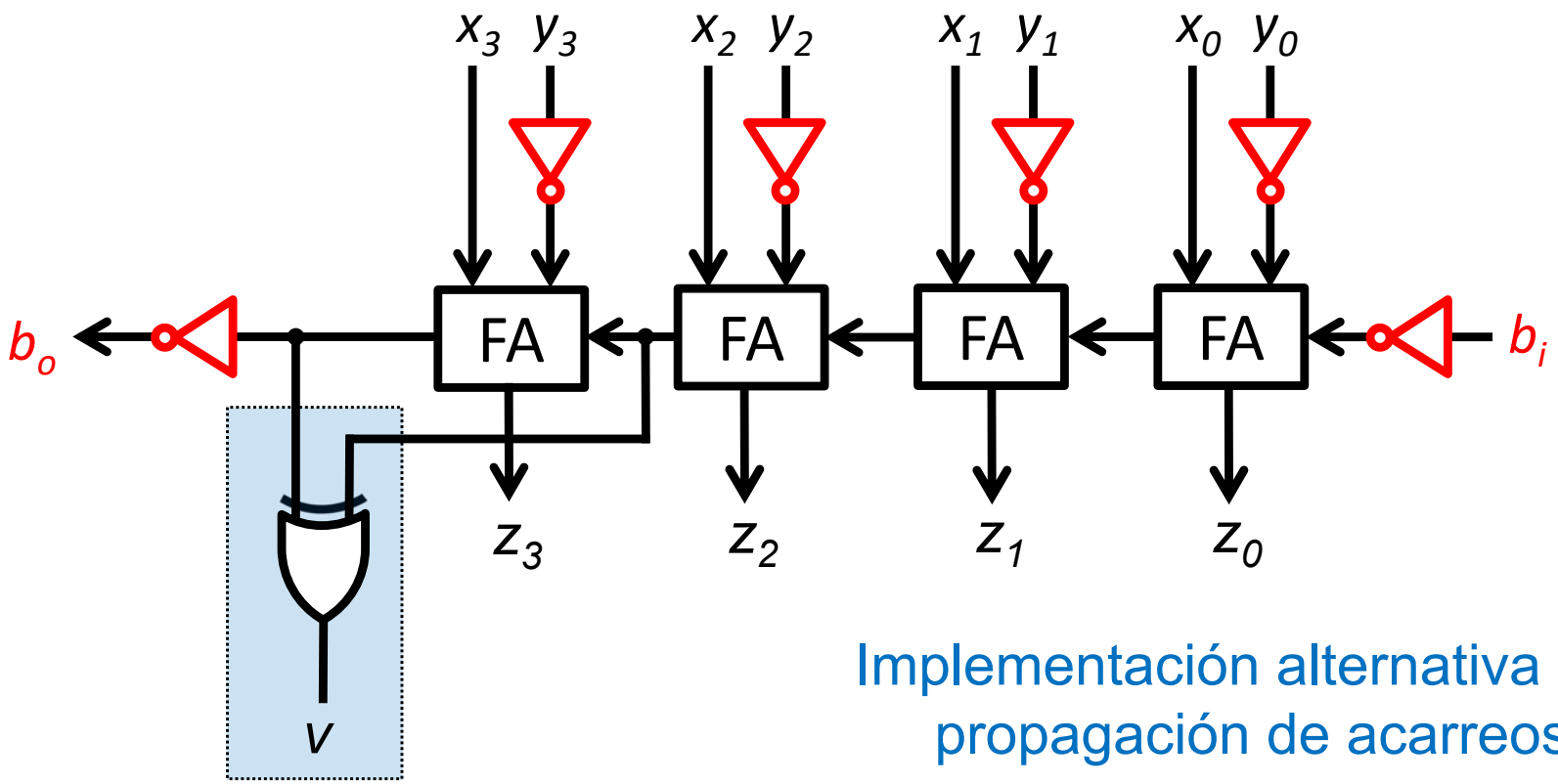
- Un restador también puede diseñarse usando un sumador



Restador

- Un restador también puede diseñarse usando un sumador

$$z = x - y - b_i = x + (-y) - b_i =_{C2} x + C2(y) - b_i = x + C1(y) + 1 - b_i = x + \overline{y} + \overline{b_i}$$



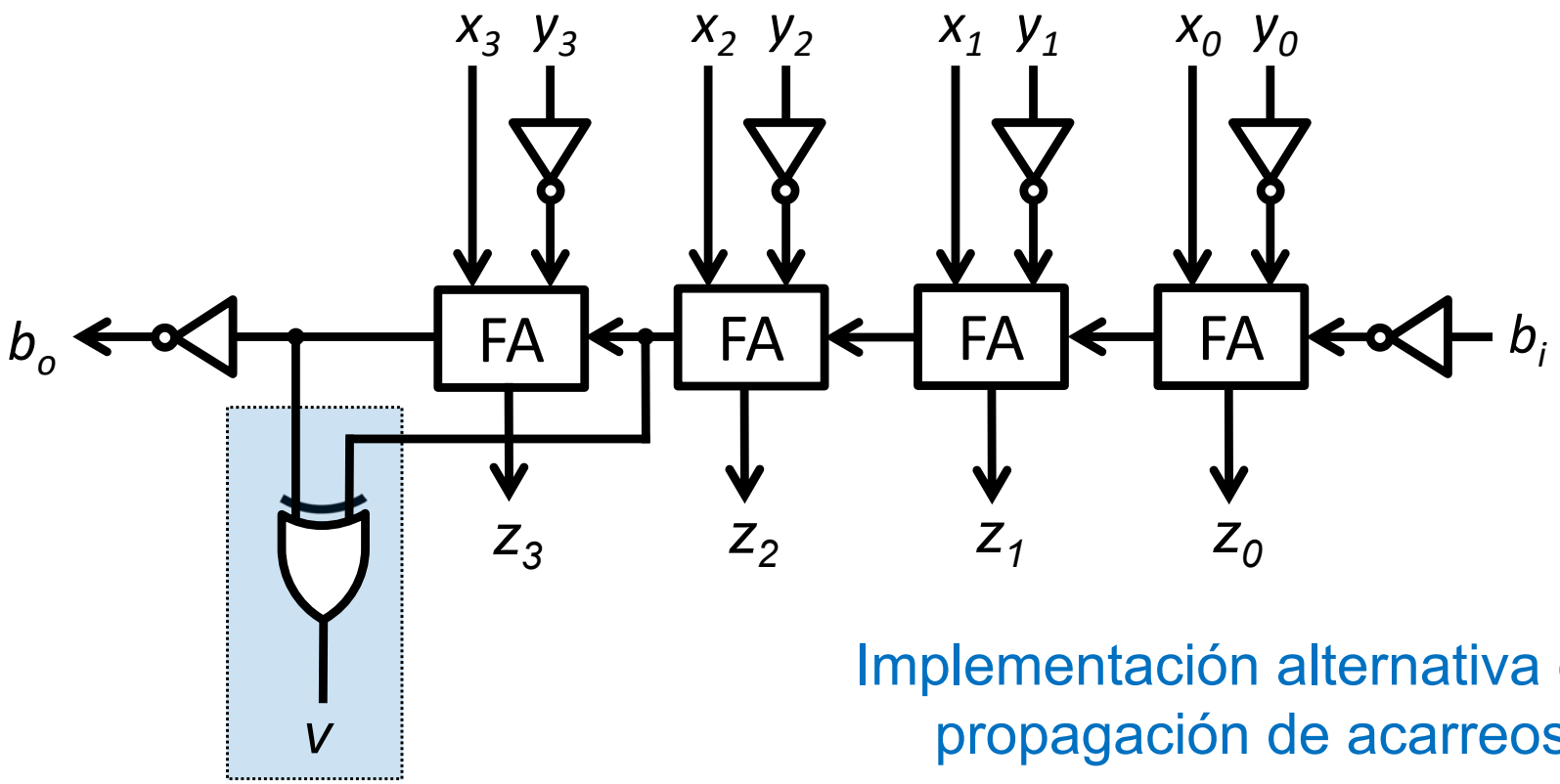
Implementación alternativa con propagación de acarreos

Restador de 4 bits

Restador

- Un restador también puede diseñarse usando un sumador

$$\underline{z} = \underline{x} - \underline{y} - b_i = \underline{x} + (-\underline{y}) - b_i =_{C2} \underline{x} + C2(\underline{y}) - b_i = \underline{x} + C1(\underline{y}) + 1 - b_i = \underline{x} + \overline{\underline{y}} + \overline{b_i}$$

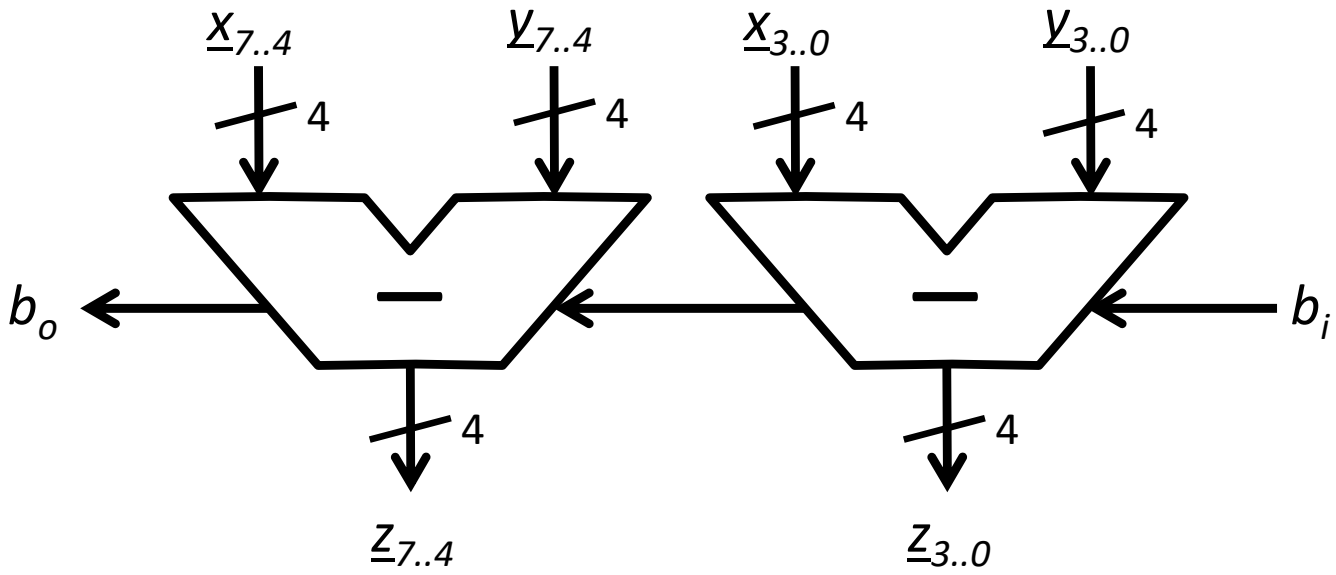


Implementación alternativa con propagación de acarreos

Restador de 4 bits

Restador

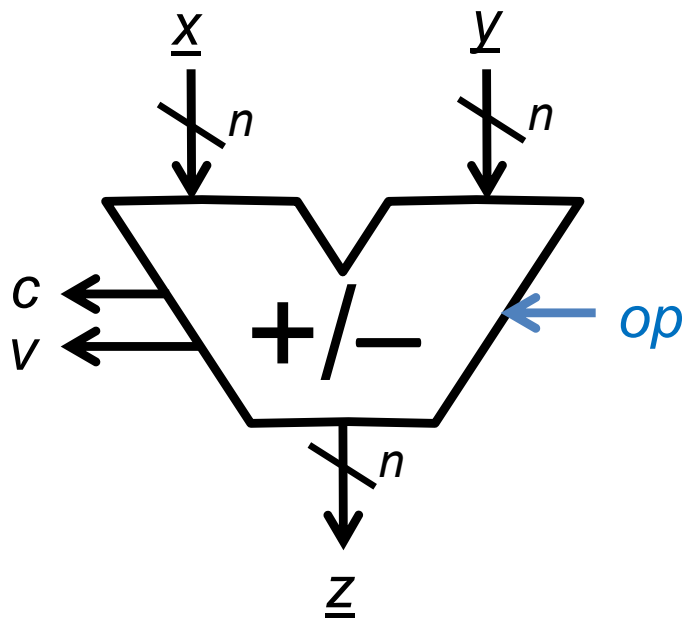
- Varios restadores se pueden componer en serie para para comportarse como un restador de mayor anchura.



Implementación serie

Restador de 8 bits

Sumador/restador



a, b	2 entradas de datos de n bits
op	1 entrada de selección de operación
z	1 salida de datos de n bits
c	1 salida de acarreo (carry)
v	1 salida de overflow (opcional)

realiza la suma/resta de x e y

$z = (x + y) \bmod 2^n$ *op = 0*

$$c = \begin{cases} 1 & \text{si } (x + y) \geq 2^n \\ 0 & \text{en otro caso} \end{cases}$$

$$v = \begin{cases} 1 & \text{si } [x_{n-1}=y_{n-1}=0 \text{ y } z_{n-1}=1] \text{ ó } [x_{n-1}=y_{n-1}=1 \text{ y } z_{n-1}=0] \\ 0 & \text{en otro caso} \end{cases}$$

$z = (x - y) \bmod 2^n$ *op = 1*

$$c = \begin{cases} 0 & \text{si } (x - y) < 0 \\ 1 & \text{en otro caso} \end{cases}$$

$$v = \begin{cases} 1 & \text{si } [z_{n-1}=y_{n-1}=0 \text{ y } x_{n-1}=1] \text{ ó } [z_{n-1}=y_{n-1}=1 \text{ y } x_{n-1}=0] \\ 0 & \text{en otro caso} \end{cases}$$

Sumador/restador

si op=0 $\underline{z} = \underline{x} + \underline{y} = \underline{x} + \underline{y} + 0$

Sumador/restador

si op=0 $\quad \underline{z} = \underline{x} + \underline{y} = \underline{x} + \underline{y} + 0$

si op=1 $\quad \underline{z} = \underline{x} - \underline{y} = \underline{x} + (-\underline{y}) =_{C2}$

$=_{C2} \underline{x} + C2(\underline{y}) = \underline{x} + C1(\underline{y}) + 1 = \underline{x} + \overline{\underline{y}} + 1$

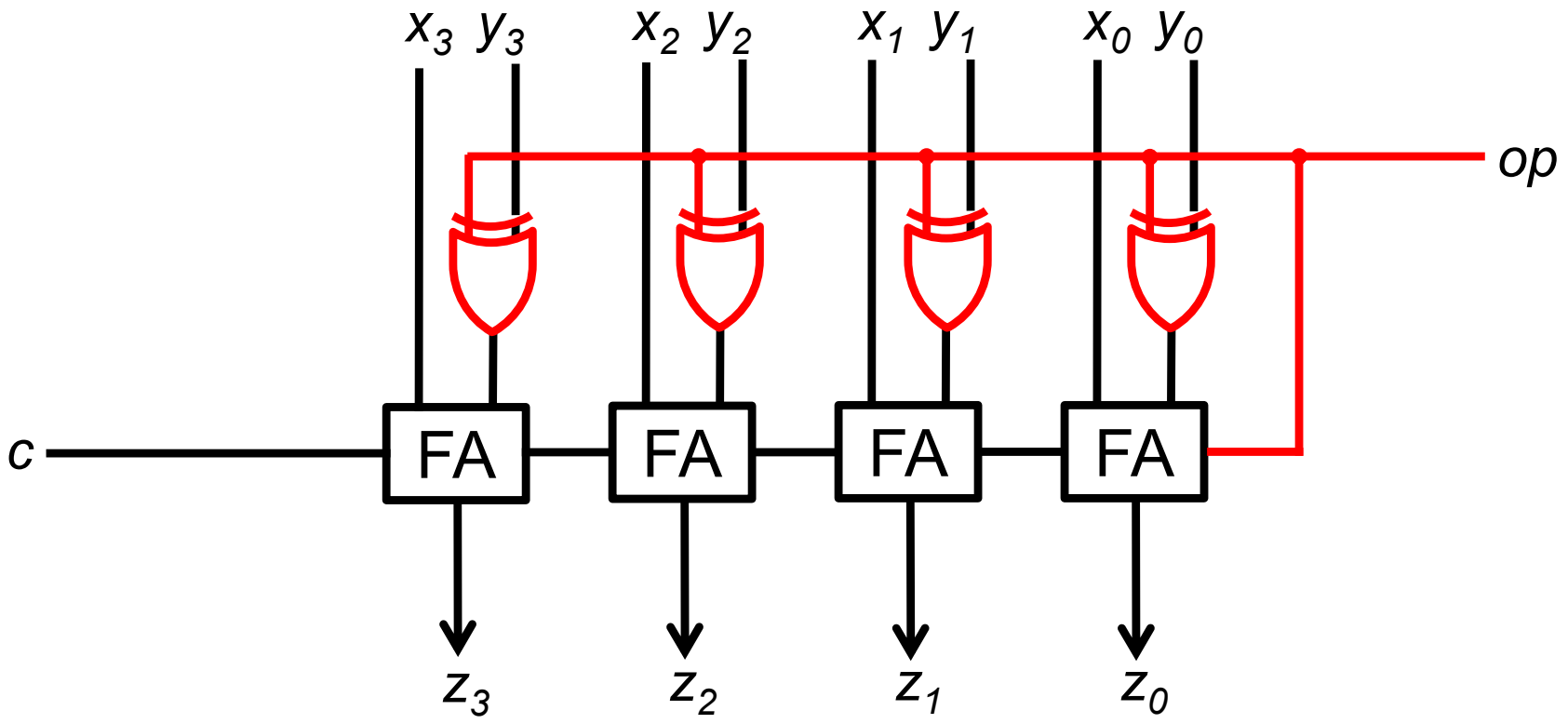
Sumador/restador

si op=0 $\quad \underline{z} = \underline{x} + \underline{y} = \underline{x} + \underline{y} + 0$

si op=1 $\quad \underline{z} = \underline{x} - \underline{y} = \underline{x} + (-\underline{y}) =_{C2}$

$=_{C2} \underline{x} + C2(\underline{y}) = \underline{x} + C1(\underline{y}) + 1 = \underline{x} + \overline{\underline{y}} + 1$

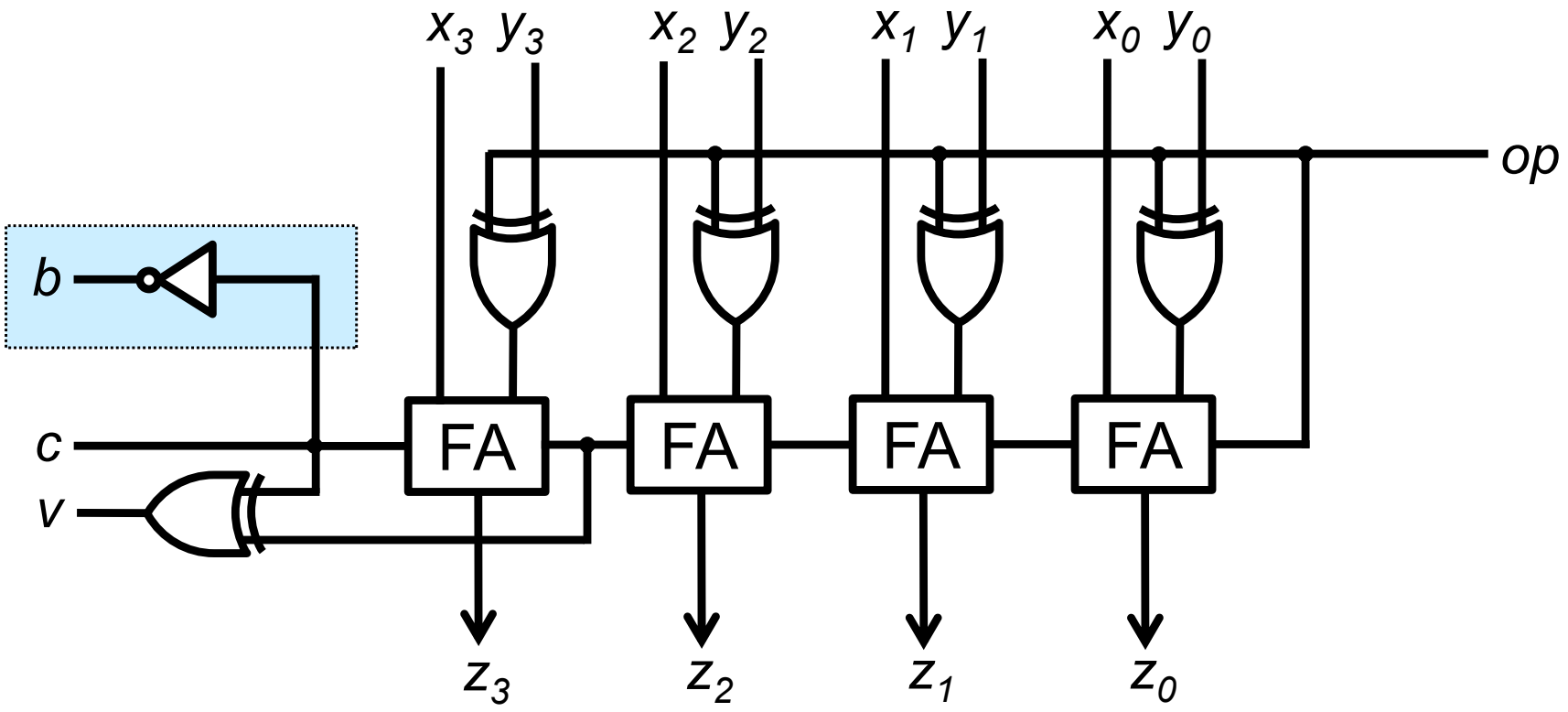
$\underline{z} = \underline{x} + (\underline{y} \oplus op) + op$



Sumador/restador

Implementación con propagación de acarreos

Sumador/restador de 4 bits



Sumador/restador

- Pueden añadirse puertos comunes para carry/borrow

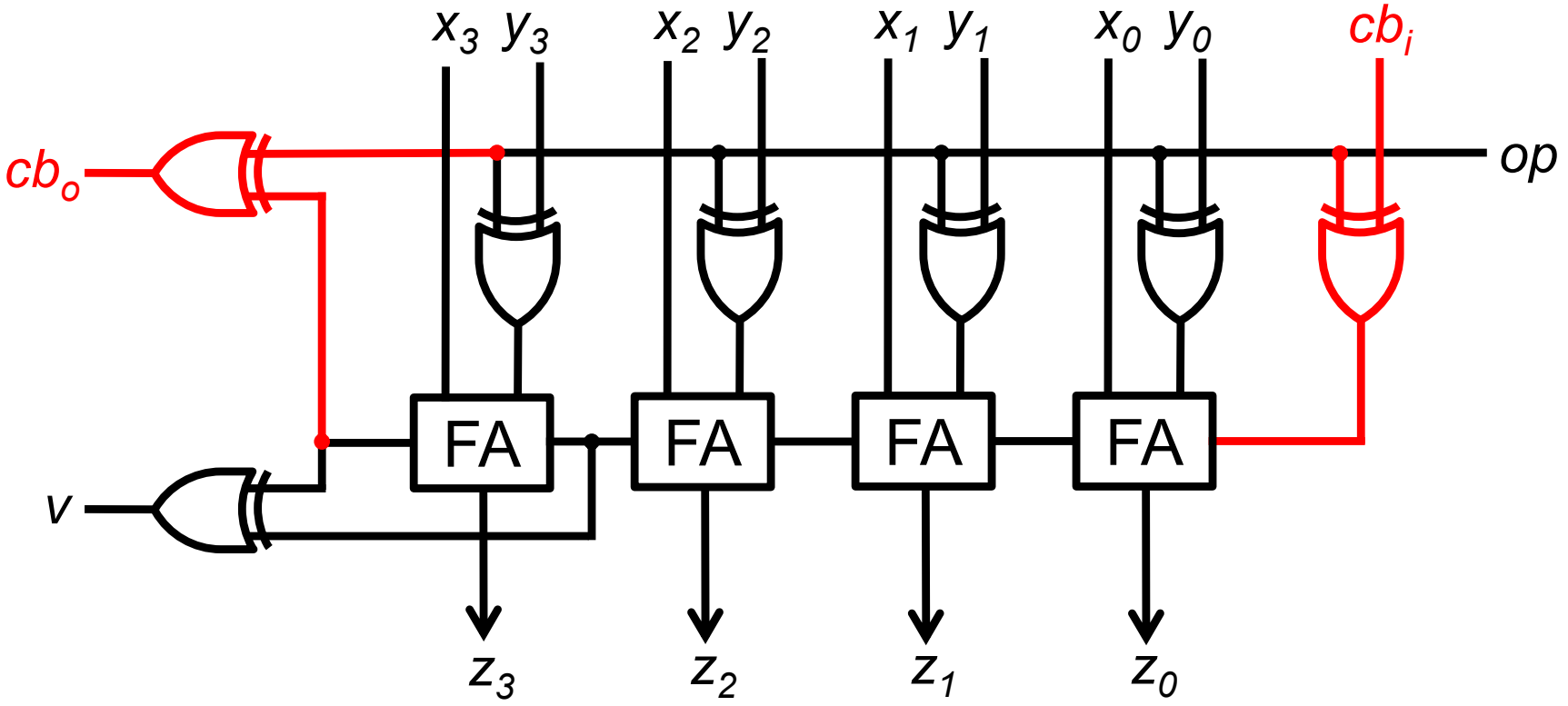
si op=0 $\quad \underline{z} = \underline{x} + \underline{y} + cb_i$

si op=1 $\quad \underline{z} = \underline{x} - \underline{y} - cb_i = \underline{x} + (-\underline{y}) - cb_i =_{C2}$

$=_{C2} \underline{x} + C2(\underline{y}) - cb_i = \underline{x} + C1(\underline{y}) + 1 - cb_i$

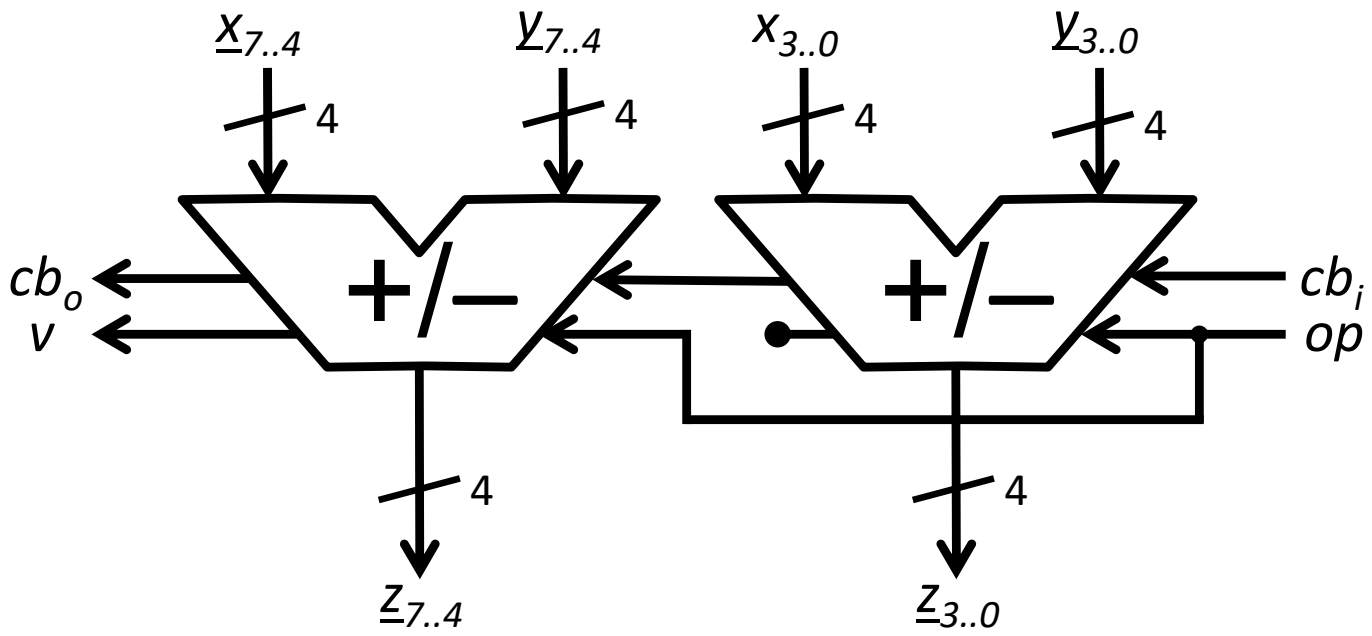
$= \underline{x} + \underline{y} + 1 - cb_i = \underline{x} + \overline{\underline{y}} + \overline{cb_i}$

$\underline{z} = \underline{x} + (\underline{y} \oplus op) + (cb_i \oplus op)$



Sumador/restador

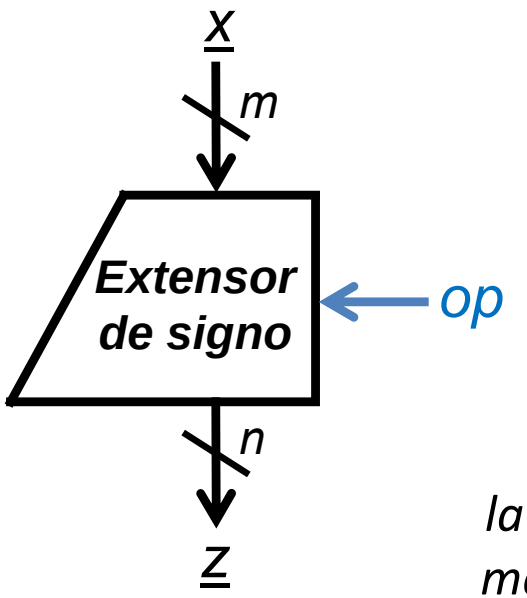
- Varios sumadores/restadores se pueden componer en serie para para comportarse como un sumador/restador de mayor anchura.



Implementación serie

Sumador/restador de 8 bits

Extensor de signo



x	1 entrada de datos de m bits
op	1 entrada de selección de operación
z	1 salida de datos n bits

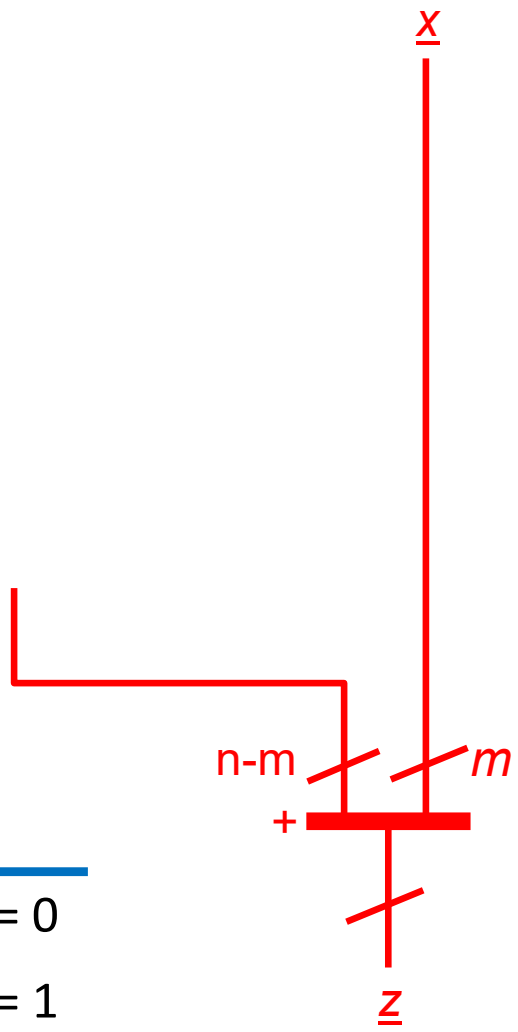
la salida representa en la misma codificación pero con mayor número de bits el número que está a la entrada

$$z = \begin{cases} zExt(x) \equiv (0...0)\ \&\ x & \text{si } op = 0 \\ sExt(x) \equiv (x_{m-1}...\ x_{m-1})\ \&\ x & \text{si } op = 1 \end{cases}$$

aplica si x codifica un número sin signo

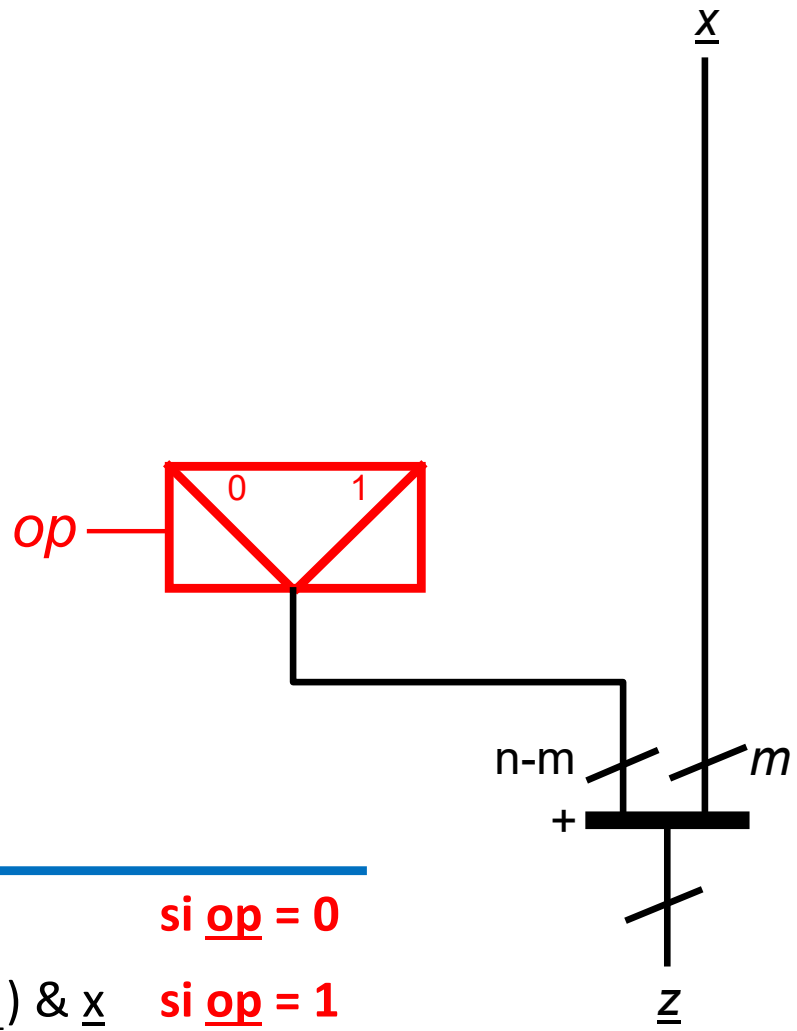
aplica si x codifica un número con signo en C2

Extensor de signo



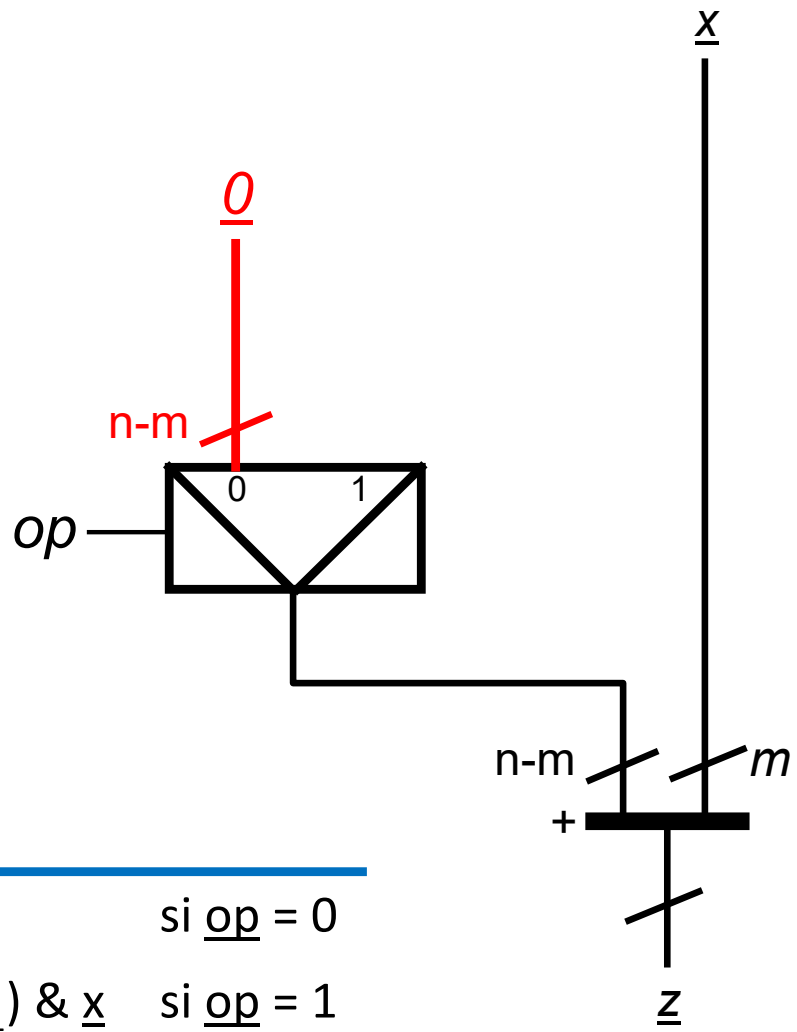
$$\underline{z} = \begin{cases} zExt(\underline{x}) \equiv (0...0)\ \&\ \underline{x} & \text{si } \underline{op} = 0 \\ sExt(\underline{x}) \equiv (x_{m-1}...\ x_{m-1})\ \&\ \underline{x} & \text{si } \underline{op} = 1 \end{cases}$$

Extensor de signo



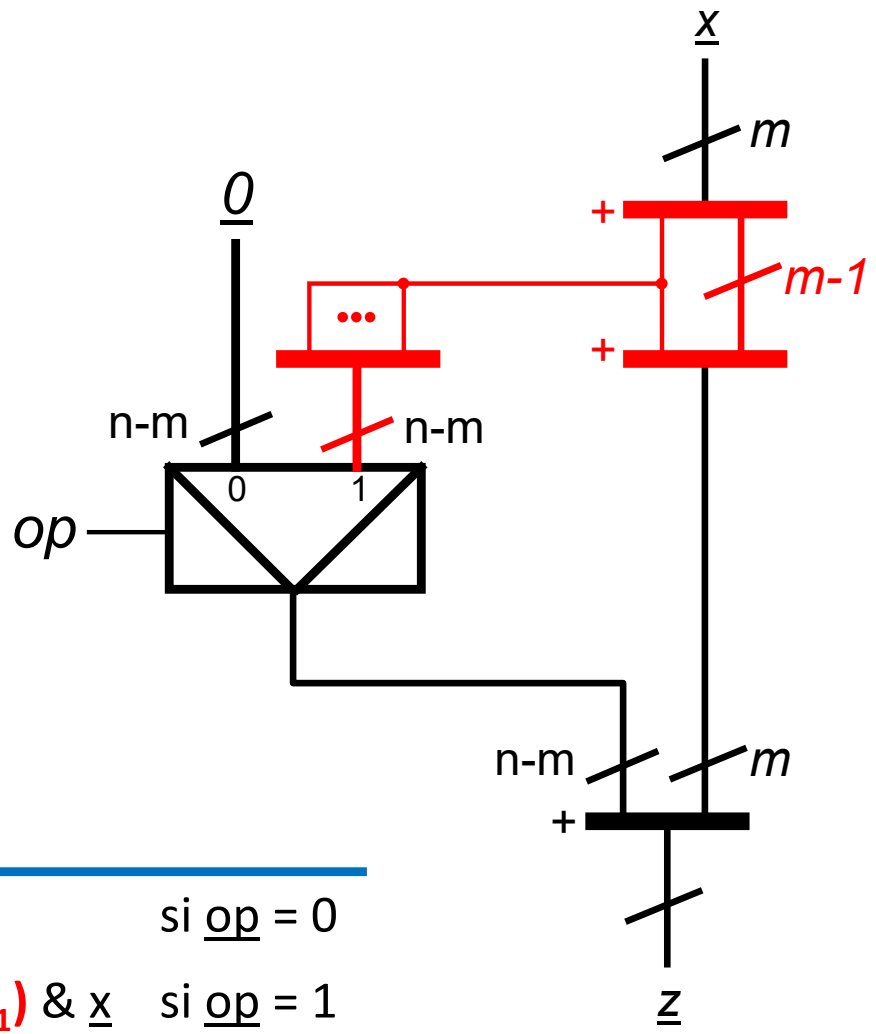
$$\underline{z} = \begin{cases} zExt(\underline{x}) \equiv (0...0)\ \&\ \underline{x} & \text{si } \underline{op} = 0 \\ sExt(\underline{x}) \equiv (x_{m-1}...\ x_{m-1})\ \&\ \underline{x} & \text{si } \underline{op} = 1 \end{cases}$$

Extensor de signo



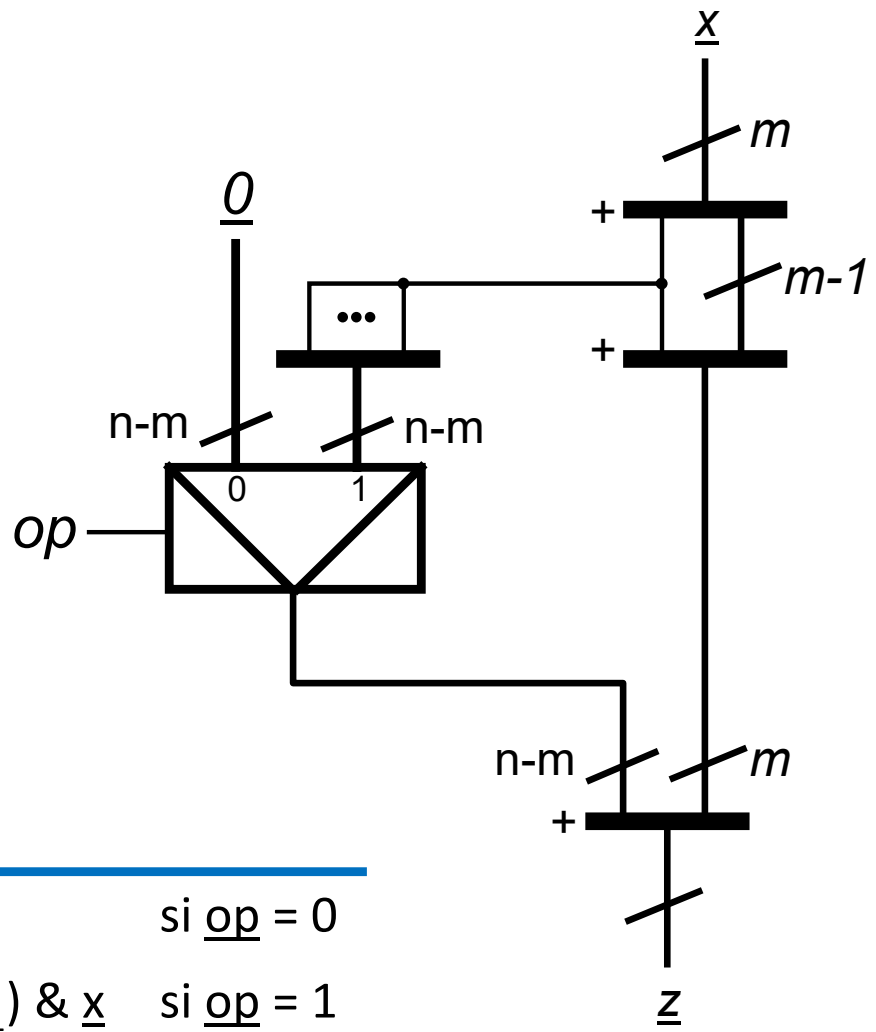
$$\underline{z} = \begin{cases} \text{zExt}(\underline{x}) \equiv \mathbf{(0\ldots0)} \;\&\; \underline{x} & \text{si } \underline{op} = 0 \\ \text{sExt}(\underline{x}) \equiv (x_{m-1} \ldots x_{m-1}) \;\&\; \underline{x} & \text{si } \underline{op} = 1 \end{cases}$$

Extensor de signo



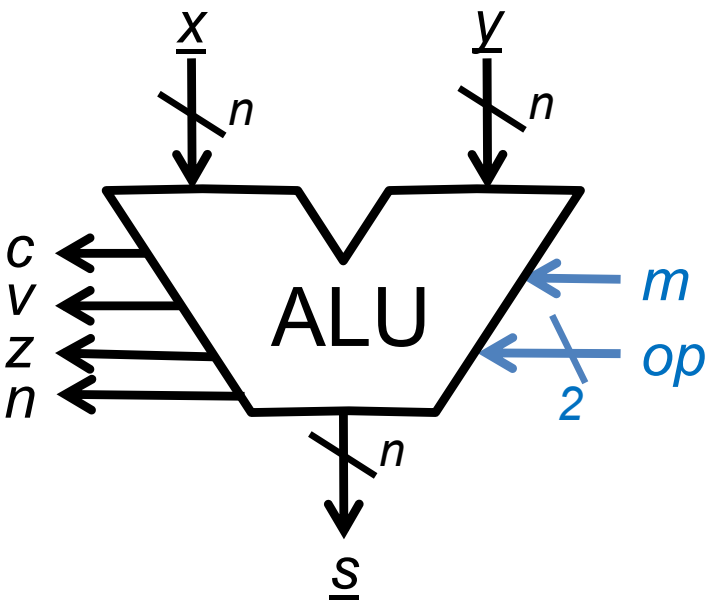
$$\underline{z} = \begin{cases} \text{zExt}(\underline{x}) \equiv (0\ldots0)\ \&\ \underline{x} & \text{si } \underline{op} = 0 \\ \text{sExt}(\underline{x}) \equiv (x_{m-1}\ldots x_{m-1})\ \&\ \underline{x} & \text{si } \underline{op} = 1 \end{cases}$$

Extensor de signo



$$\underline{z} = \begin{cases} zExt(\underline{x}) \equiv (0...0)\ \&\ \underline{x} & \text{si } \underline{op} = 0 \\ sExt(\underline{x}) \equiv (x_{m-1}...\ x_{m-1})\ \&\ \underline{x} & \text{si } \underline{op} = 1 \end{cases}$$

ALU (Arithmetic-Logic Unit)



x, y	2 entradas de datos de n bits
m	1 entrada de selección de modo
op	1 entrada de selección de operación
s	1 salida de datos de n bits
c	1 salida de acarreo (carry)
v	1 salida de overflow
z	1 salida de detección de cero
n	1 salida de detección de negativo

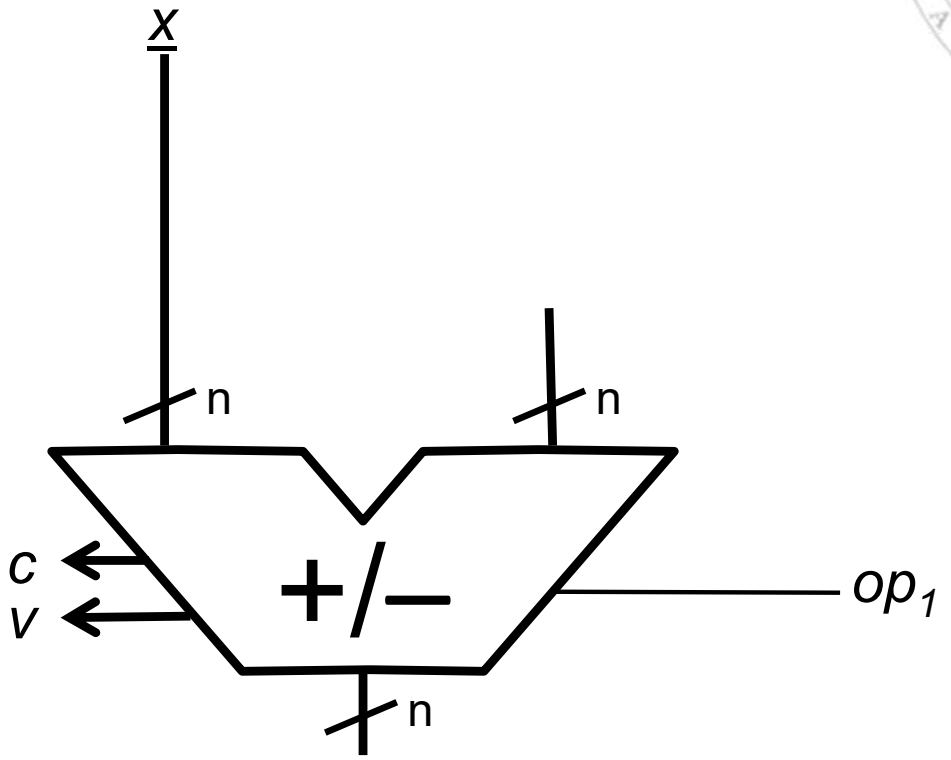
operaciones lógicas

m	op_1	op_0	s
0	0	0	not(x)
0	0	1	and(x, y)
0	1	0	x
0	1	1	or(x, y)

operaciones aritméticas

m	op_1	op_0	s	
1	0	0	x + y	
1	0	1	x – 1	= x + (-1) $=_{C2}$ x + 1
1	1	0	x – y	
1	1	1	x + 1	= x – (-1) $=_{C2}$ x – 1

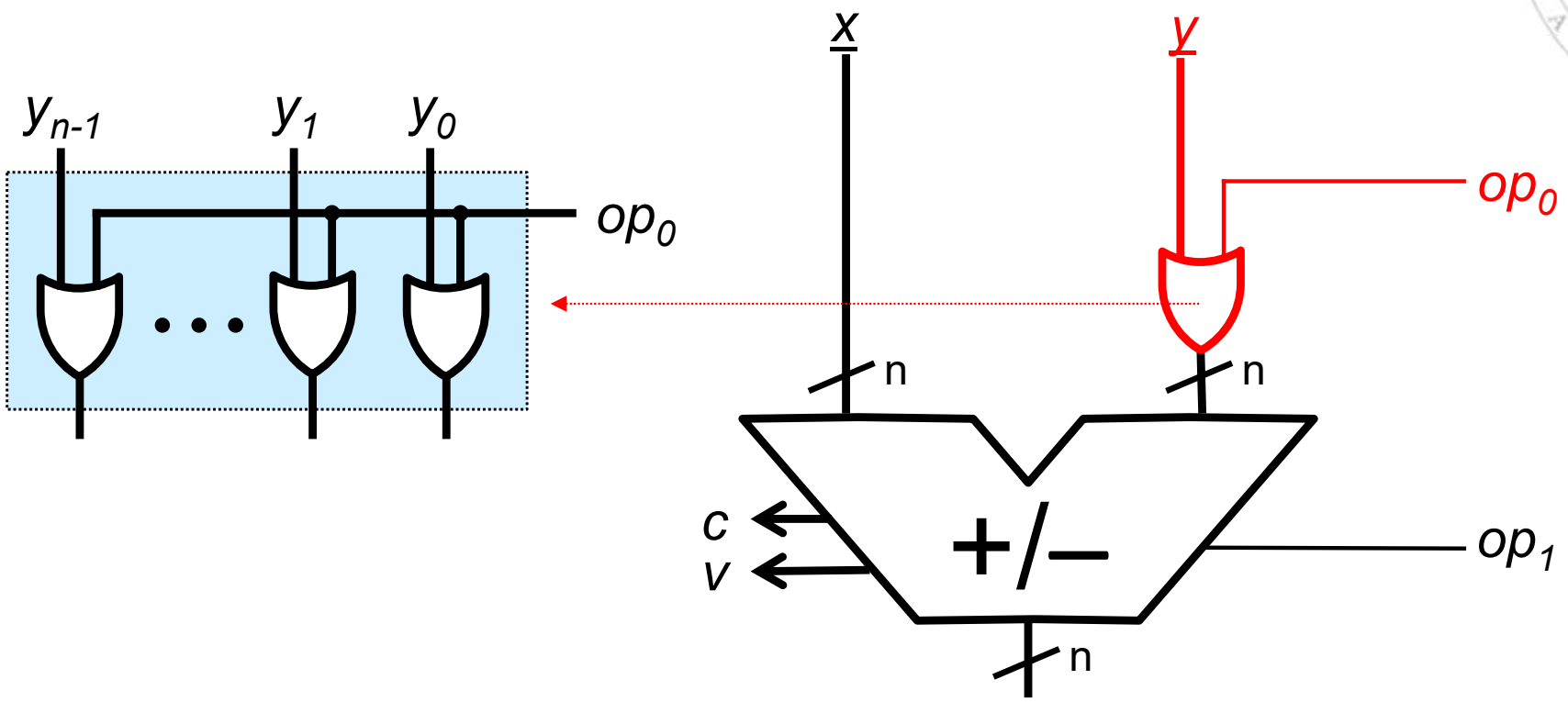
ALU (Arithmetic-Logic Unit)



operaciones aritméticas

m	op_1	op_0	s	
1	0	0	x + y	
1	0	1	x − 1	= x + (-1) $=_{C2}$ x + 1
1	1	0	x − y	
1	1	1	x + 1	= x − (-1) $=_{C2}$ x − 1

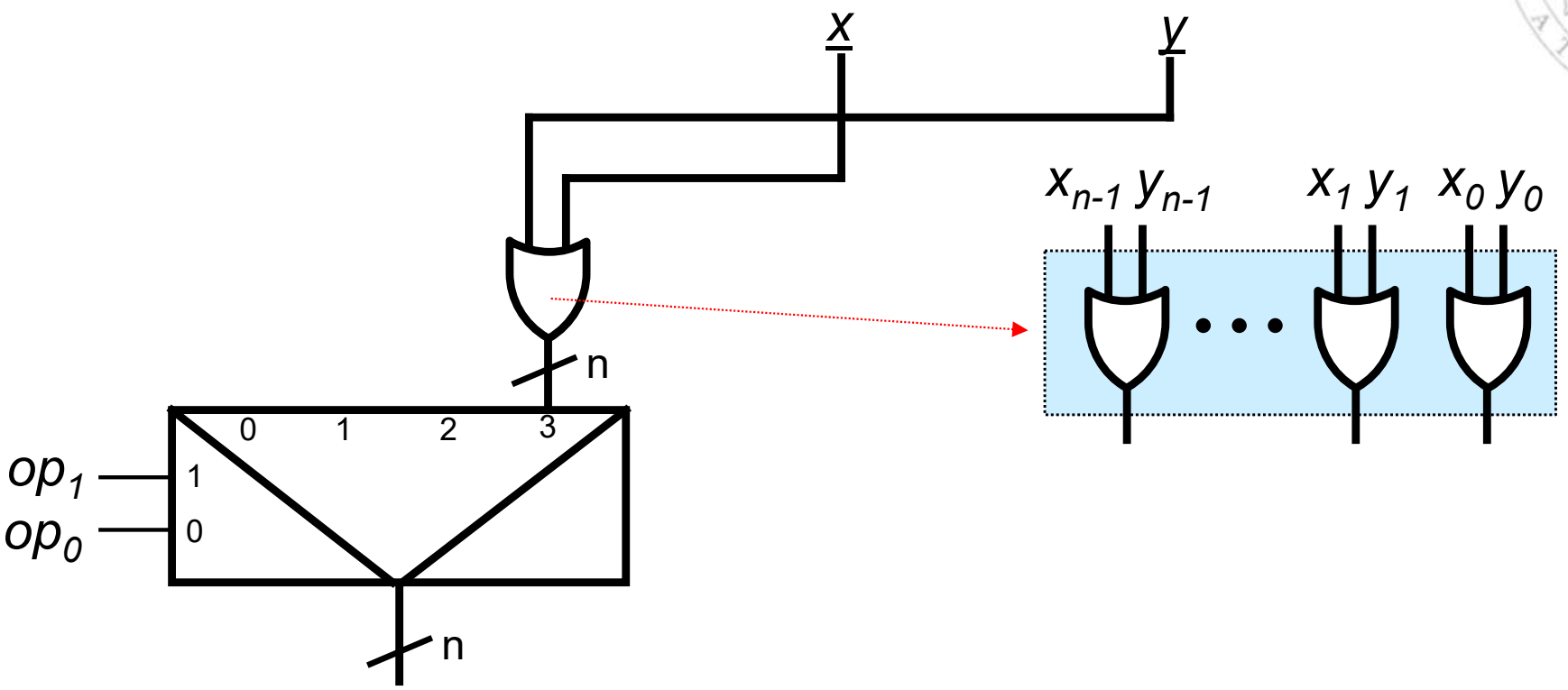
ALU (Arithmetic-Logic Unit)



operaciones aritméticas

m	op_1	op_0	s	
1	0	0	x + y	
1	0	1	x − 1	= x + (-1) $=_{C2}$ x + 1
1	1	0	x − y	
1	1	1	x + 1	= x − (-1) $=_{C2}$ x − 1

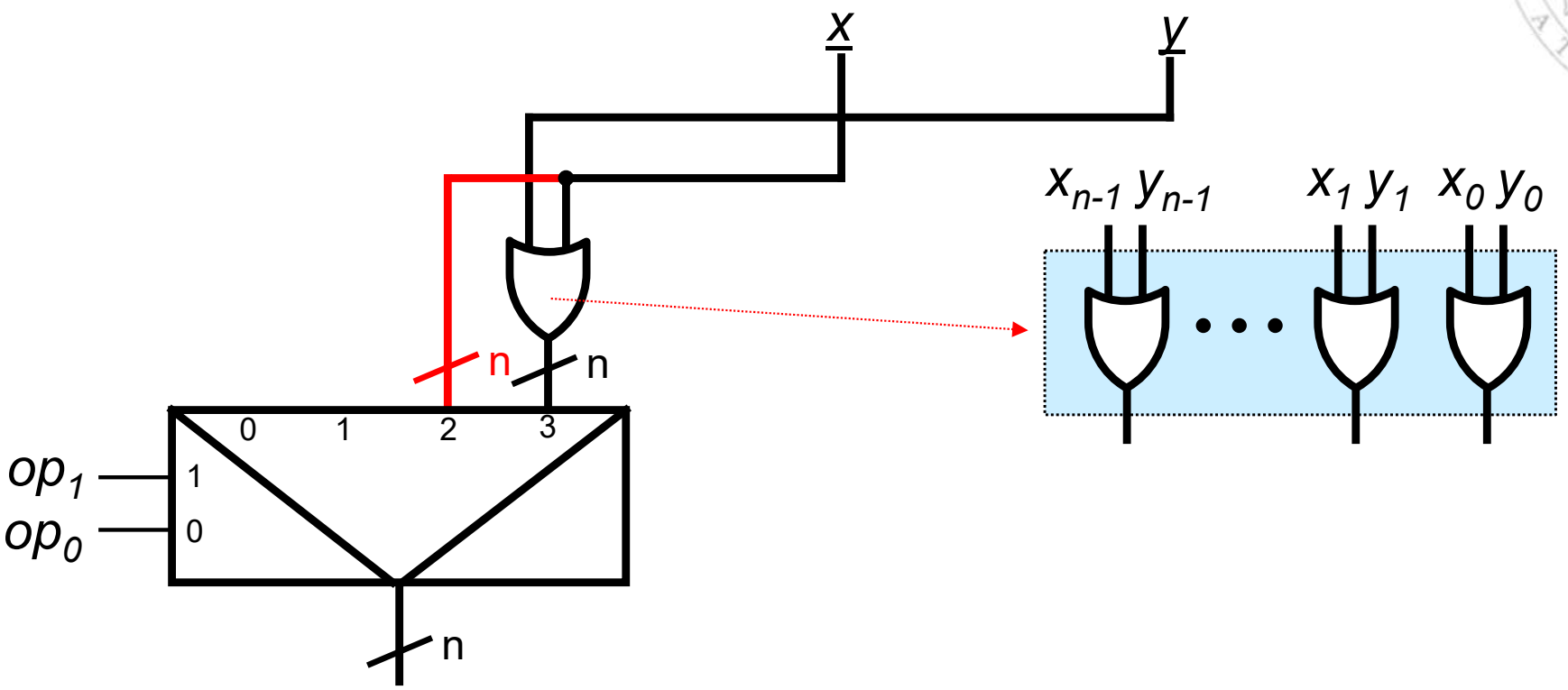
ALU (Arithmetic-Logic Unit)



operaciones lógicas

m	op_1	op_0	s
0	0	0	not(x)
0	0	1	and(x, y)
0	1	0	x
0	1	1	or(x, y)

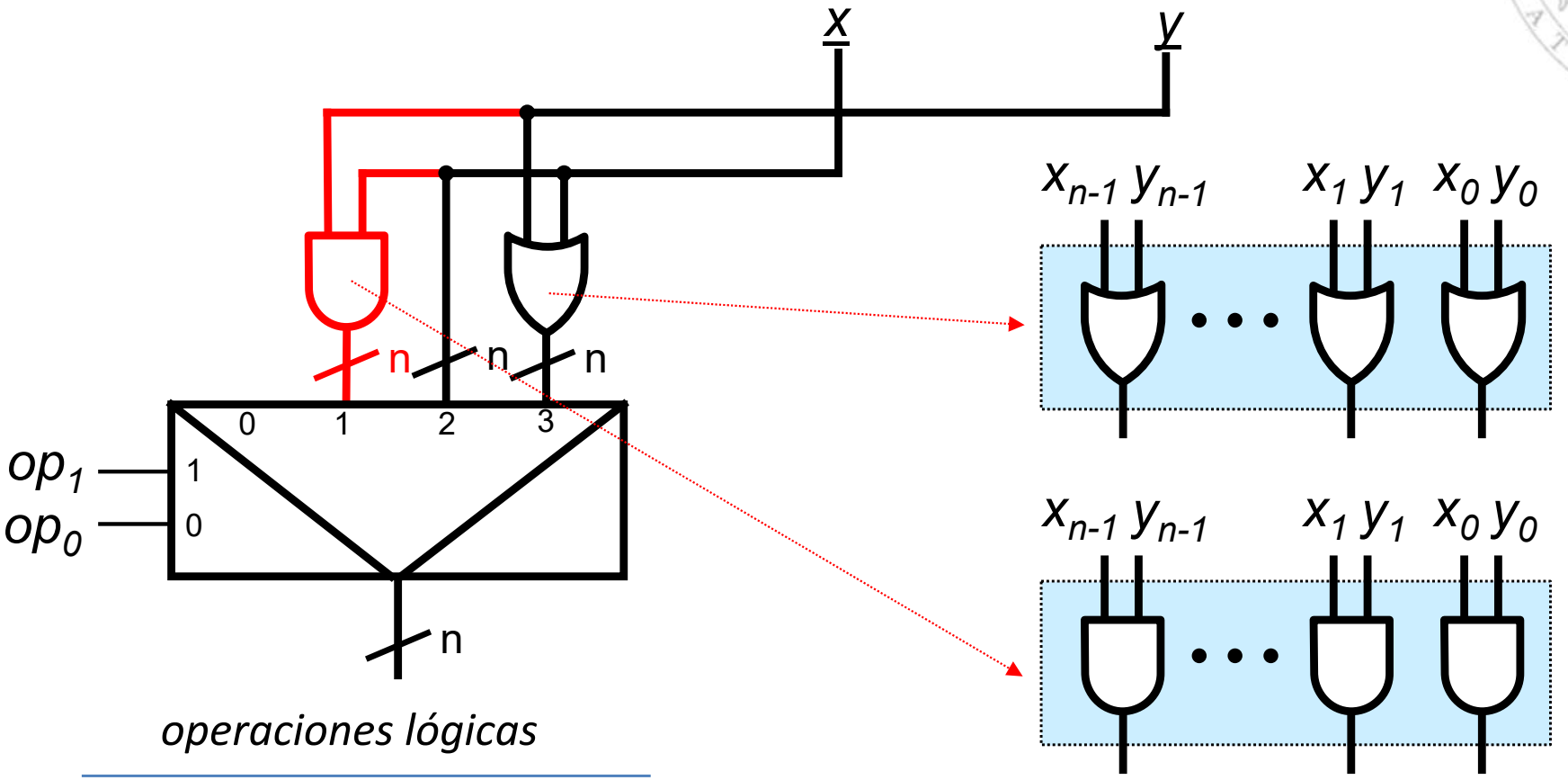
ALU (Arithmetic-Logic Unit)



operaciones lógicas

m	op_1	op_0	s
0	0	0	not(x)
0	0	1	and(x, y)
0	1	0	x
0	1	1	or(x, y)

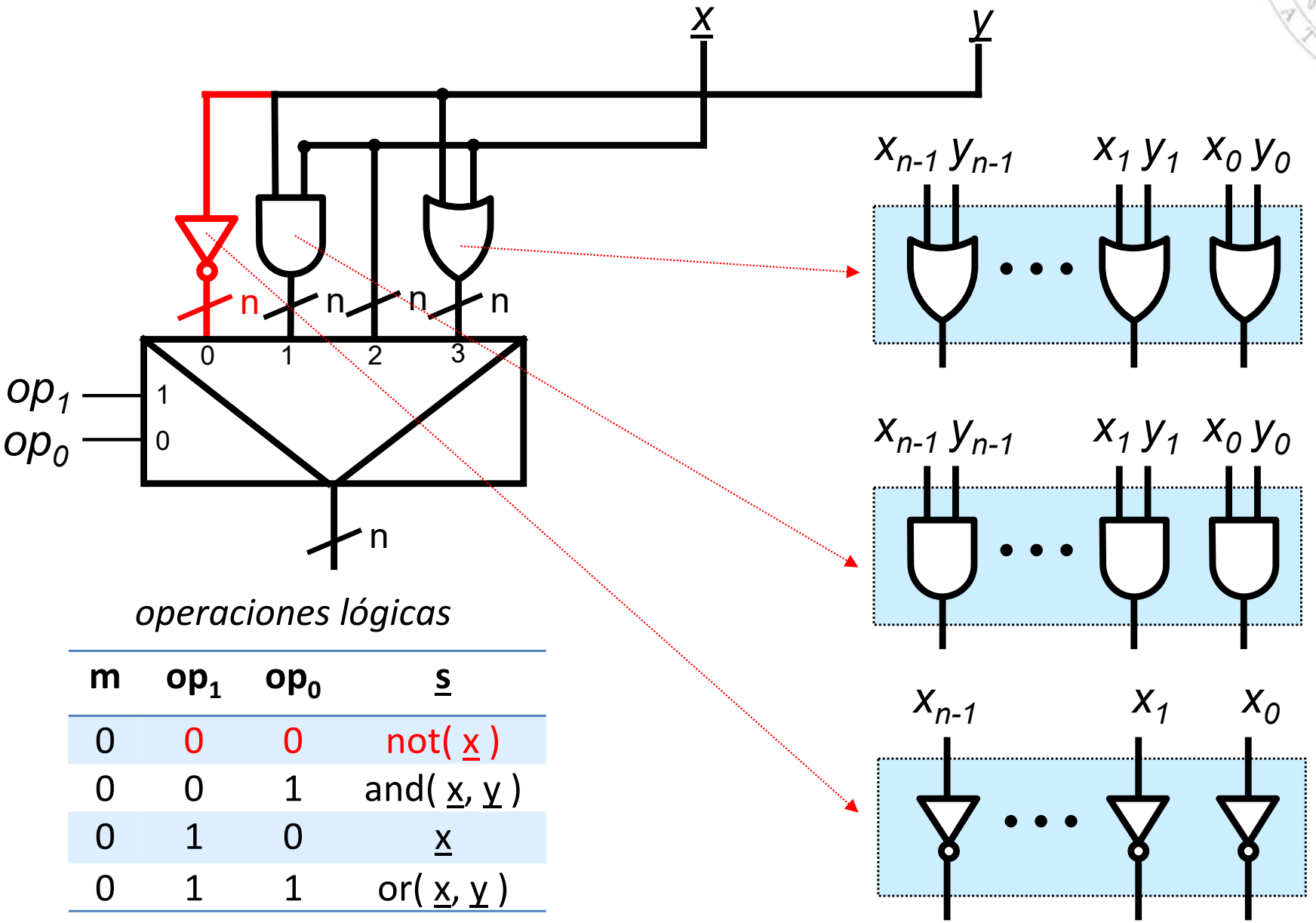
ALU (Arithmetic-Logic Unit)



operaciones lógicas

m	op_1	op_0	s
0	0	0	not(x)
0	0	1	and(x, y)
0	1	0	x
0	1	1	or(x, y)

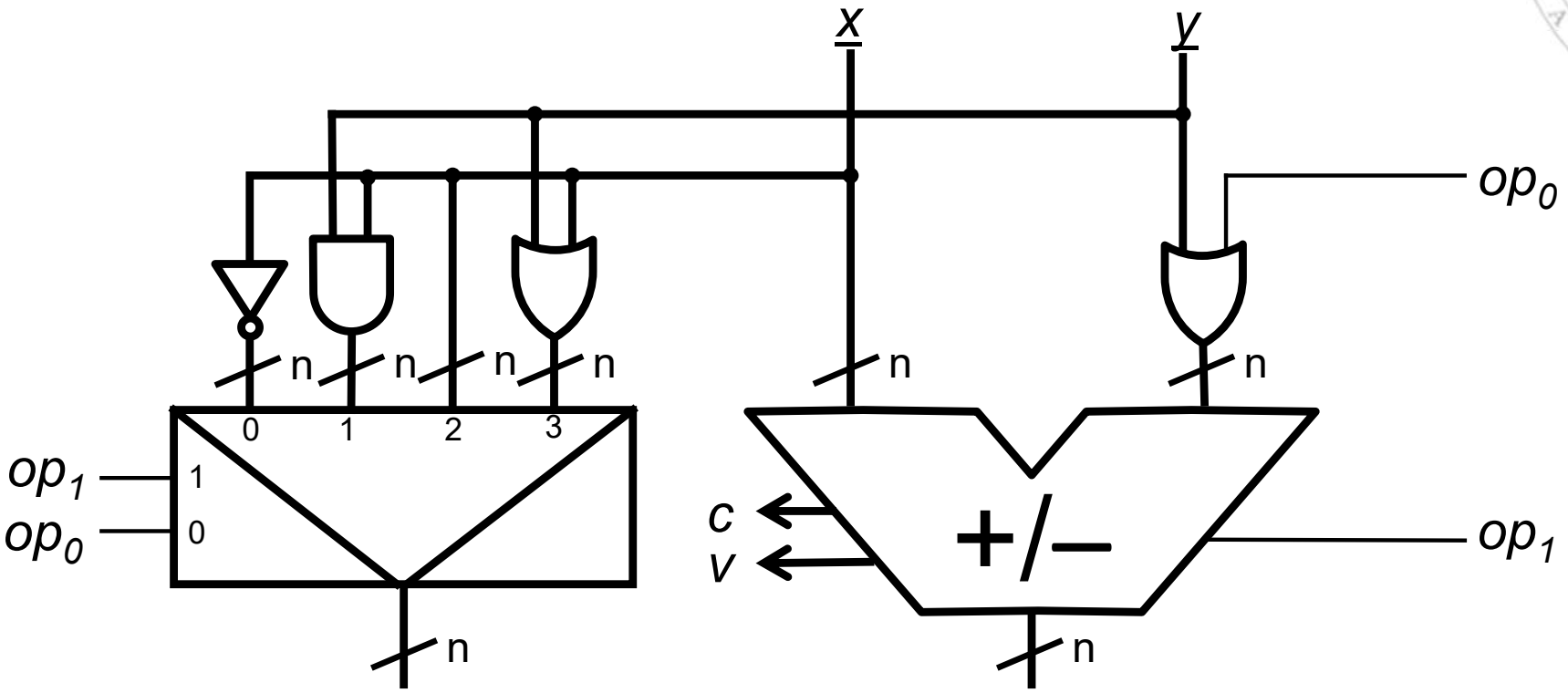
ALU (Arithmetic-Logic Unit)



operaciones lógicas

m	op_1	op_0	s
0	0	0	not(x)
0	0	1	and(x, y)
0	1	0	x
0	1	1	or(x, y)

ALU (Arithmetic-Logic Unit)



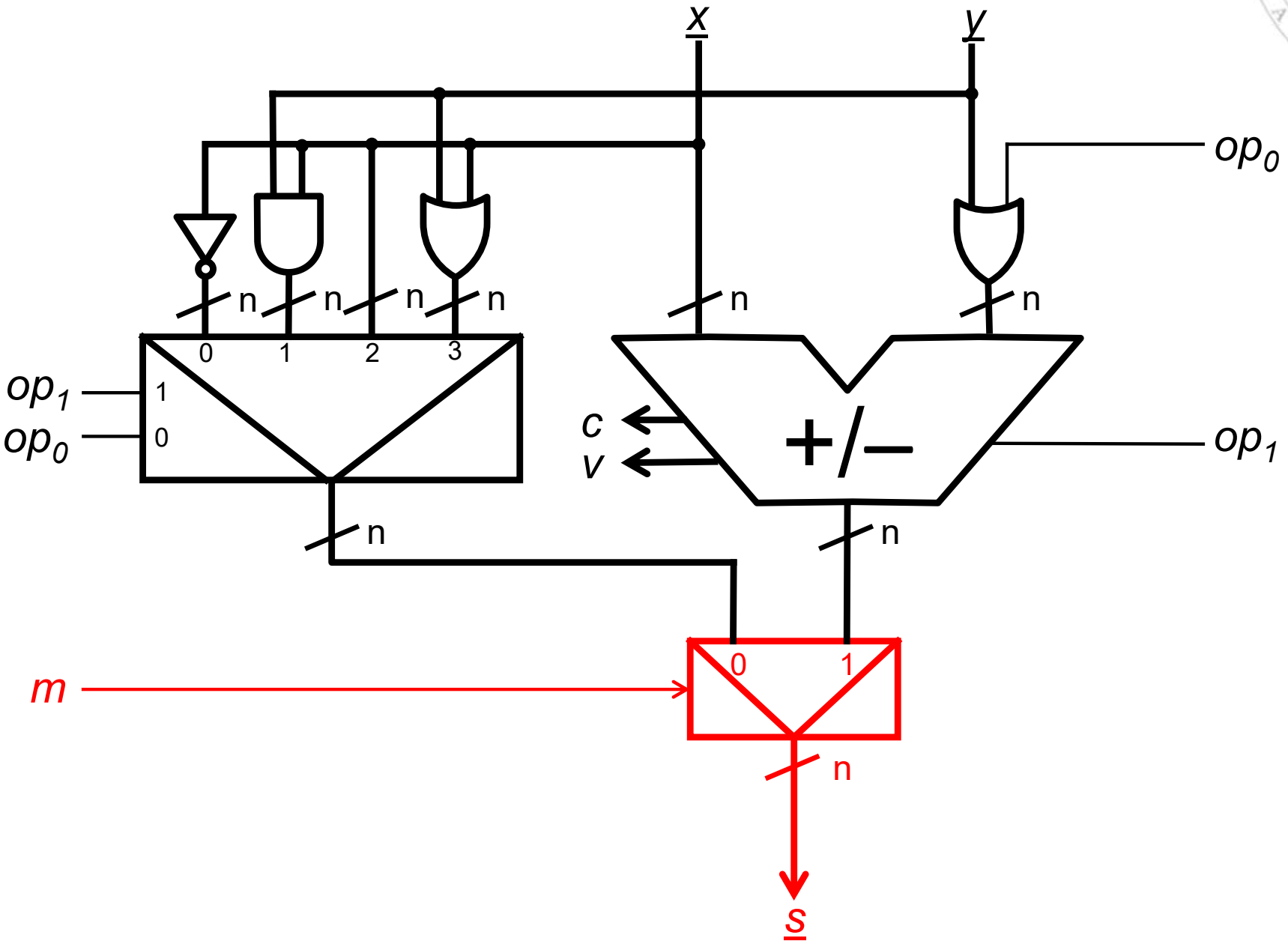
operaciones lógicas

m	op_1	op_0	s
0	0	0	not(a)
0	0	1	and(a, b)
0	1	0	a
0	1	1	or(a, b)

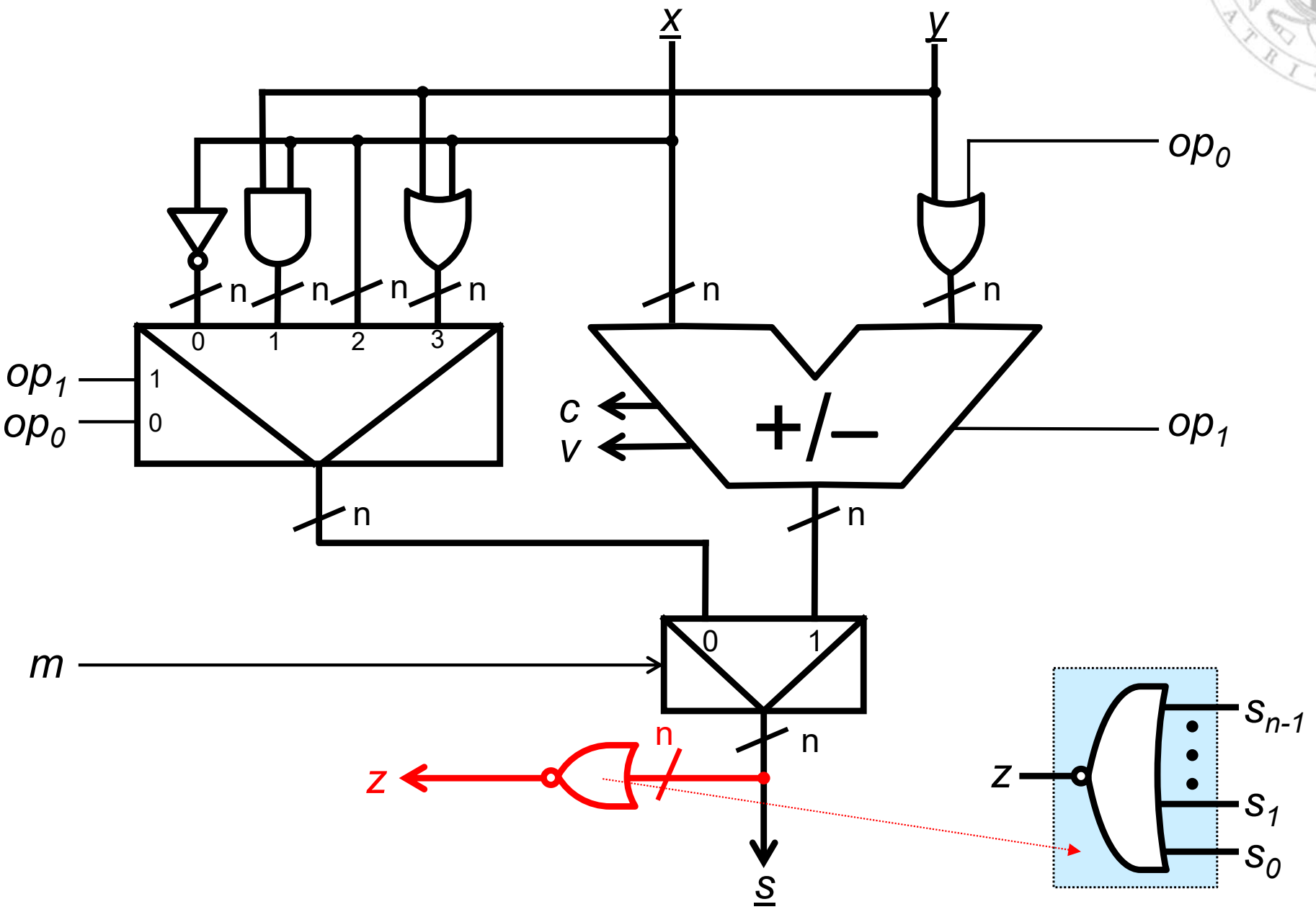
operaciones aritméticas

m	op_1	op_0	s	
1	0	0	a + b	
1	0	1	a – 1	= a + (-1) $=_{C2}$ a + 1
1	1	0	a – b	
1	1	1	a + 1	= a – (-1) $=_{C2}$ a – 1

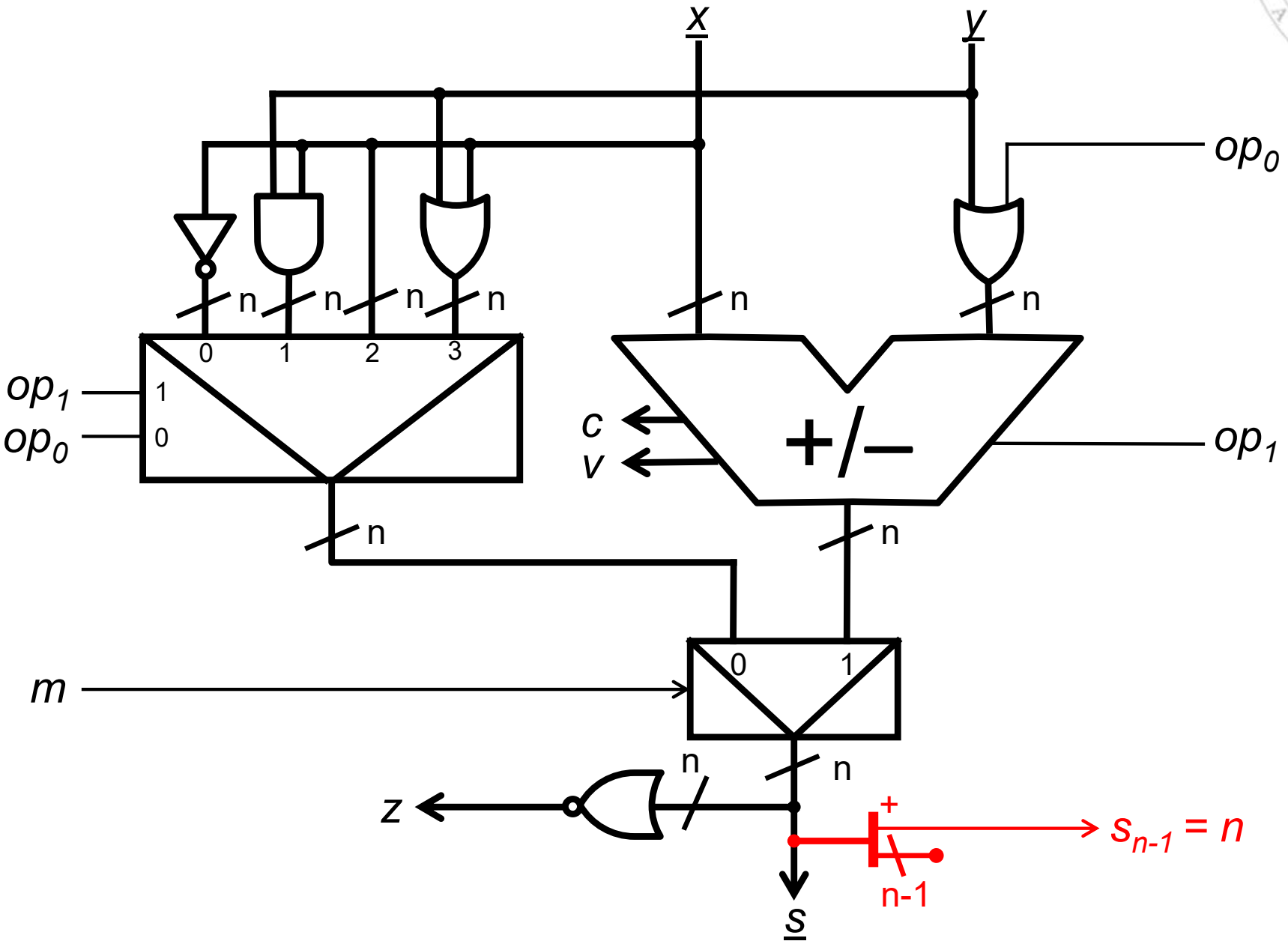
ALU (Arithmetic-Logic Unit)



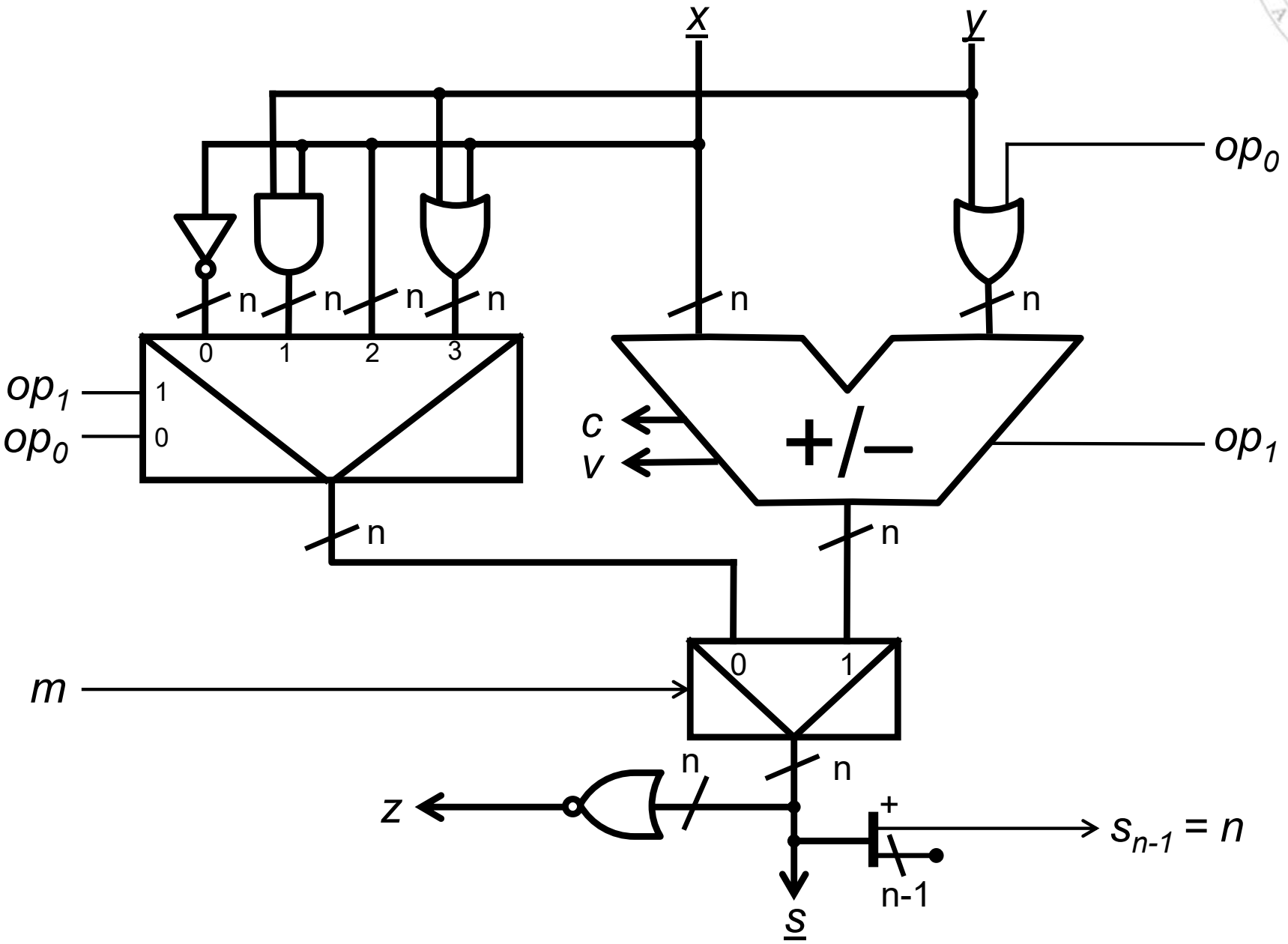
ALU (Arithmetic-Logic Unit)



ALU (Arithmetic-Logic Unit)



ALU (Arithmetic-Logic Unit)



ALU (Arithmetic-Logic Unit)

- La ALU puede usarse para comparar números
  - Para ello no es necesario añadir un comparador, basta con restar los operandos y comprobar el estado de los flags.
  - Por ejemplo, al restar x – y el flag Z se activa solo cuando x = y

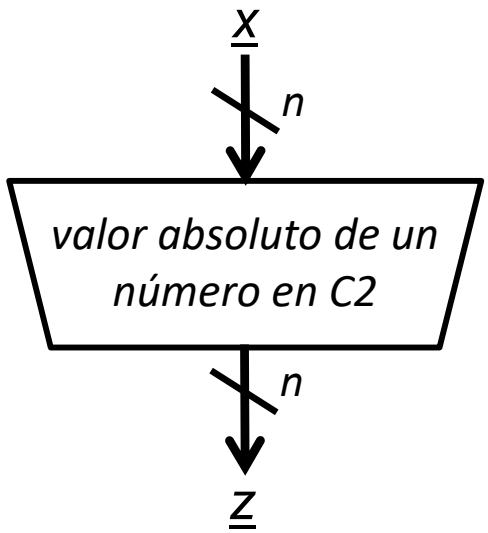
	flags a comprobar tras realizar x – y	
comparación	operandos sin signo	operandos con signo (en C2)
x = y	Z	Z
x ≠ y	$\overline{Z}$	$\overline{Z}$
x < y	$\overline{C}$	$N \oplus V$
x ≤ y	$Z + \overline{C}$	$Z + (N \oplus V)$
x > y	$\overline{Z} \cdot C$	$\overline{Z} \cdot \overline{(N \oplus V)}$
x ≥ y	C	$\overline{(N \oplus V)}$

Recapitulación

- Los módulos presentan algunas características interesantes:
  - Tienen estructuras genéricas fácilmente escalables.
  - Procesan palabras de datos y no solo bits individuales.
  - Pueden realizar distintas funciones según el valor de ciertas entradas de control.
  - Tienen funcionalidades abstractas que permiten diseñar/describir de manera estructurada sistemas complejos sin tener recurrir a EC/FC:
    - Basta con interconectarlos sin crear realimentaciones
    - Y usar discrecionalmente puertas (glue logic) para adaptar señales.

Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.

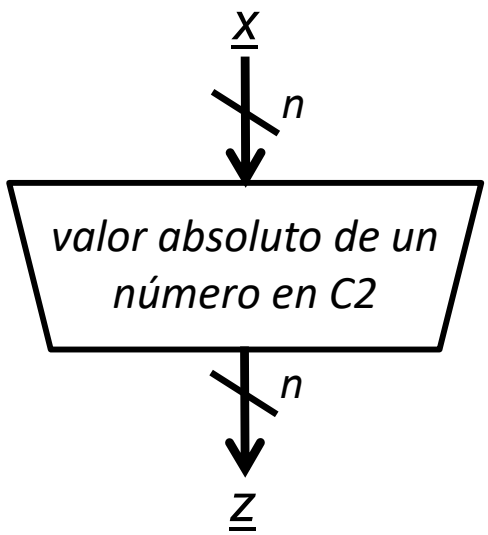


$$\underline{z} = \text{abs}(\underline{x}) = \begin{cases} \underline{x} & \text{si } x \geq 0 \\ -\underline{x} & \text{si } x < 0 \end{cases}$$

$$\underline{z} = \text{abs}(\underline{x}) =_{C2} \begin{cases} \underline{x} & \text{si } x_{n-1} = 0 \\ C2(\underline{x}) = \text{not}(\underline{x})+1 & \text{si } x_{n-1} = 1 \end{cases}$$

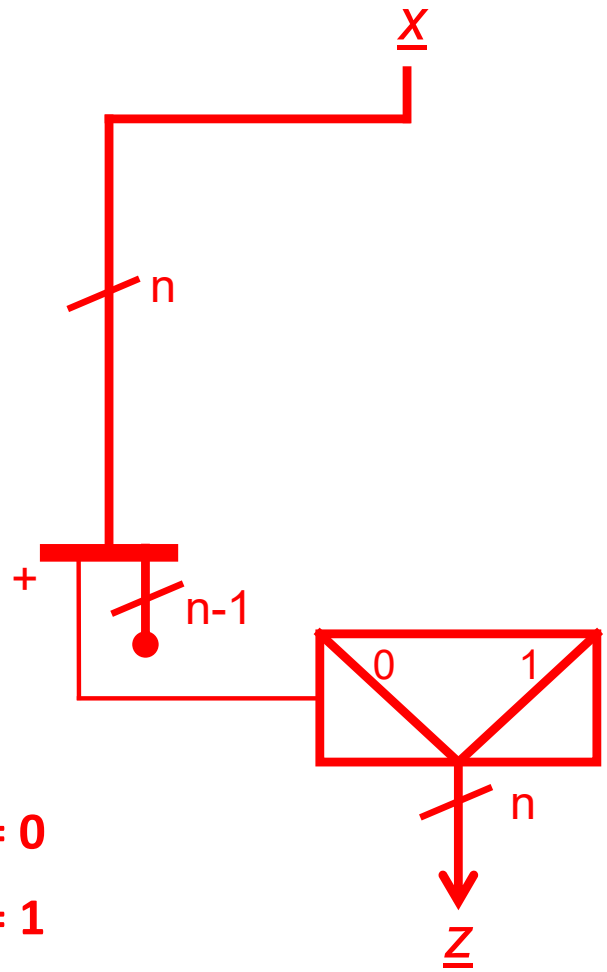
Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



$$z = abs(x) = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

$$z = abs(x) =_{C2} \begin{cases} x & \\ C2(x) = not(x)+1 & \end{cases}$$

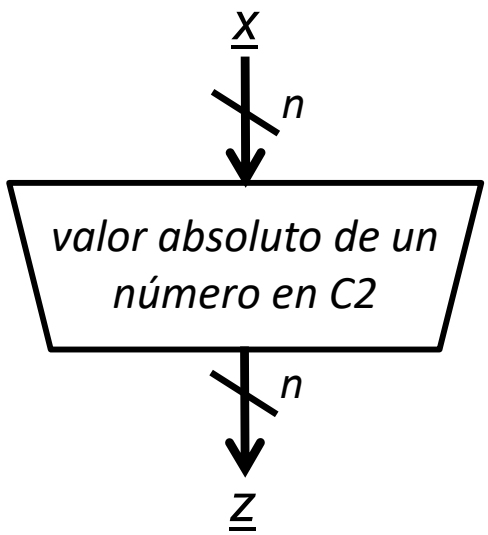


si $x_{n-1} = 0$

si $x_{n-1} = 1$

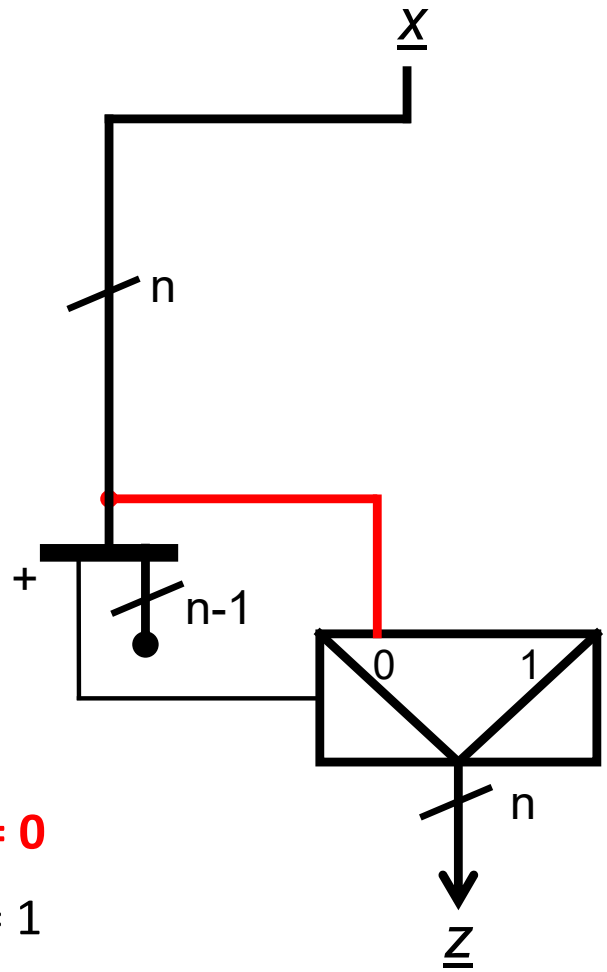
Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



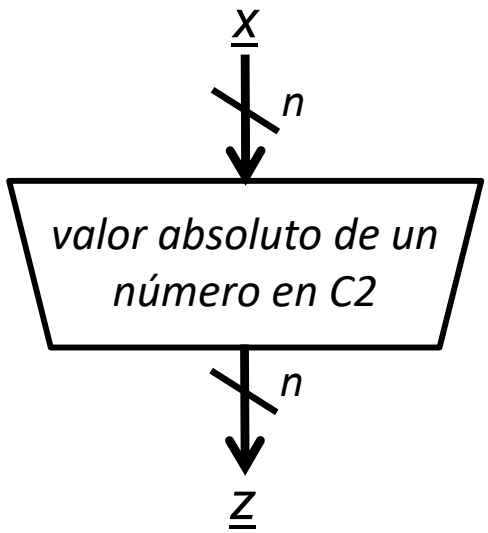
$$\underline{z} = abs(\underline{x}) = \begin{cases} \underline{x} & \text{si } x \geq 0 \\ -\underline{x} & \text{si } x < 0 \end{cases}$$

$$\underline{z} = abs(\underline{x}) =_{C2} \begin{cases} \underline{x} & \text{si } x_{n-1} = 0 \\ C2(\underline{x}) = not(\underline{x})+1 & \text{si } x_{n-1} = 1 \end{cases}$$



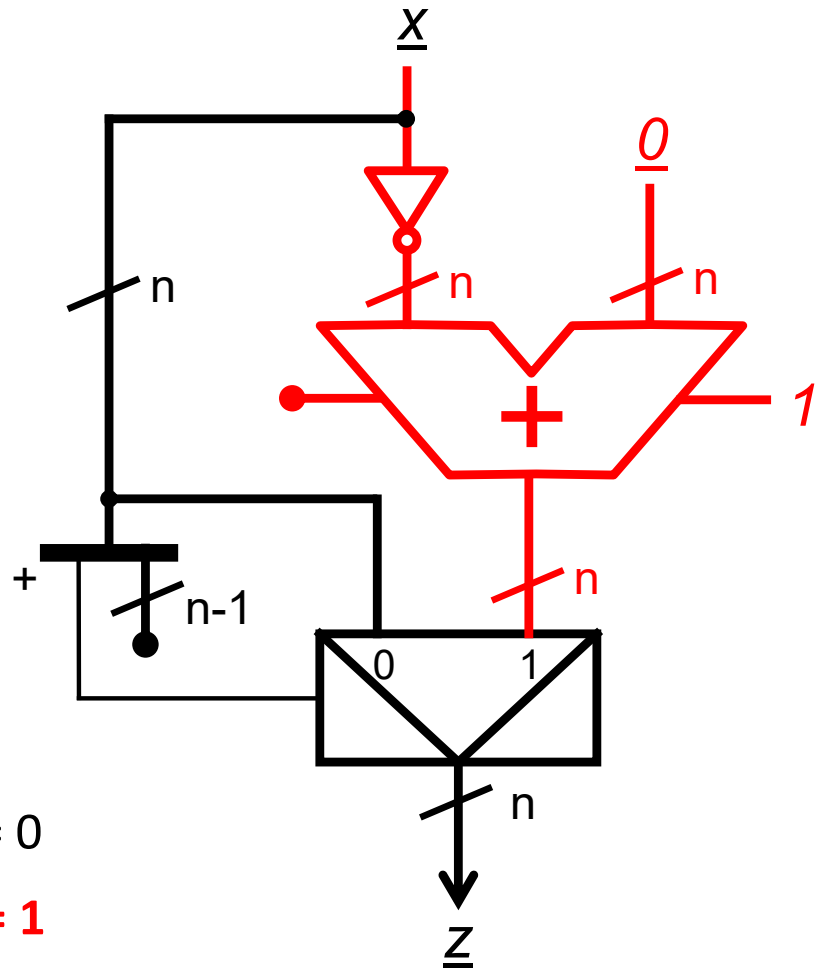
Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



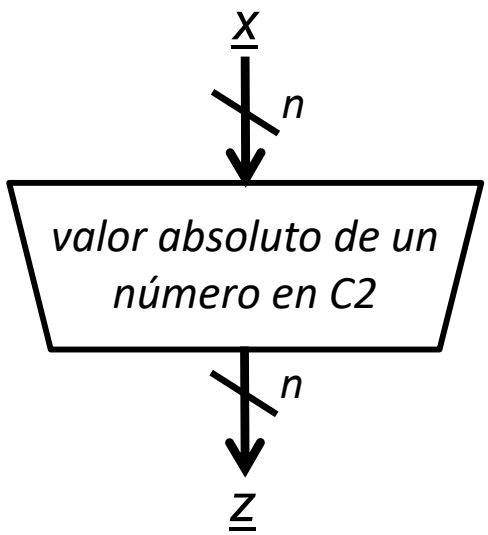
$$\underline{z} = abs(\underline{x}) = \begin{cases} \underline{x} & \text{si } x \geq 0 \\ -\underline{x} & \text{si } x < 0 \end{cases}$$

$$\underline{z} = abs(\underline{x}) =_{C2} \begin{cases} \underline{x} & \text{si } x_{n-1} = 0 \\ C2(\underline{x}) = \mathbf{not(\underline{x})+1} & \mathbf{si\ x_{n-1} = 1} \end{cases}$$



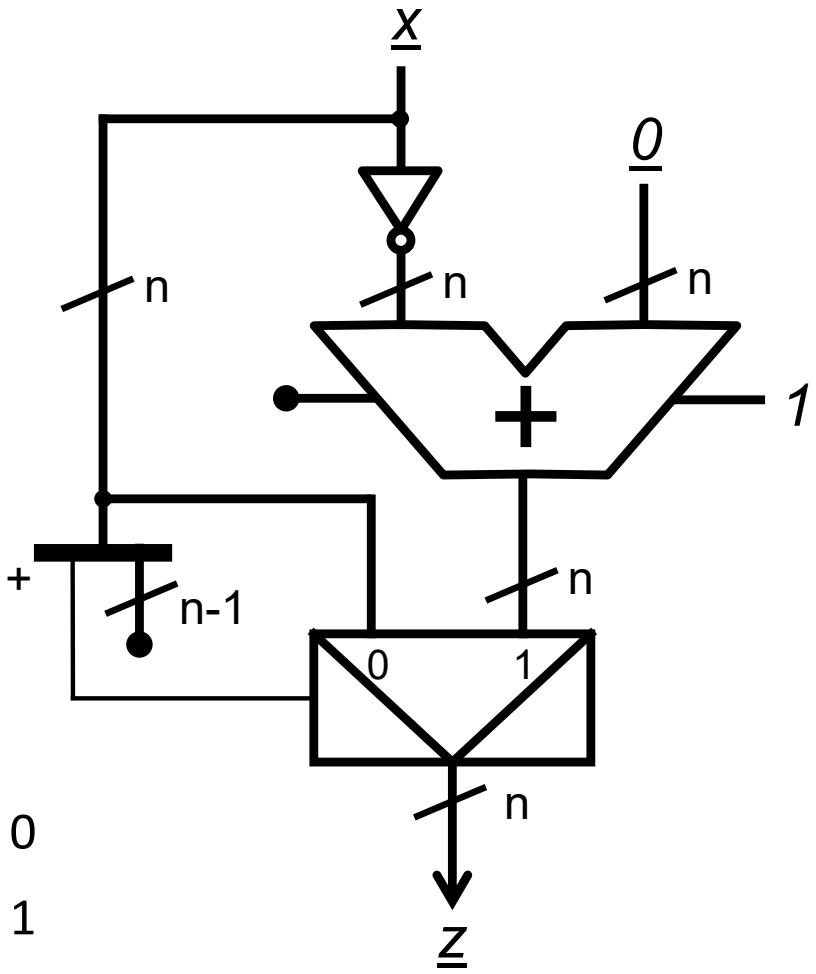
Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



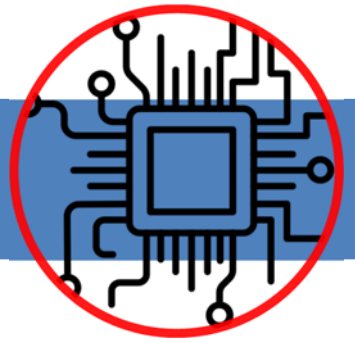
$$\underline{z} = abs(\underline{x}) = \begin{cases} \underline{x} & \text{si } x \geq 0 \\ -\underline{x} & \text{si } x < 0 \end{cases}$$

$$\underline{z} = abs(\underline{x}) =_{C2} \begin{cases} \underline{x} & \text{si } x_{n-1} = 0 \\ C2(\underline{x}) = not(\underline{x})+1 & \text{si } x_{n-1} = 1 \end{cases}$$



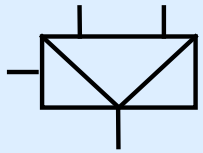
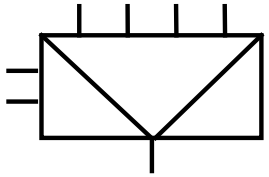
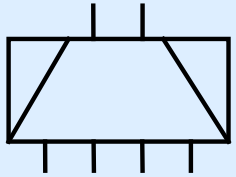
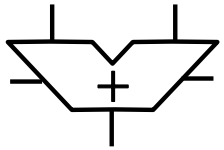
- Biblioteca de celdas.

Apéndice tecnológico



Biblioteca de celdas

CMOS 90 nm

Módulo	Área (μm^2)	Retardo (ps)	Consumo estático (nW)	Consumo dinámico (nW/MHz)
	11.0592	223	84	8639
	23.0400	250	163	15169
	29.4912	191 (z_0) 189 (z_1) 132 (z_2) 127 (z_3)	23	543
	29.4912	205 (s) 226 (c)	159	5374 (s) 713 (c)

fuente: Synopsys (SAED EDK 90 nm)

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/