

Organización del Procesador

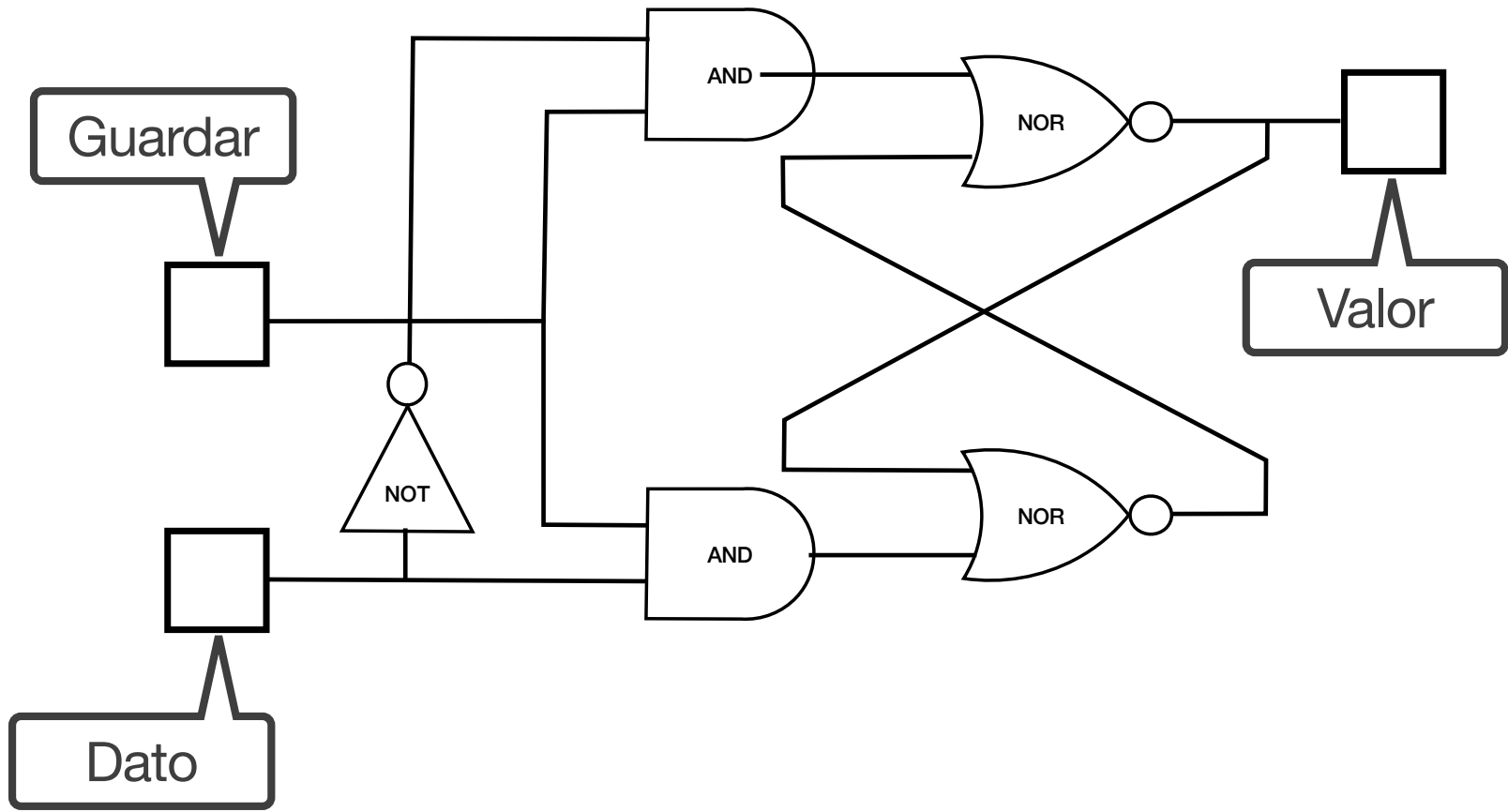
Cómo computar utilizando la electricidad

Departamento de Computación - UNRC

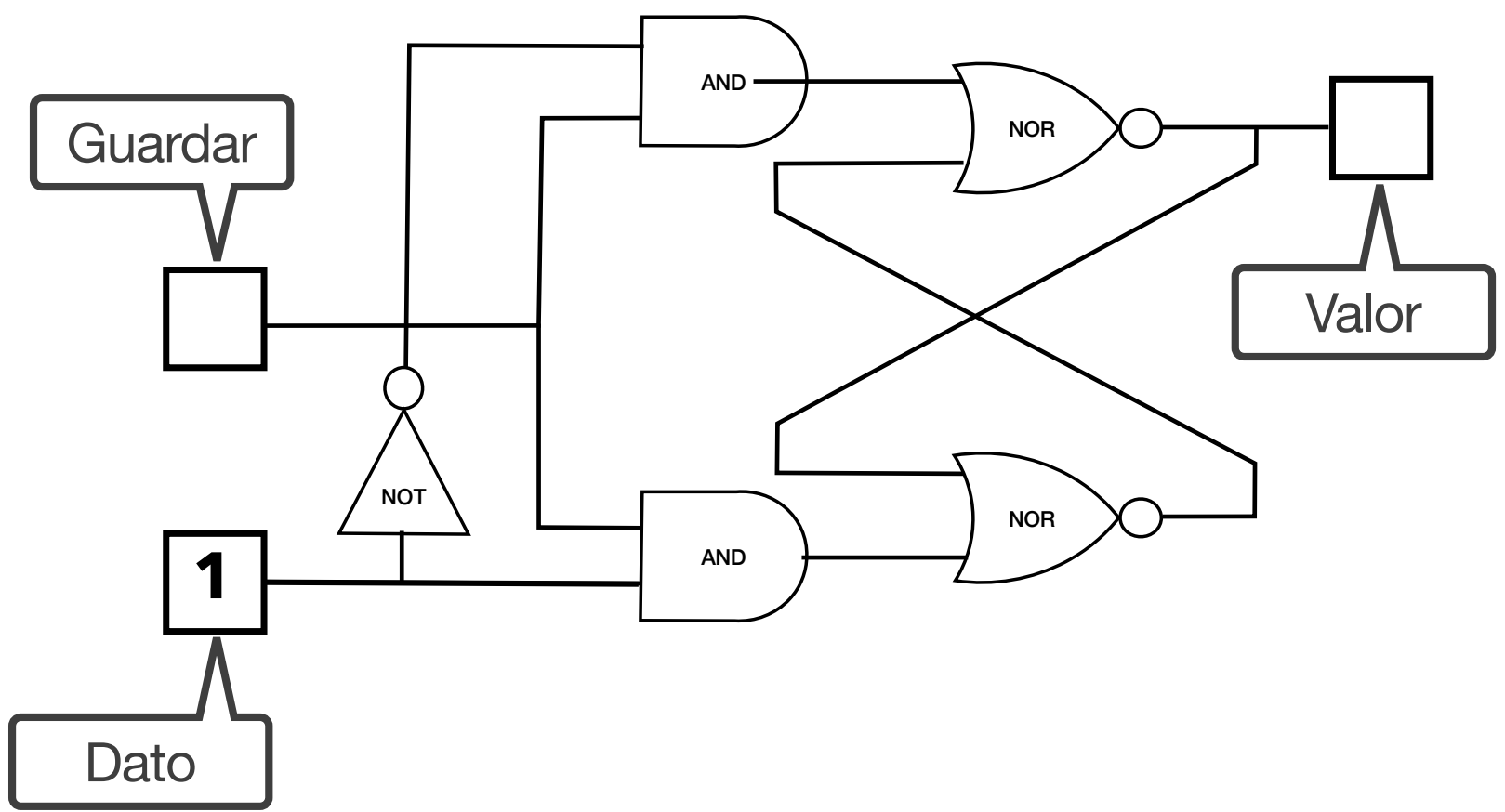
El camino a recorrer

- Un poco de Historia y Sistemas Numéricos
- Introducción a la Electrónica
- Representación de Información
- **Cómo computar utilizando la electricidad**
- Evolución y Funcionamiento abstracto de una computadora
- Assembly X86
- Micro-programación (cómo fabricar un procesador)
- Eficiencia
 - Pipelines
 - Memoria Caché
 - Memoria Virtual

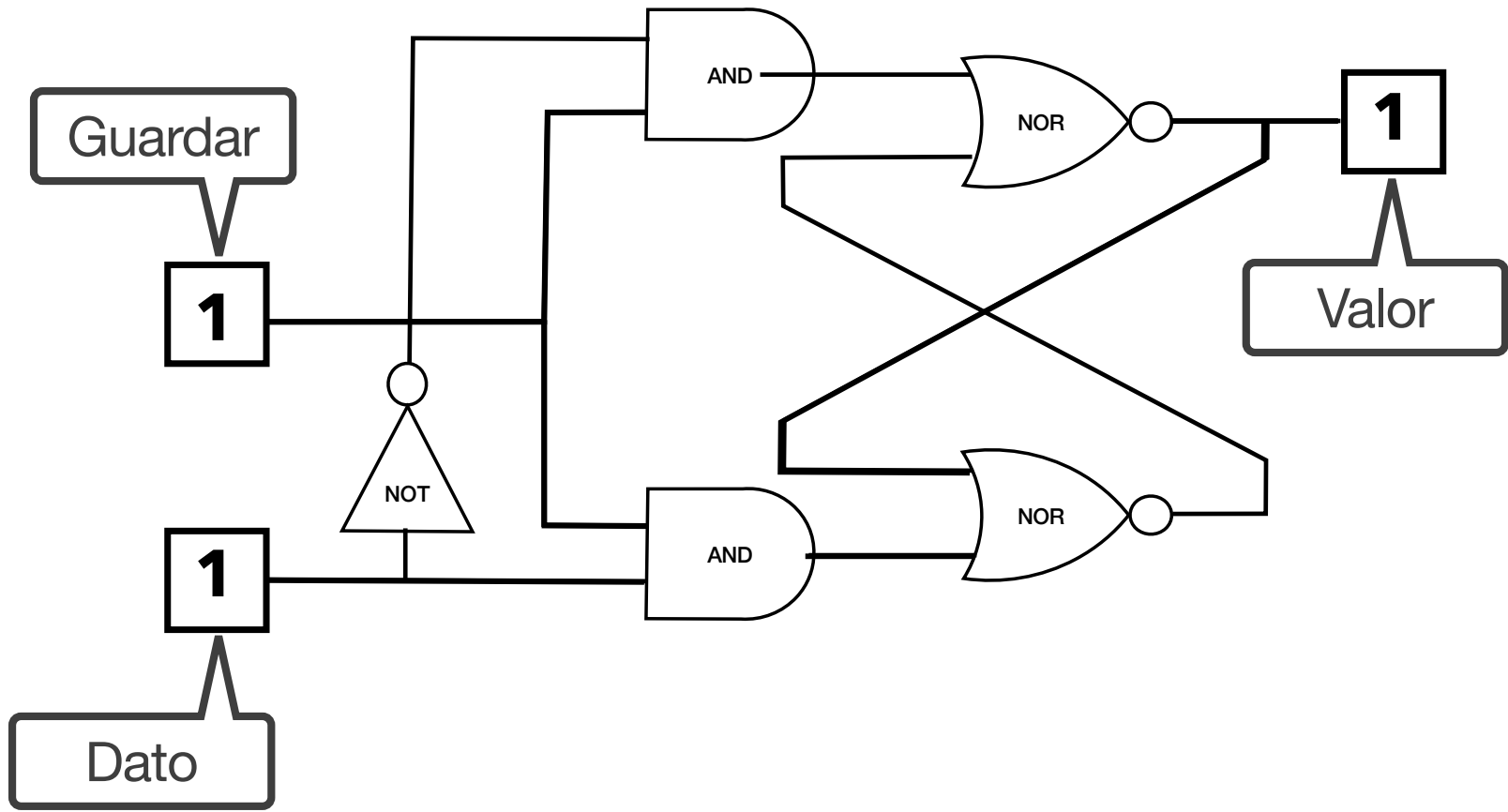
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



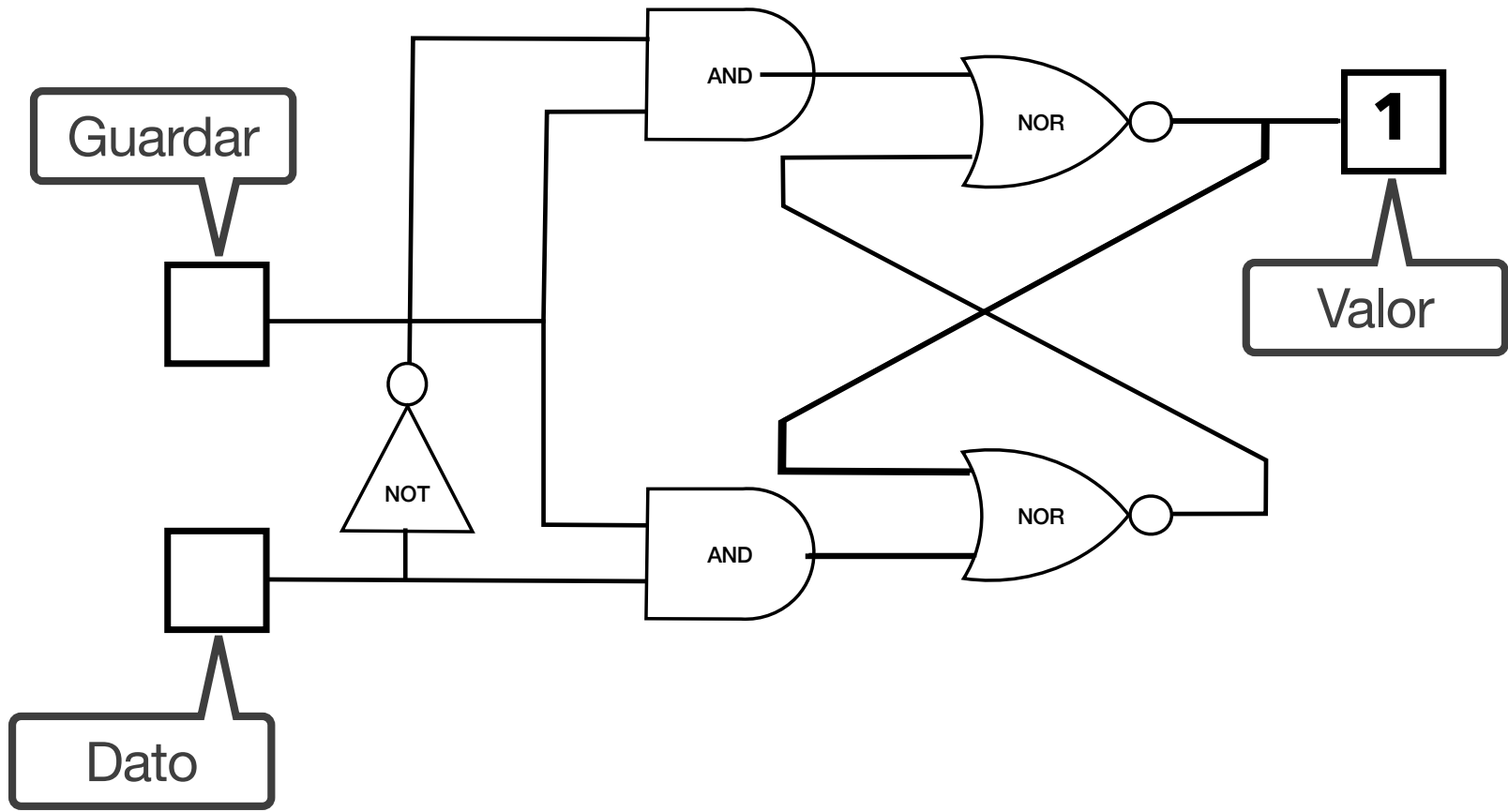
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



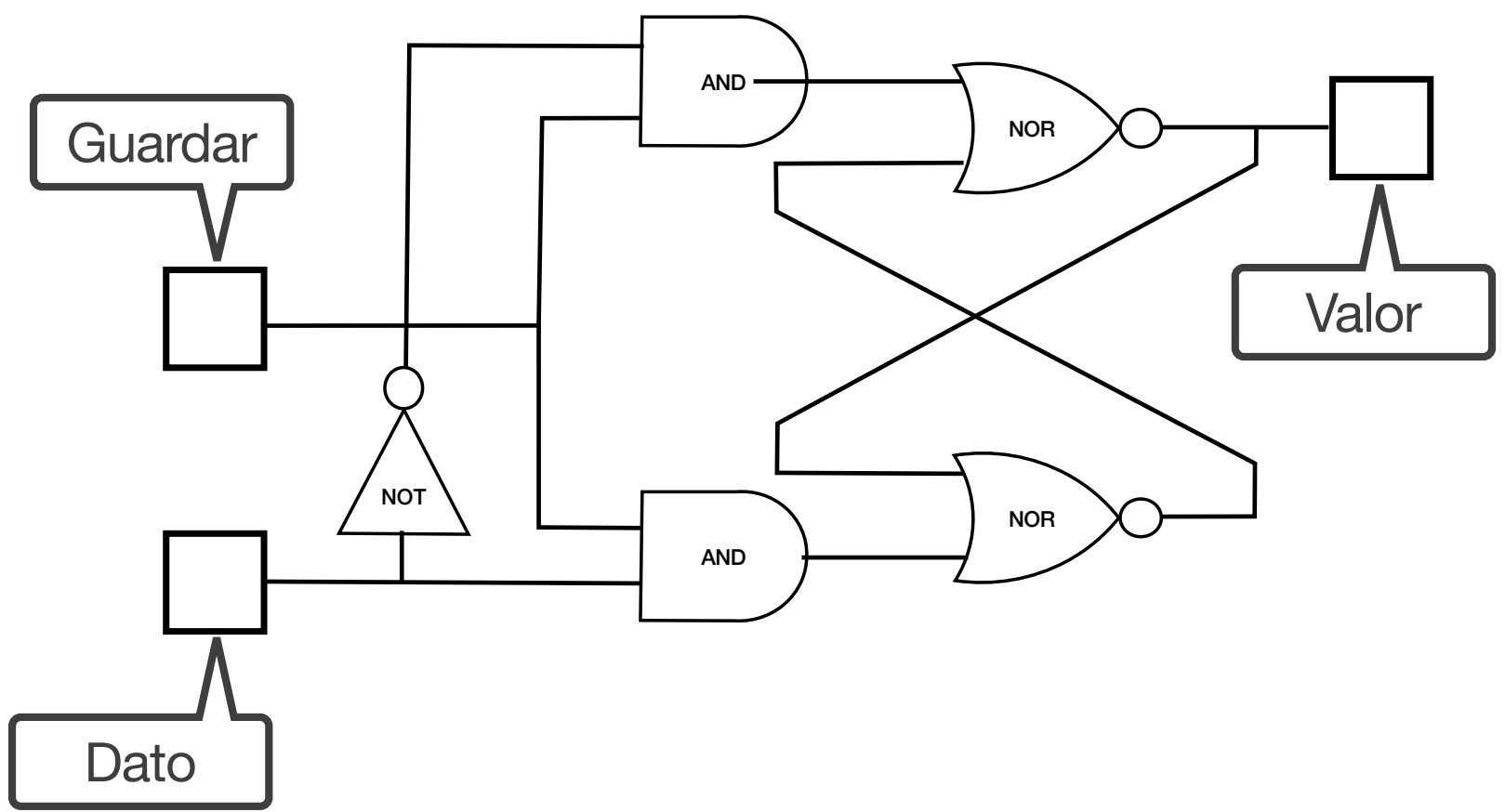
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



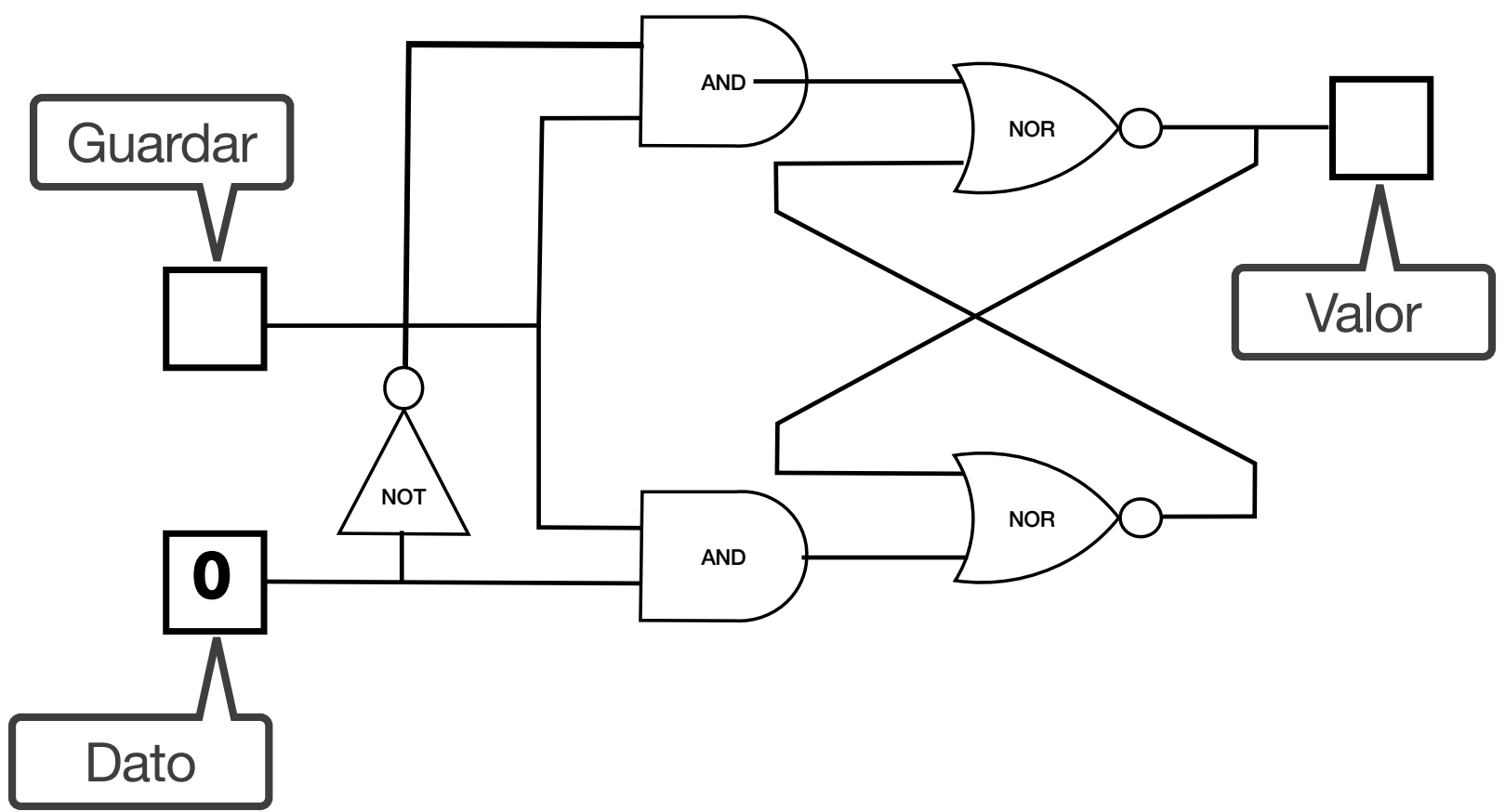
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



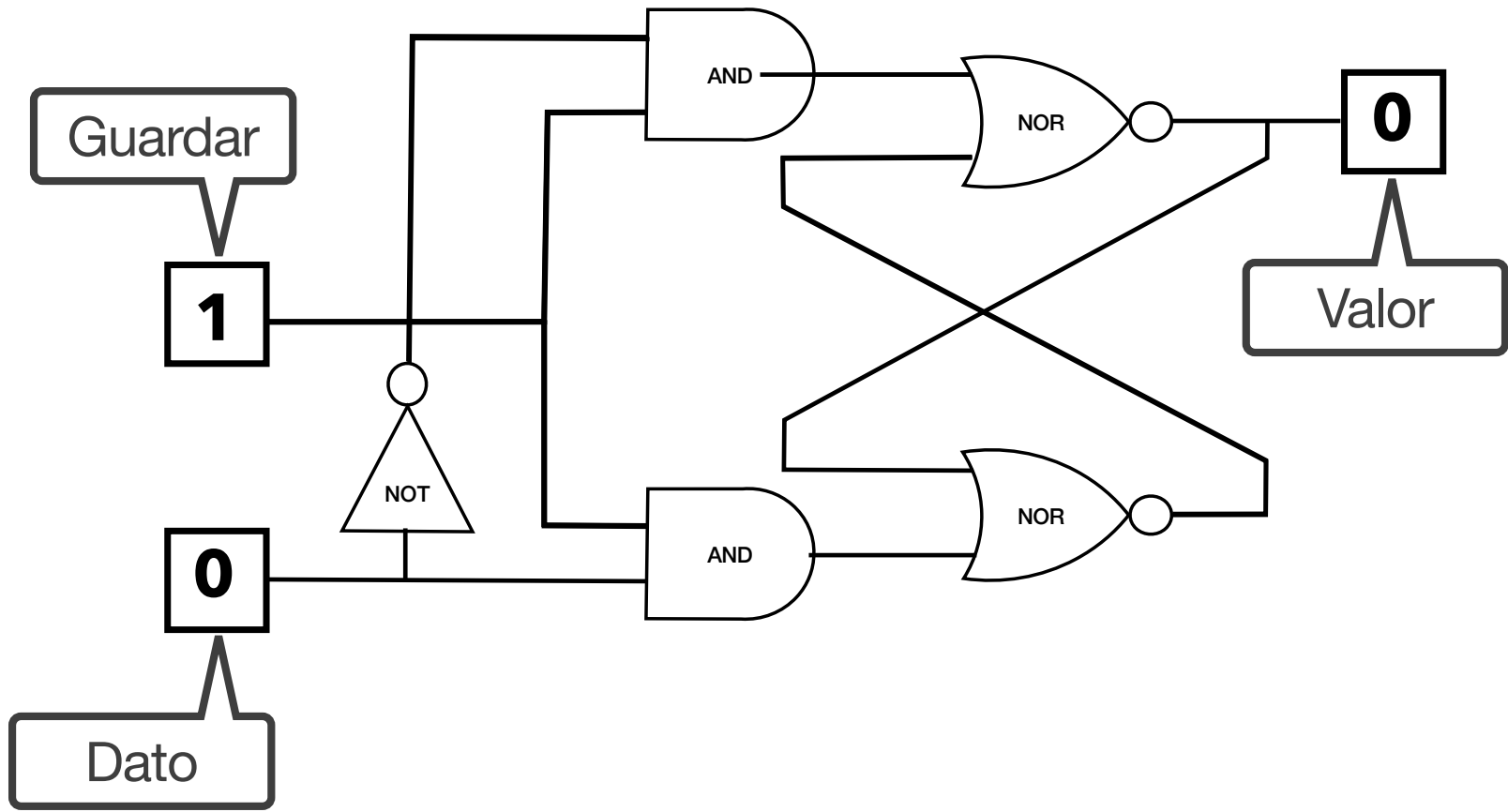
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



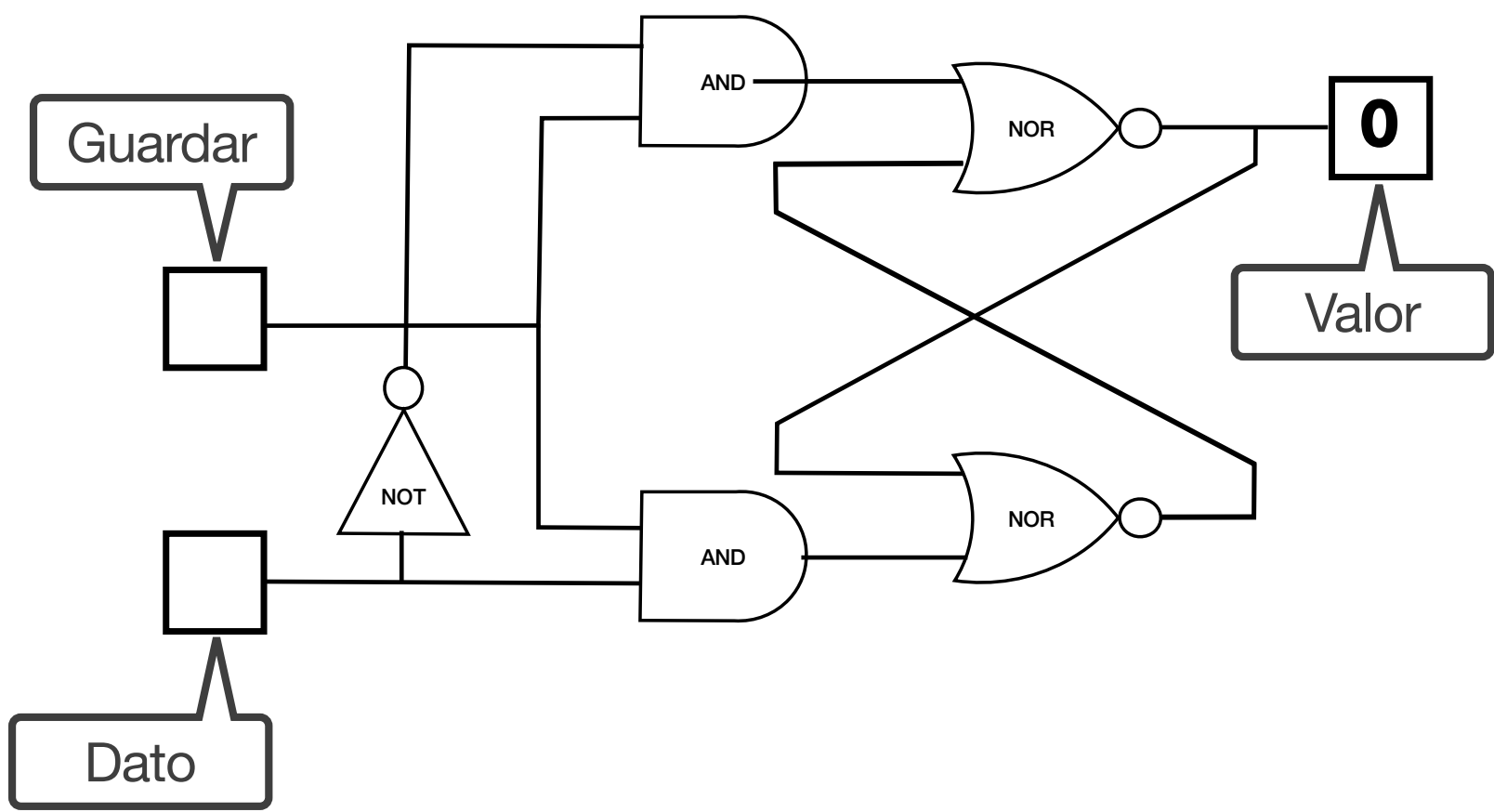
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



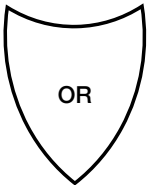

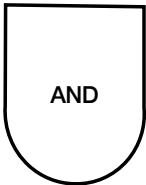
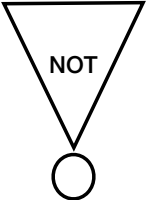
Cómo podemos controlar la corriente eléctrica - Un bit de memoria



Cómo podemos controlar la corriente eléctrica - Un bit de memoria

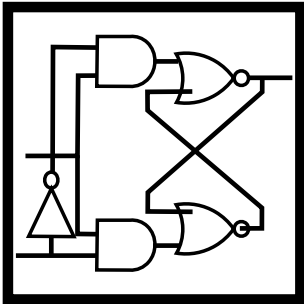


Cómo podemos controlar la corriente eléctrica - Compuertas Lógicas

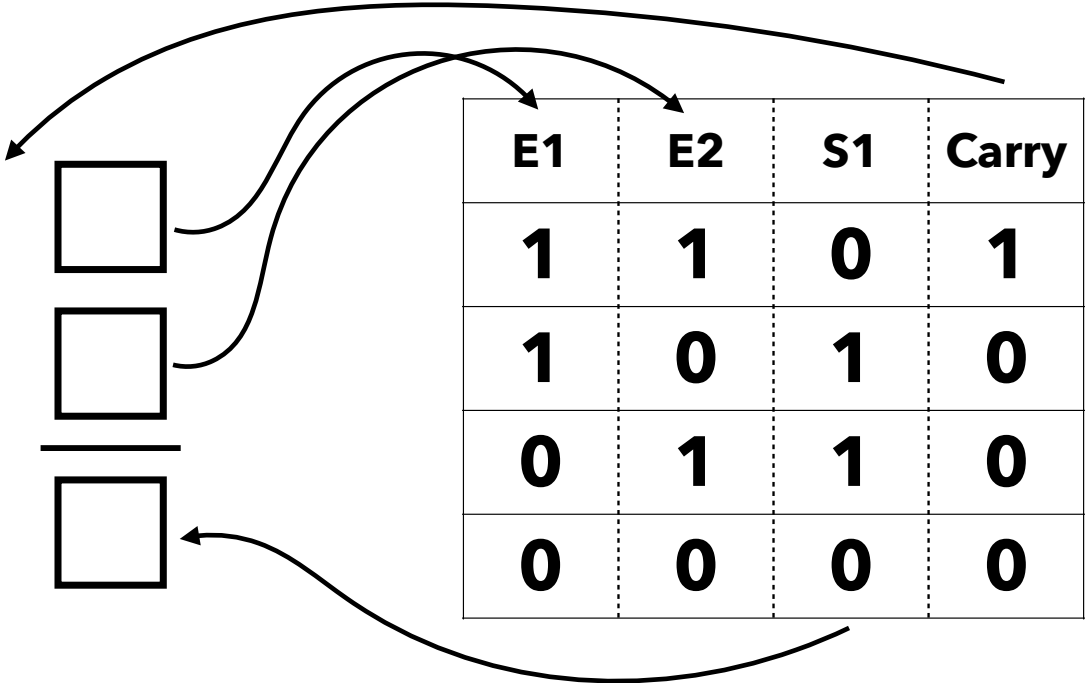
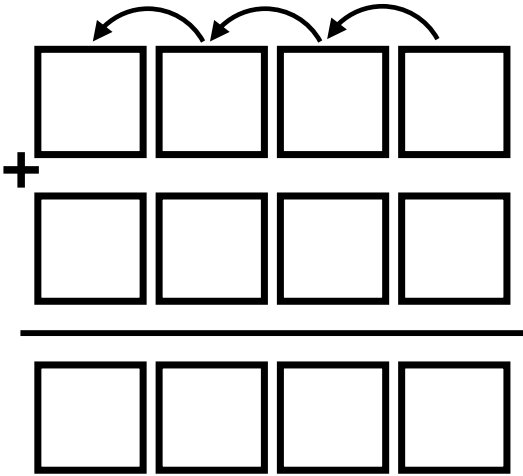
					
E1	E2	$E1 \wedge E2$	$E1 \vee E2$	$E1 \underline{\vee} E2$	$\neg E1$
1	1	1	1	0	0
1	0	0	1	1	
0	1	0	1	1	1
0	0	0	0	0	

Comencemos - Rehaciendo la PASCALINA con electricidad

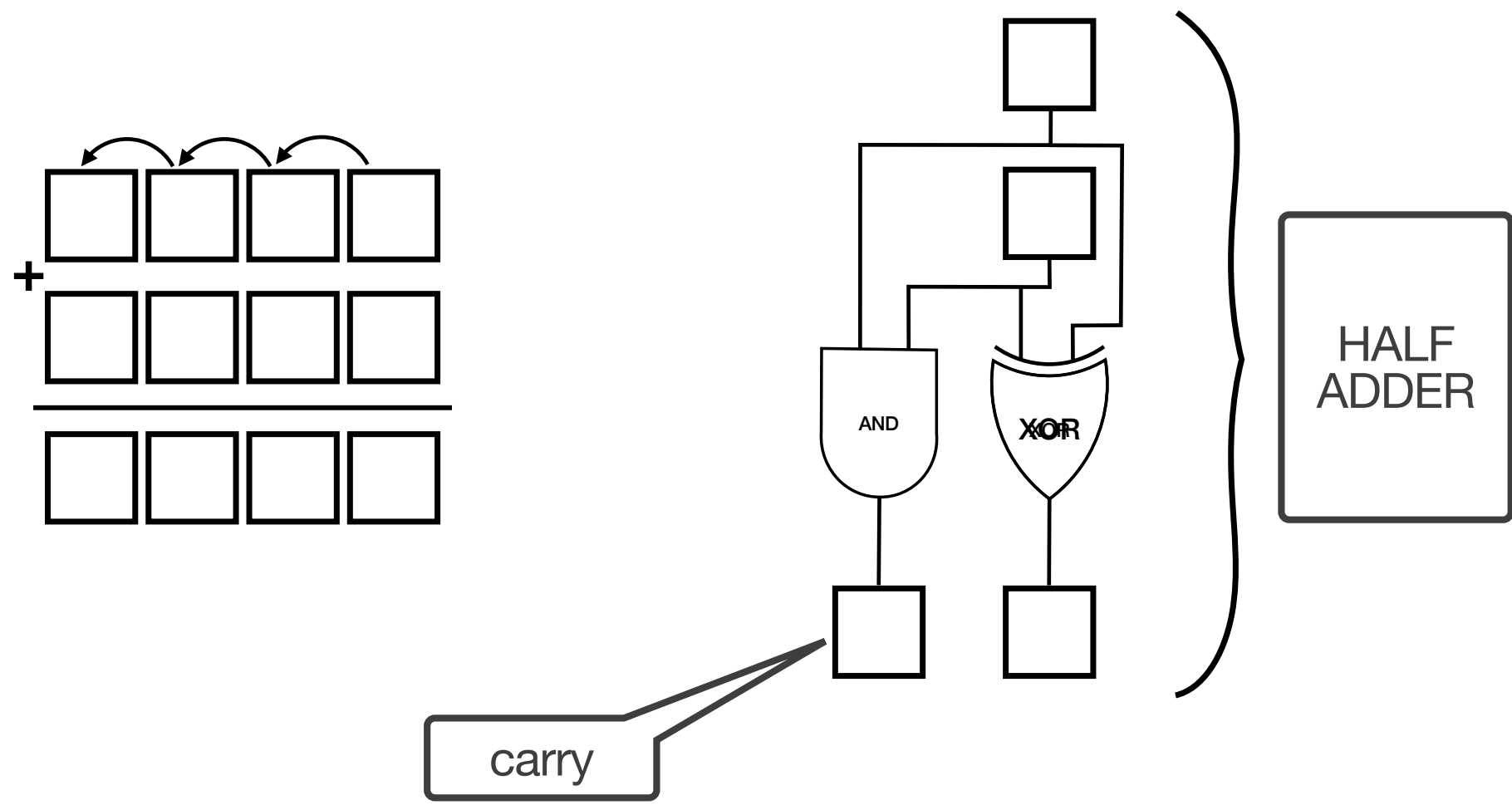
carry



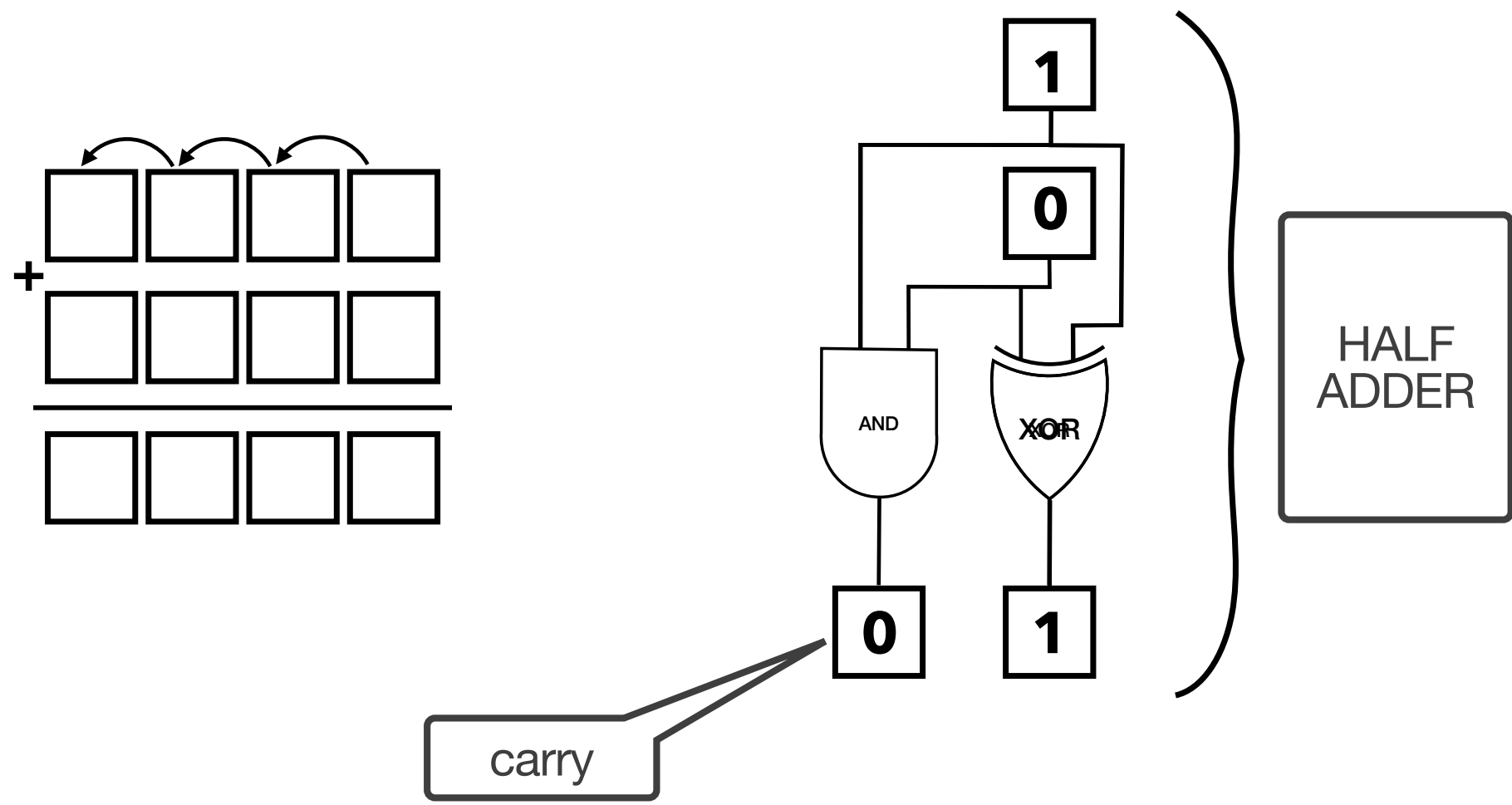
1 bit



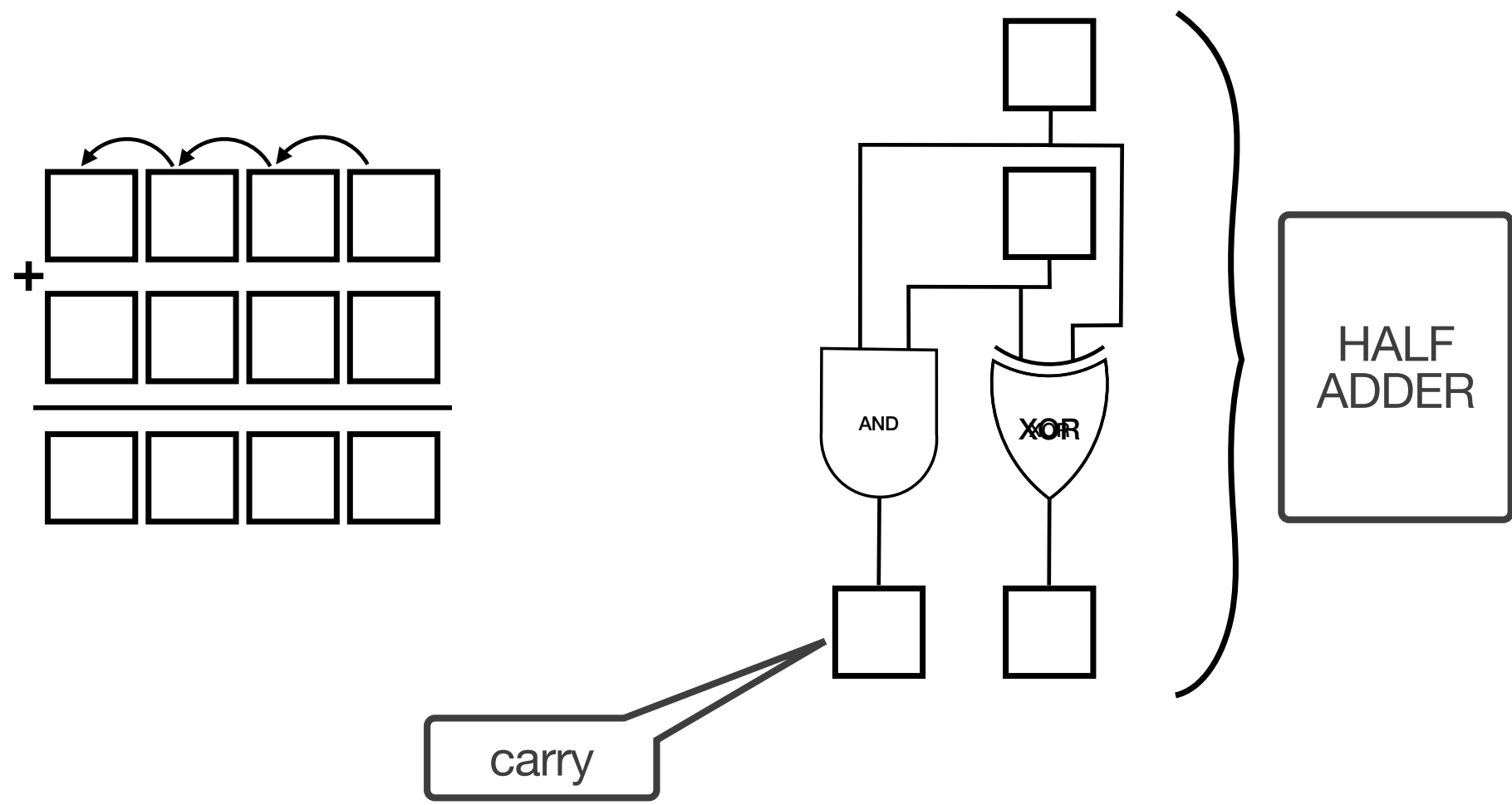
Comencemos - Rehaciendo la PASCALINA con electricidad



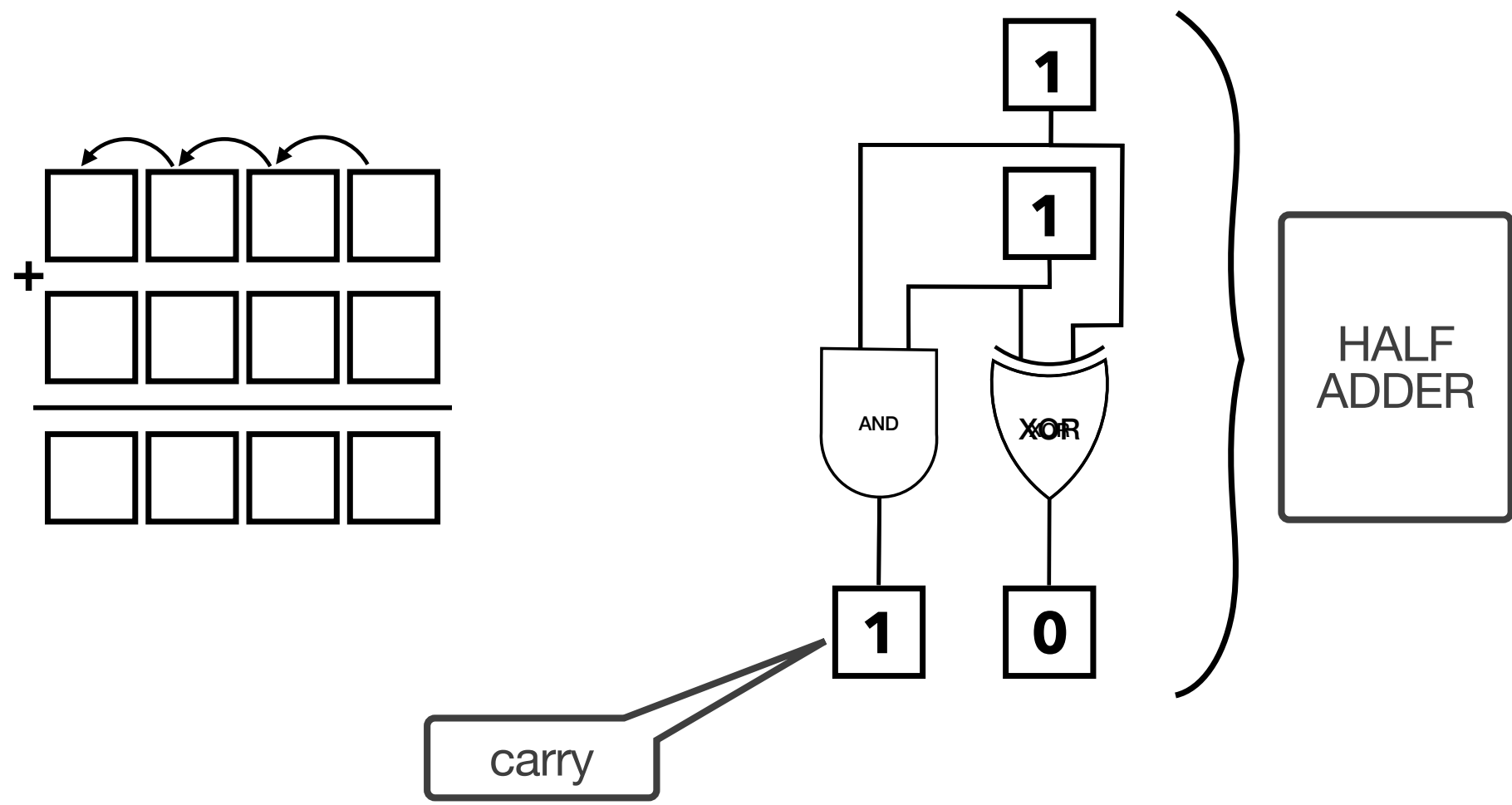
Comencemos - Rehaciendo la PASCALINA con electricidad



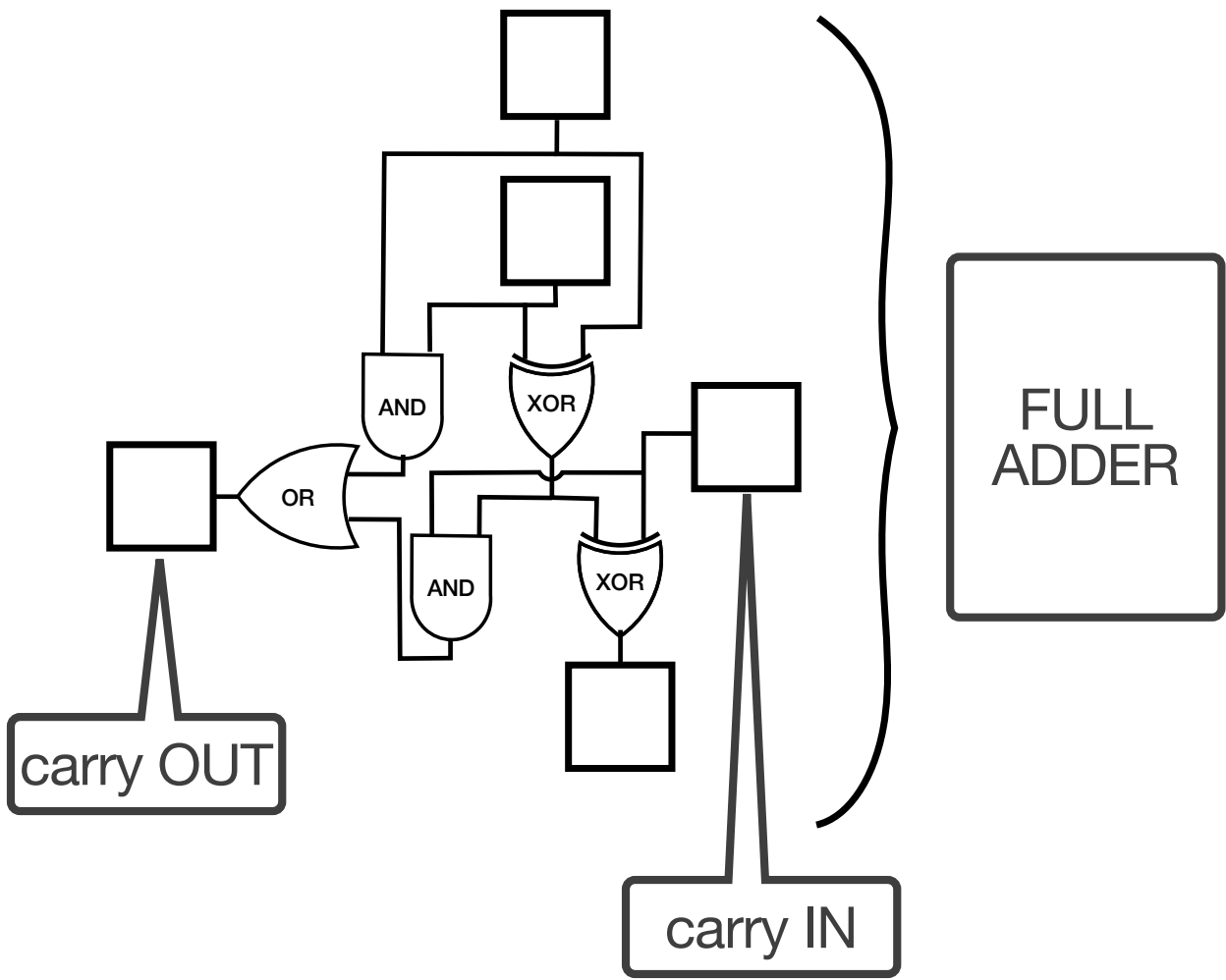
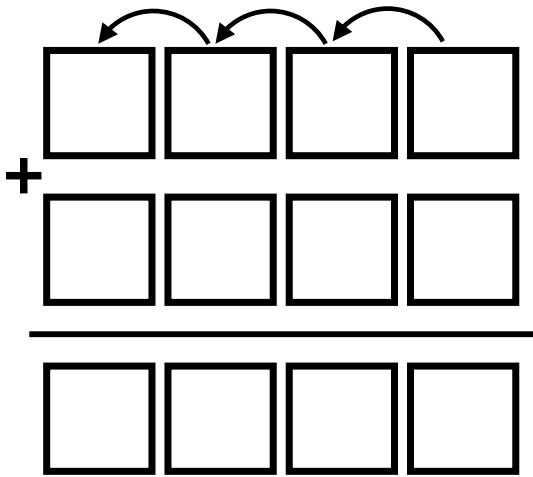
Comencemos - Rehaciendo la PASCALINA con electricidad



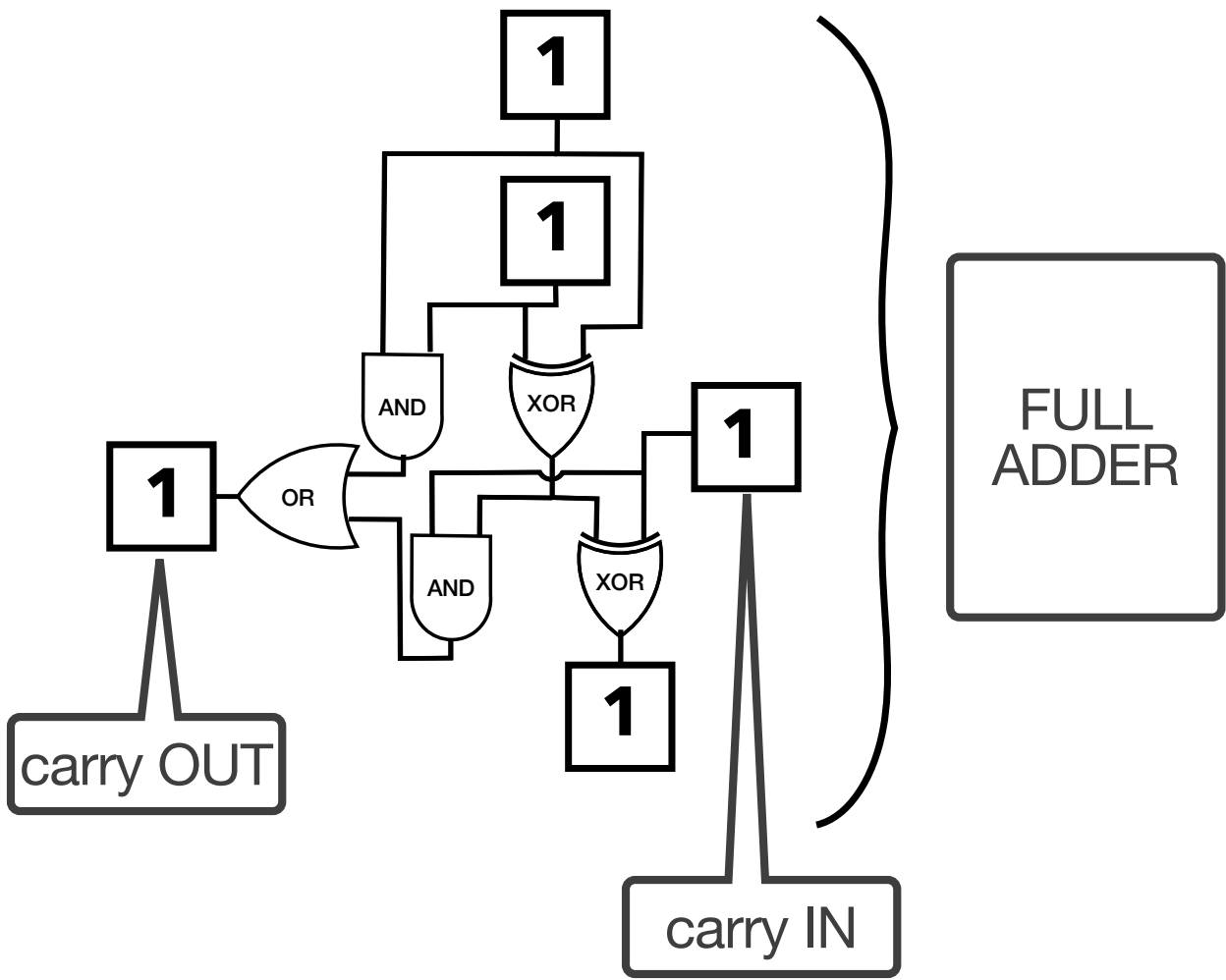
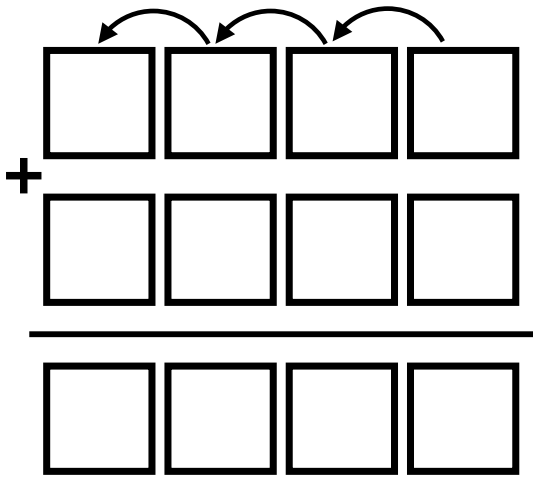
Comencemos - Rehaciendo la PASCALINA con electricidad



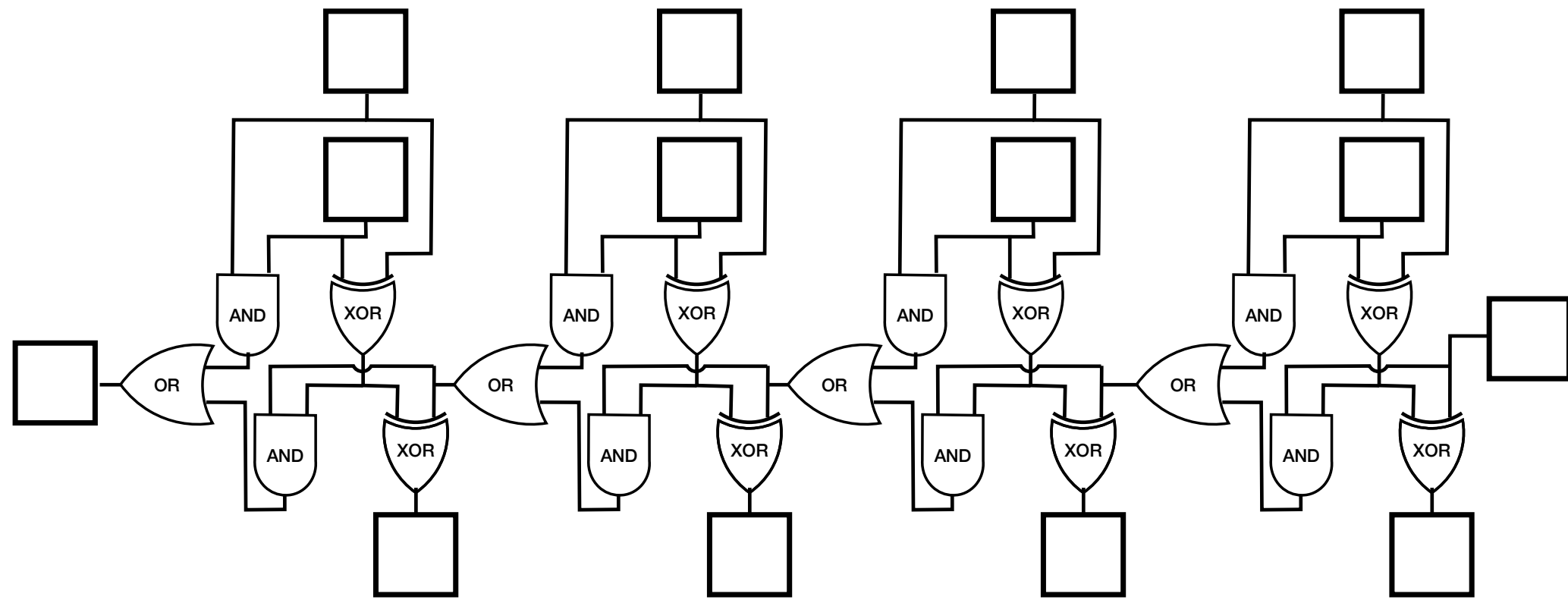
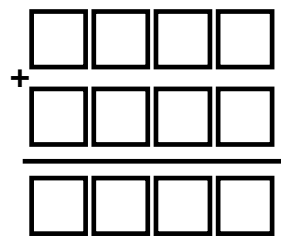
Comencemos - Rehaciendo la PASCALINA con electricidad



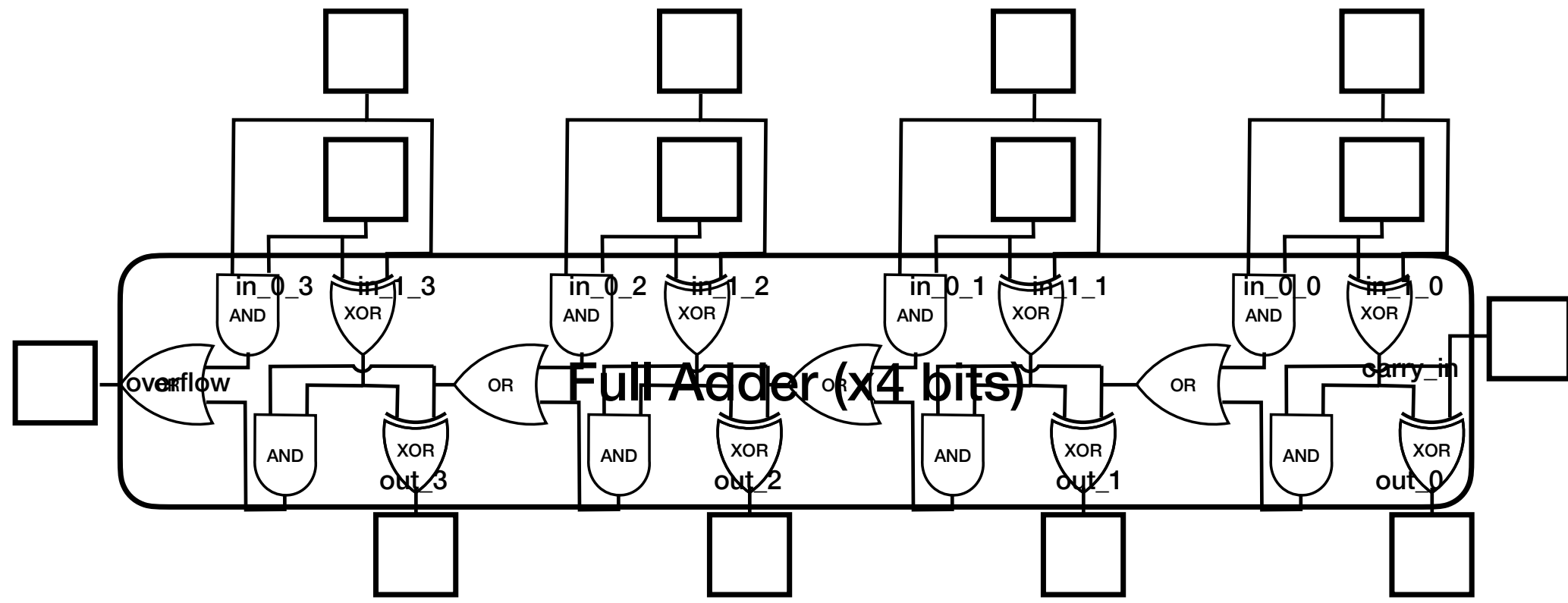
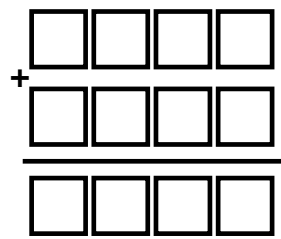
Comencemos - Rehaciendo la PASCALINA con electricidad



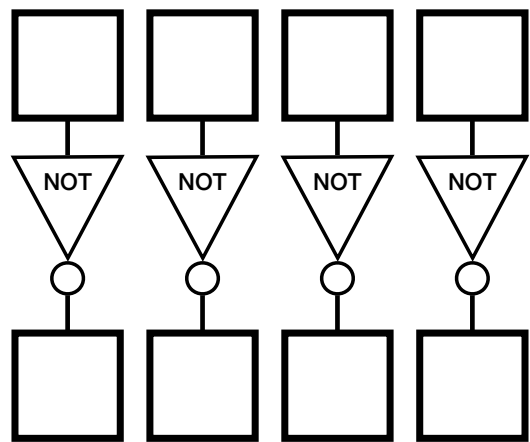
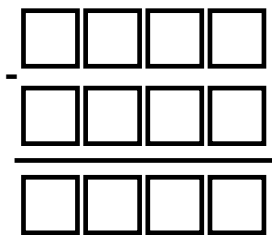
Comencemos - Rehaciendo la PASCALINA con electricidad



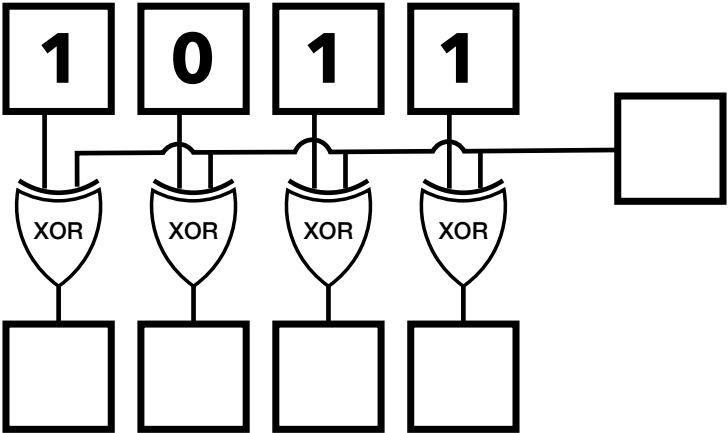
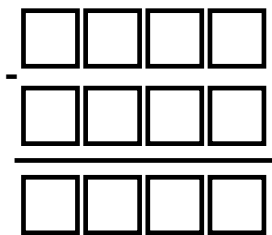
Comencemos - Rehaciendo la PASCALINA con electricidad



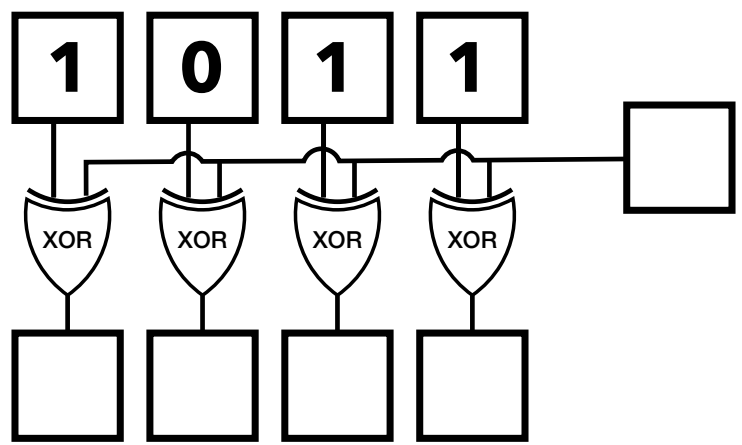
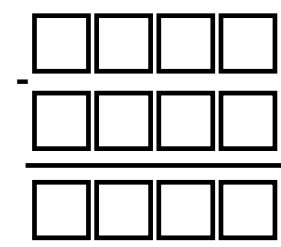
Comencemos - ¿Cómo restamos? - Complemento



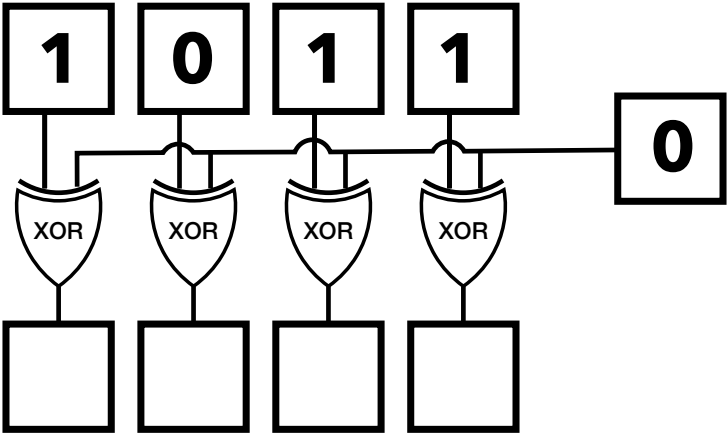
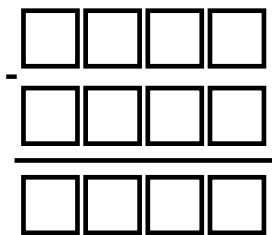
Comencemos - ¿Cómo restamos? - Complemento



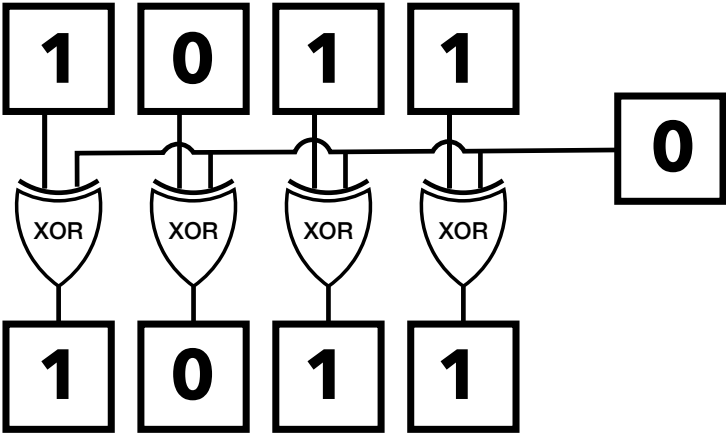
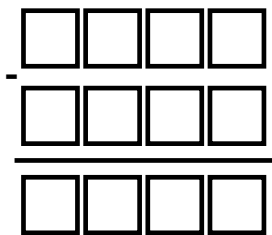
Comencemos - ¿Cómo restamos? - Complemento



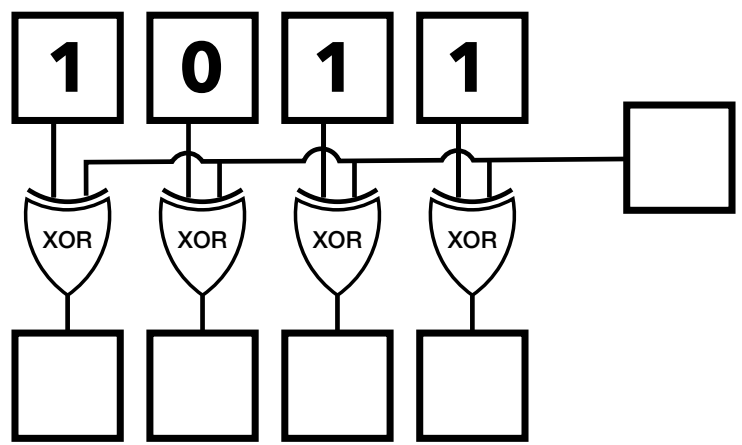
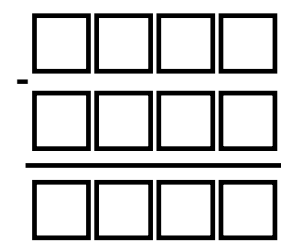
Comencemos - ¿Cómo restamos? - Complemento



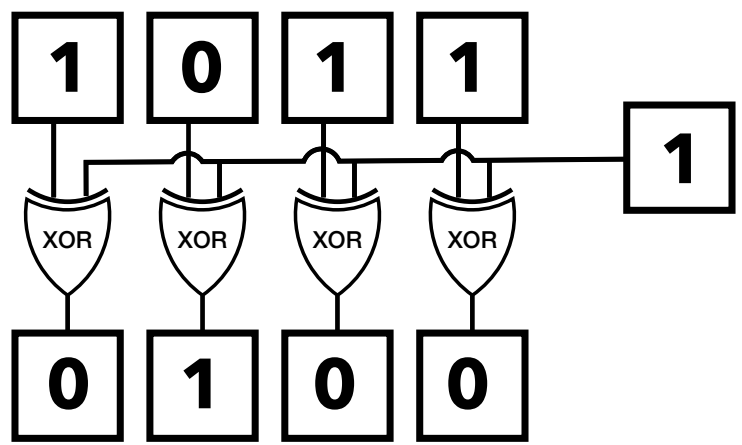
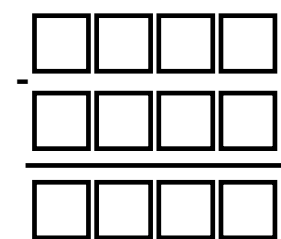
Comencemos - ¿Cómo restamos? - Complemento



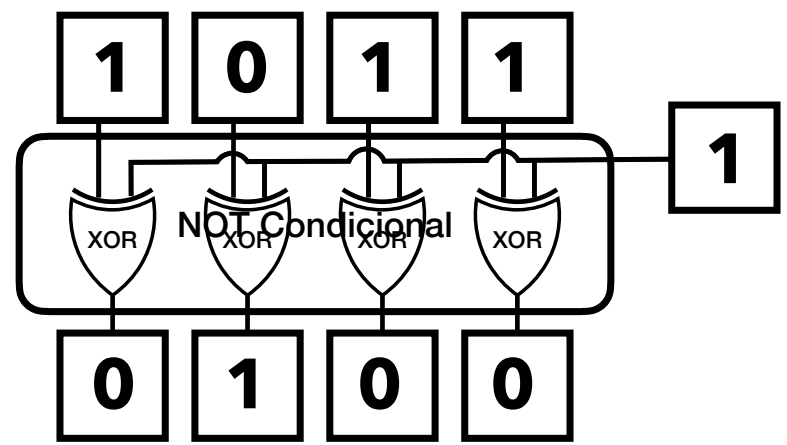
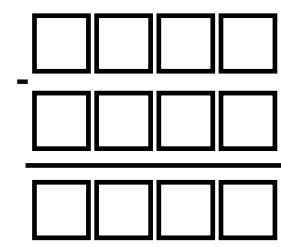
Comencemos - ¿Cómo restamos? - Complemento



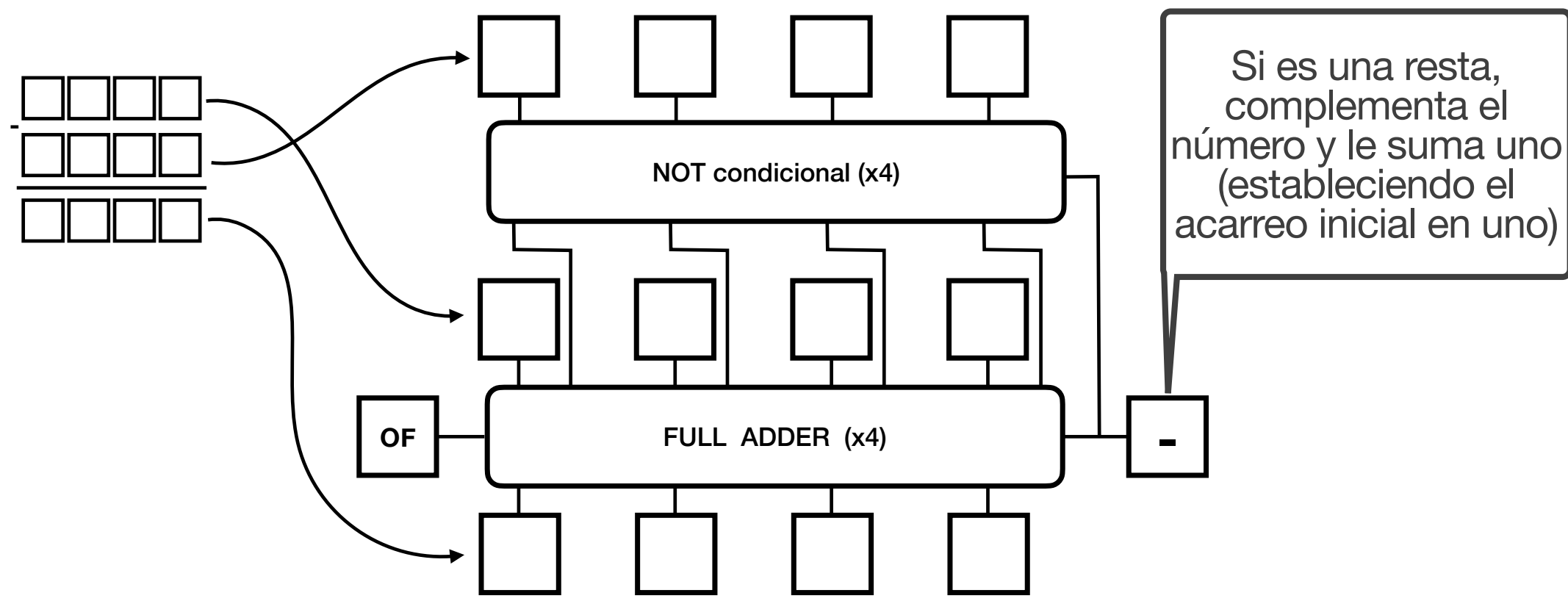
Comencemos - ¿Cómo restamos? - Complemento



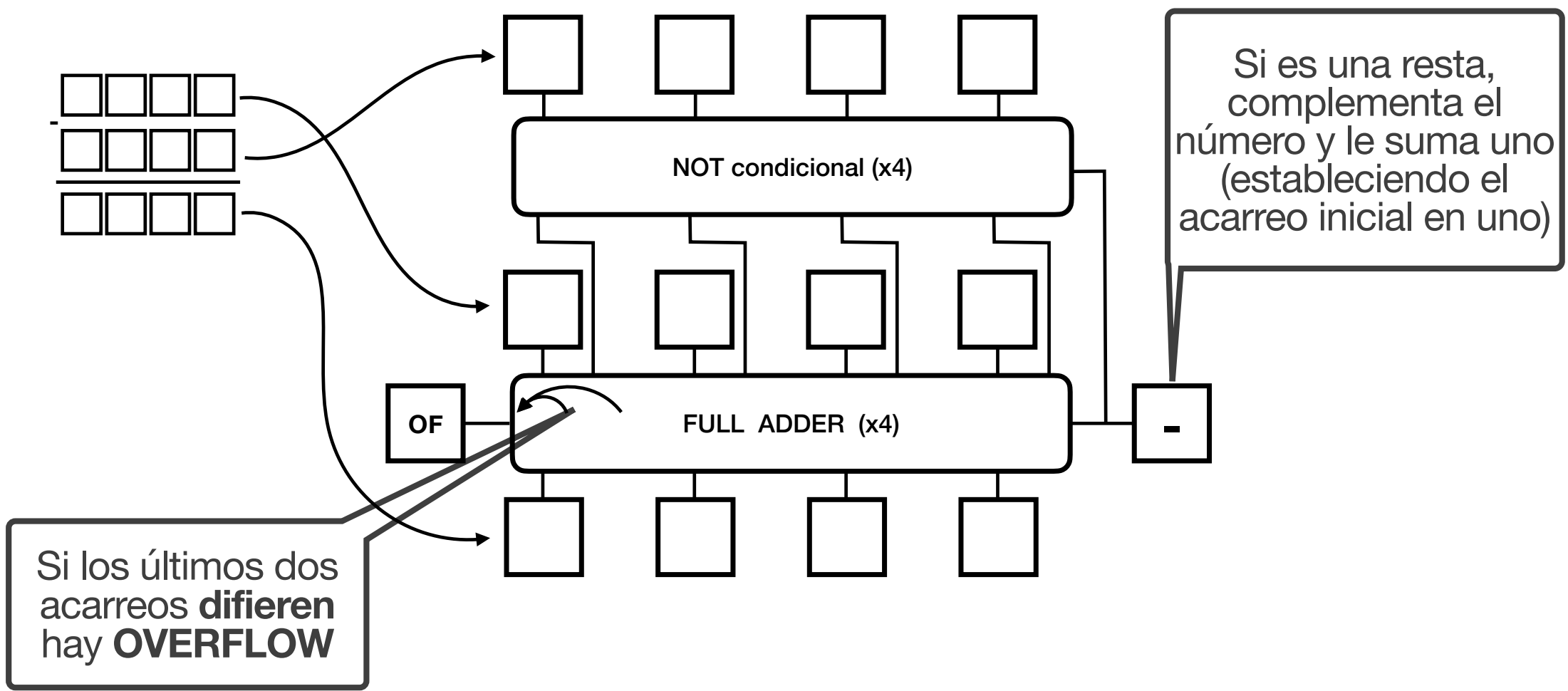
Comencemos - ¿Cómo restamos? - Complemento



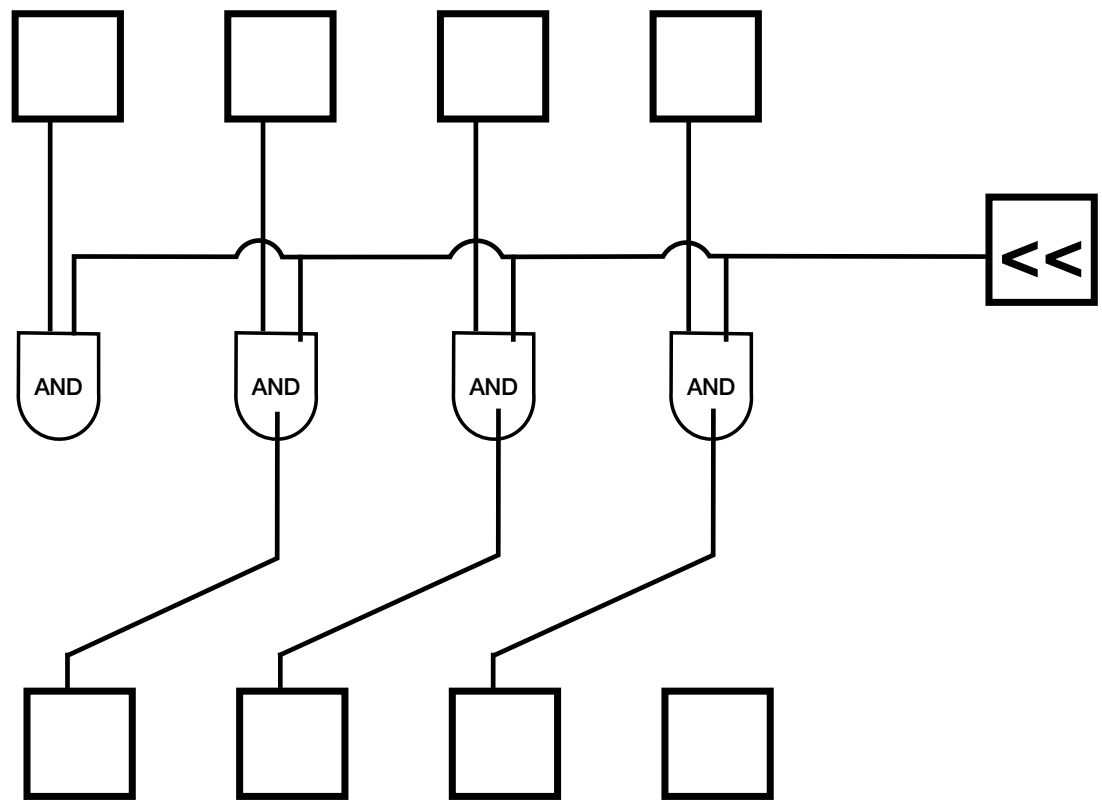
Comencemos - ¿Cómo restamos? - Complemento



Comencemos - ¿Cómo restamos? - Complemento



Algunas operaciones necesarias - SHIFT (desplazar)

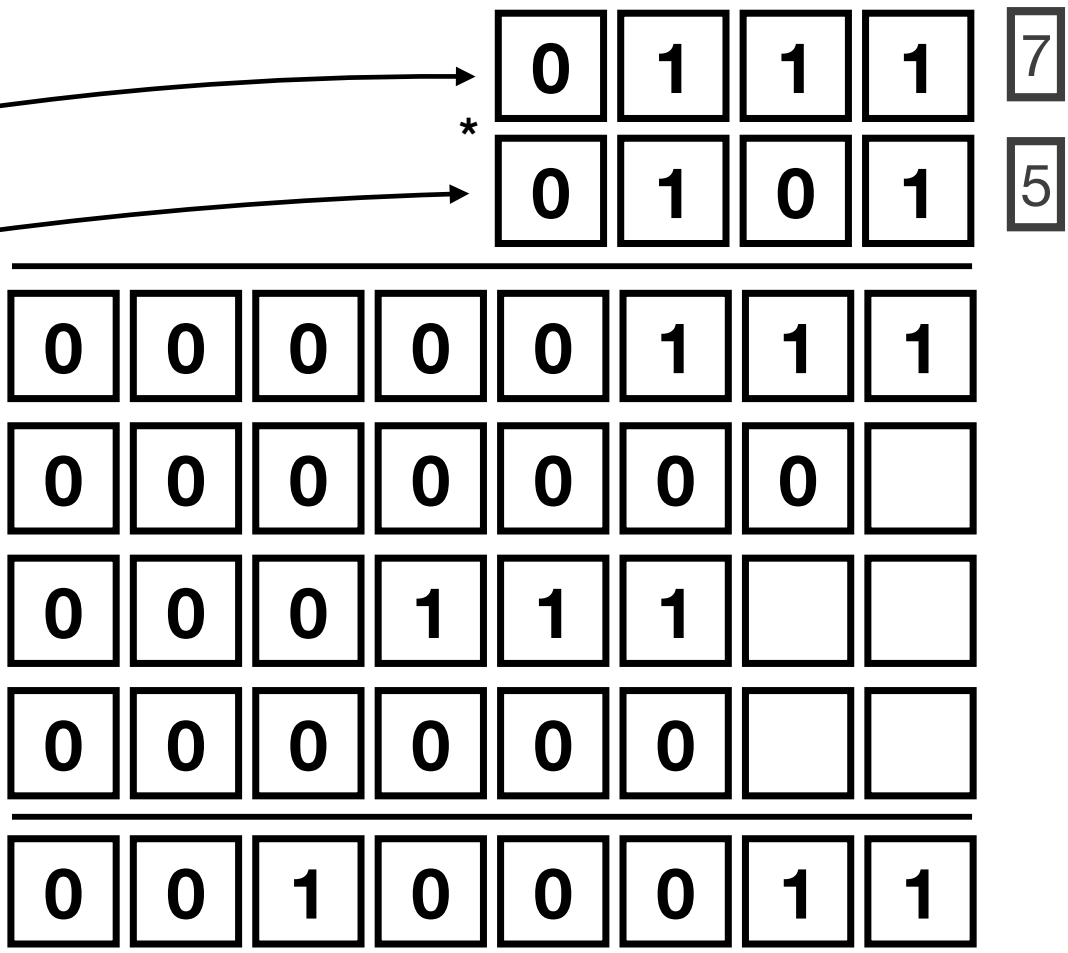


¿Cómo multiplicamos? - algunos tips

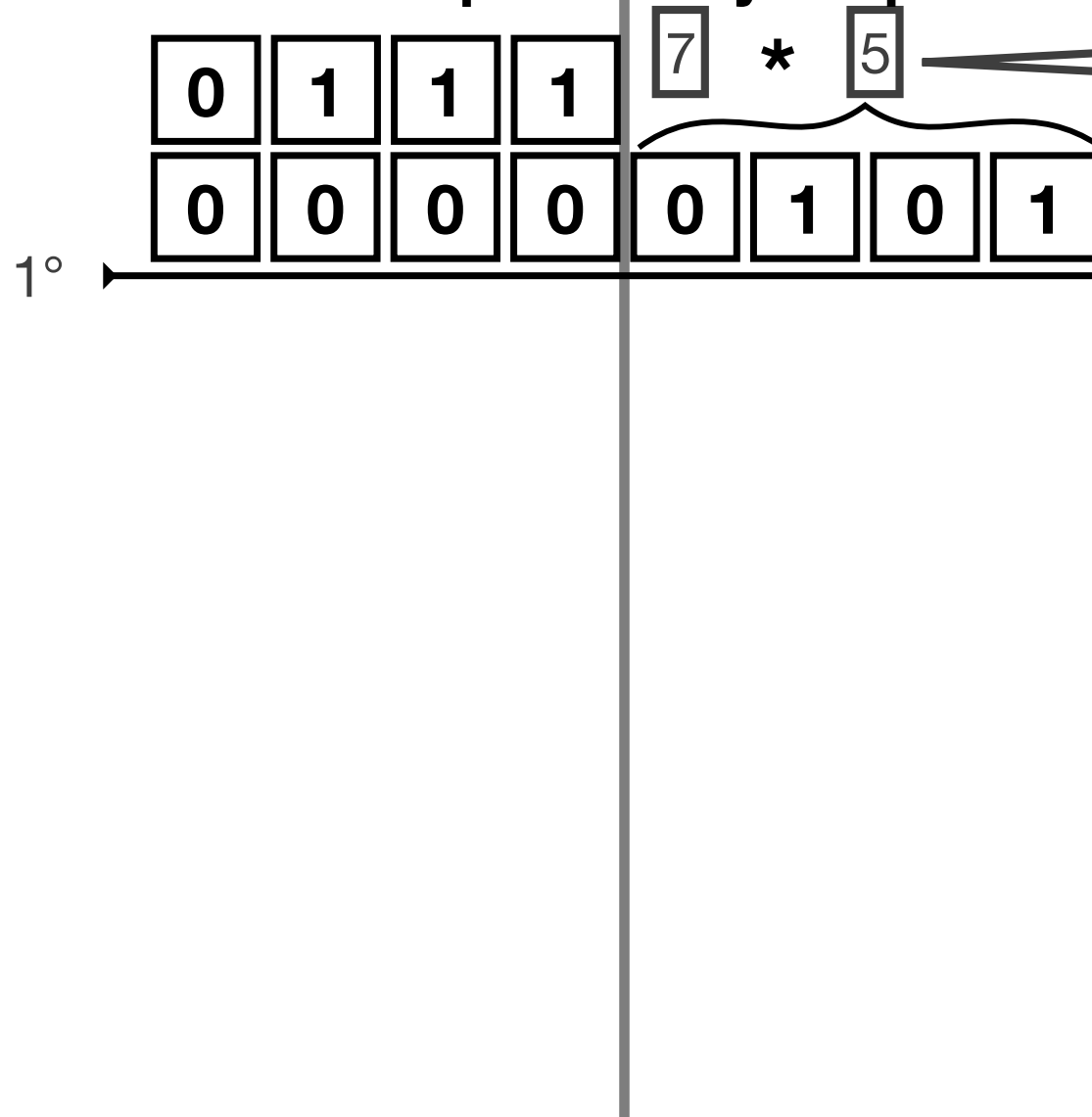
Podemos ir haciendo **sumas parciales del multiplicando**, cuando la posición del **multiplicador** que estamos tratando es 1. Luego en cualquier de los casos desplazamos.

El tamaño del resultado puede necesitar **hasta el doble de tamaño** de representación !

35

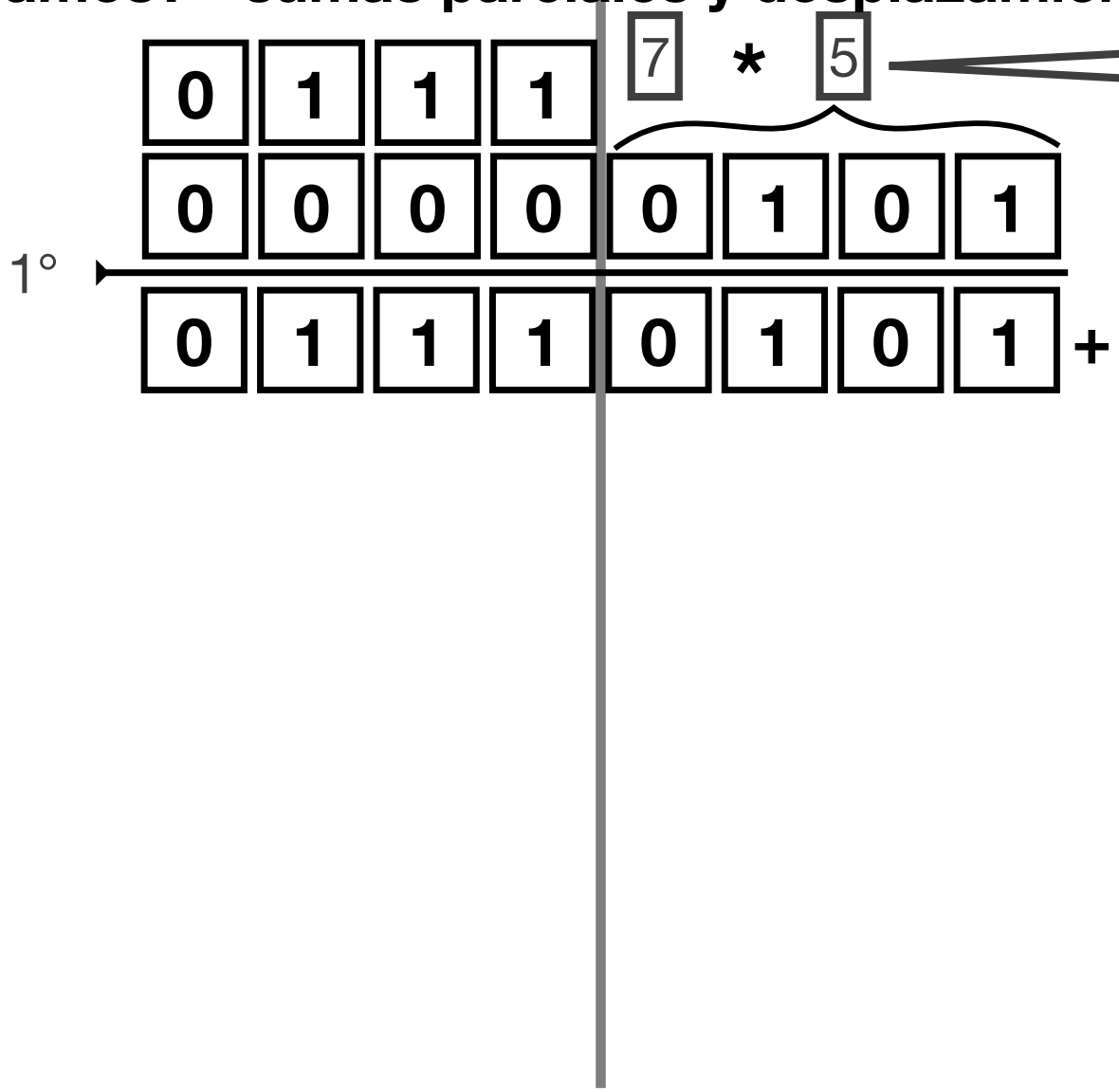


¿Cómo multiplicamos? - sumas parciales y desplazamiento



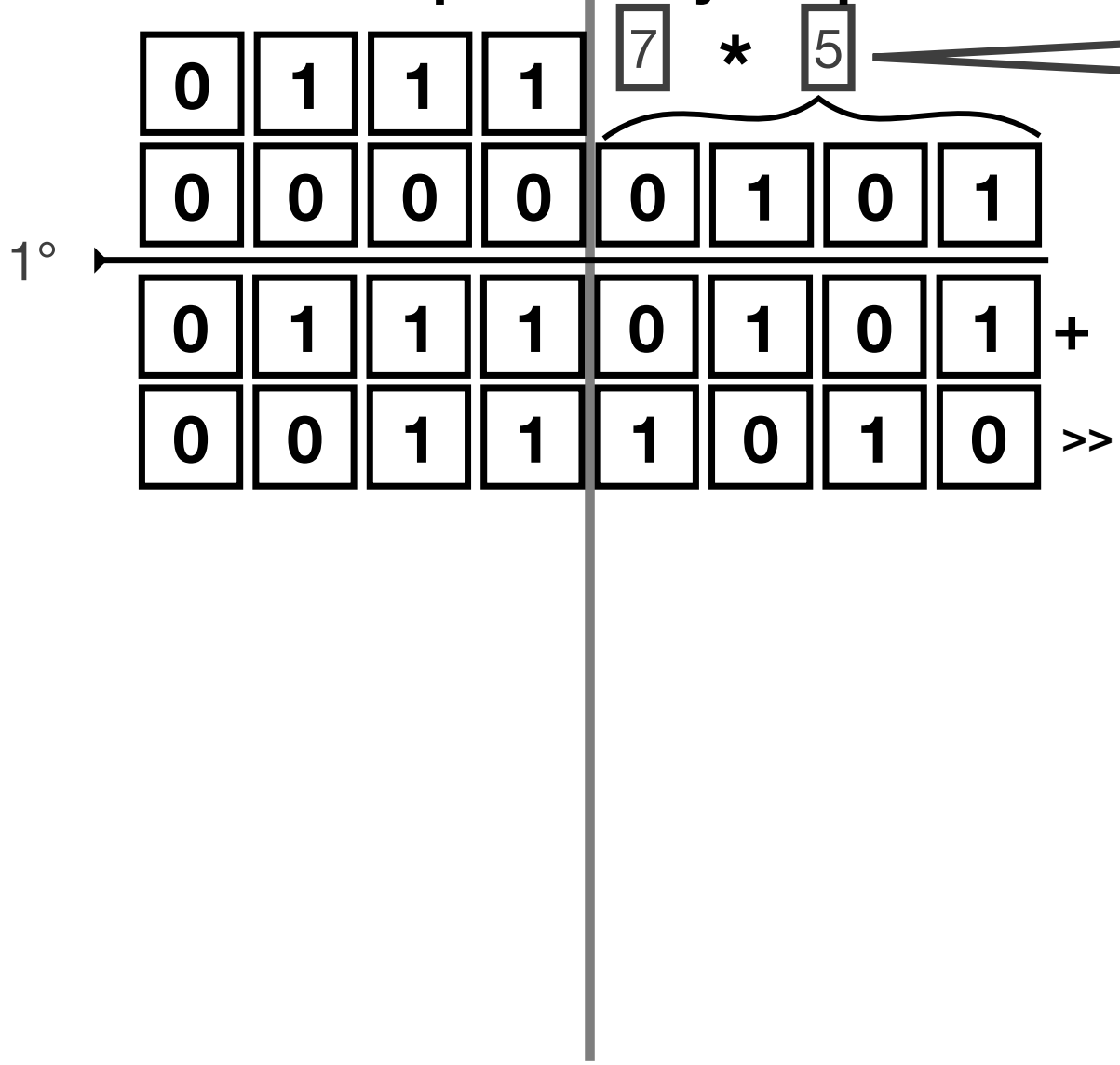
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



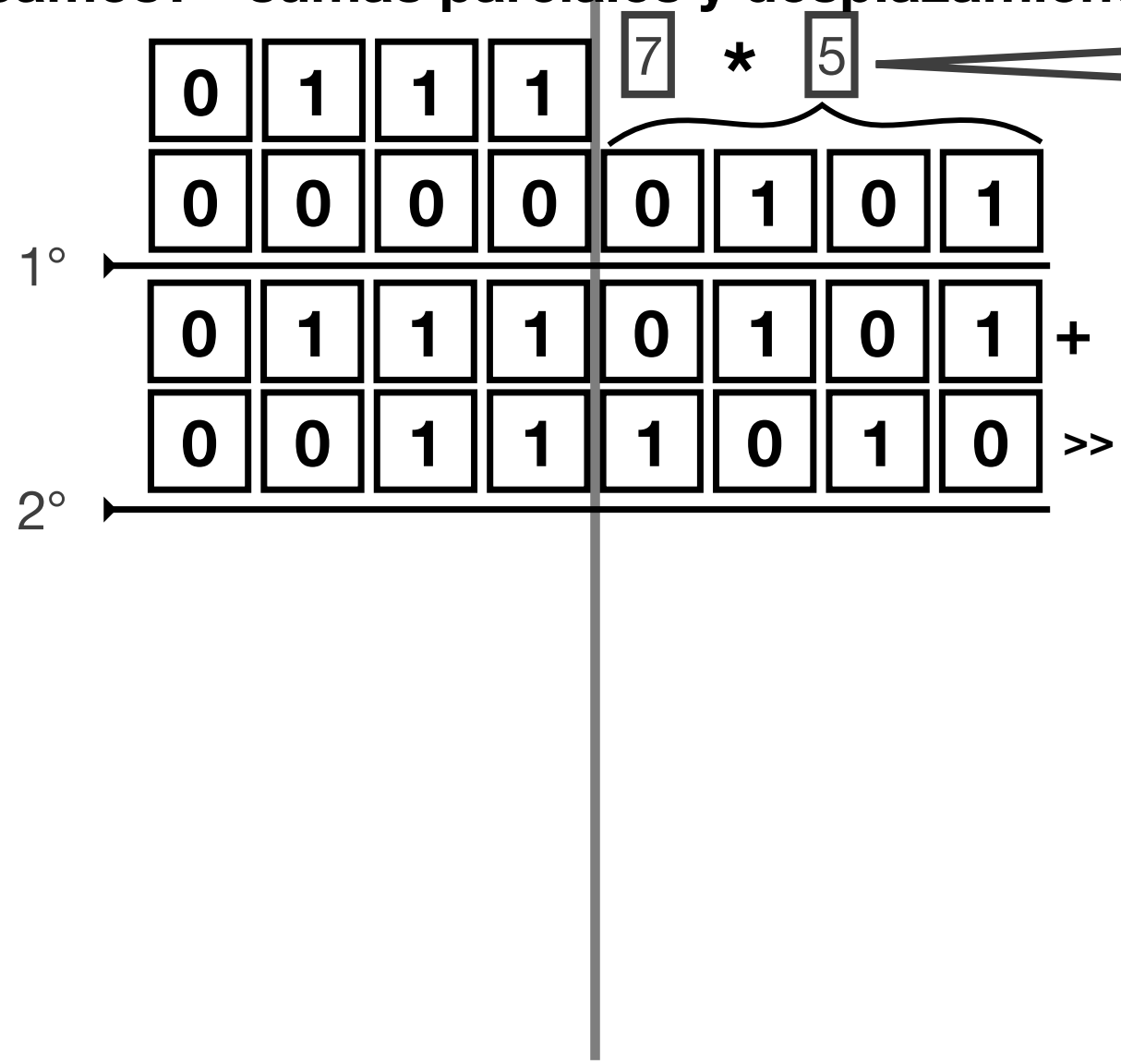
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



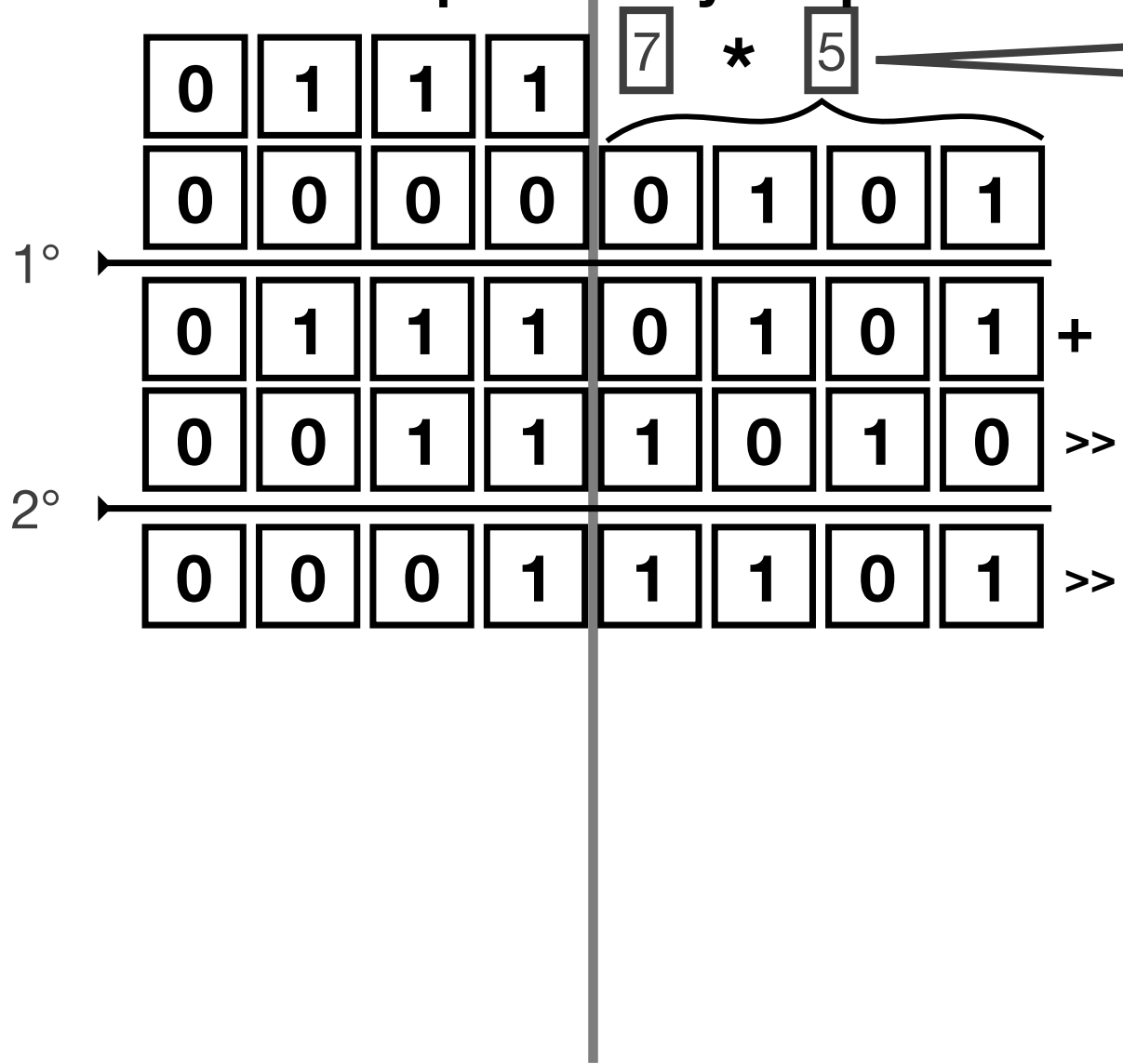
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



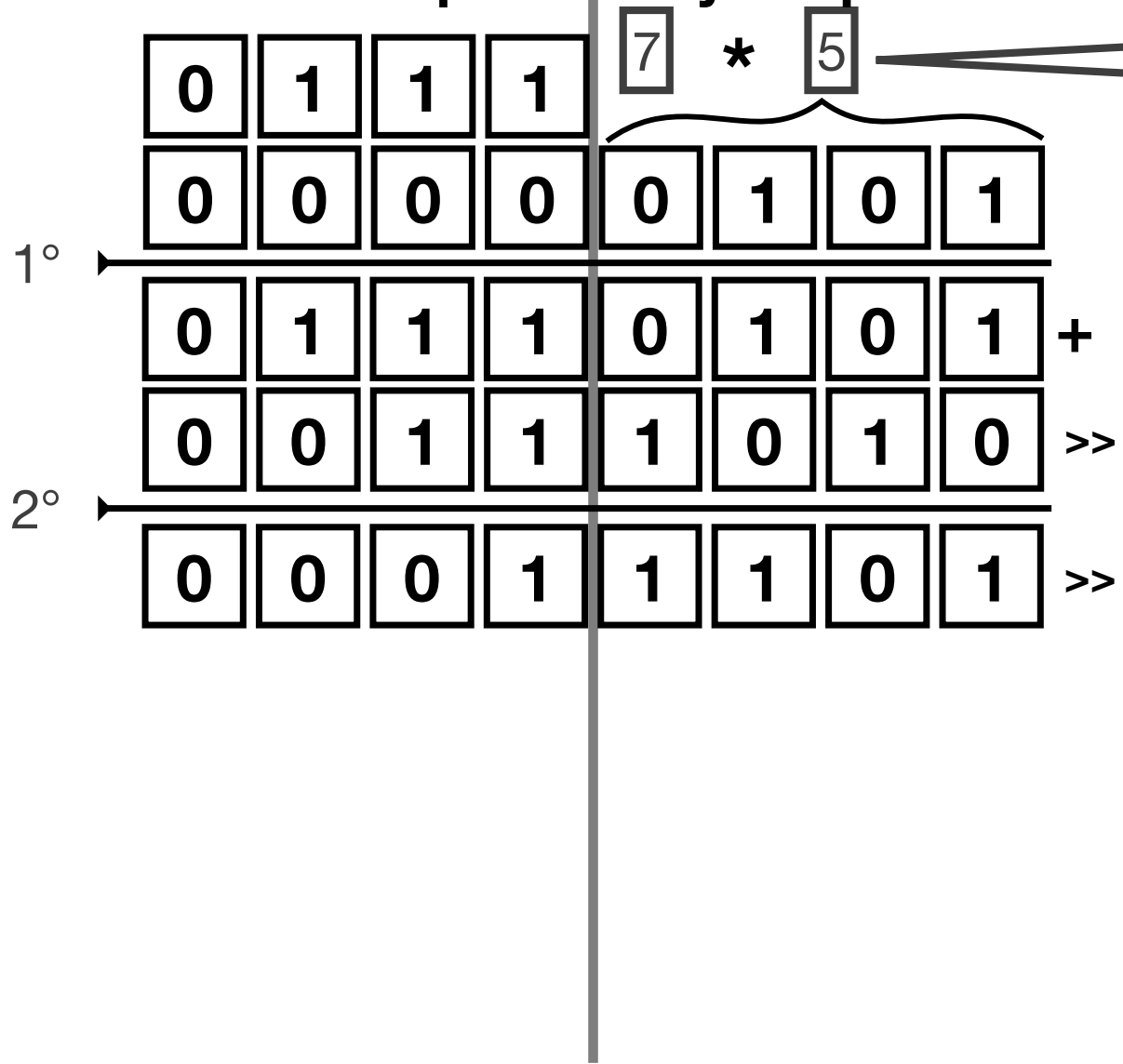
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



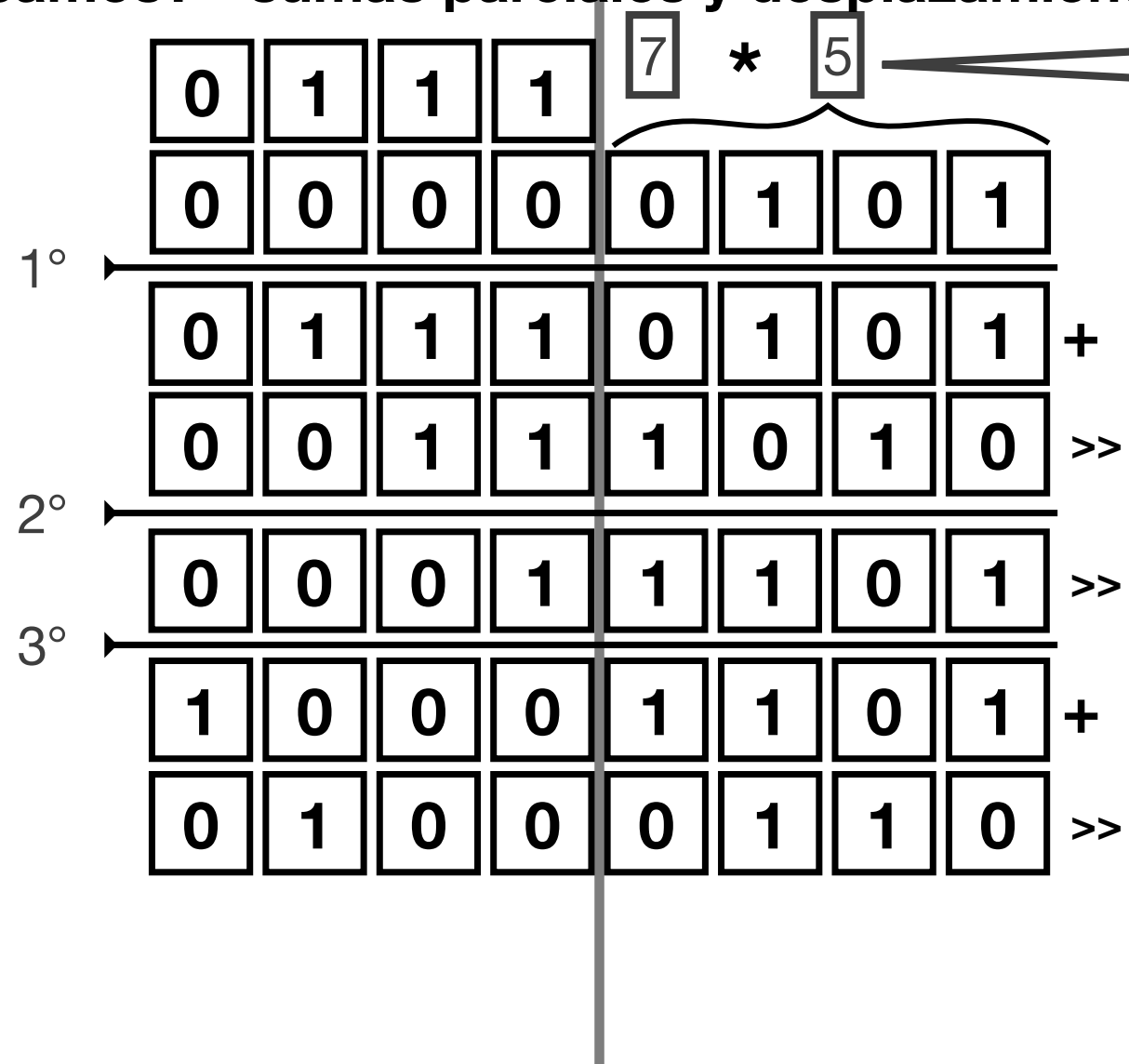
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



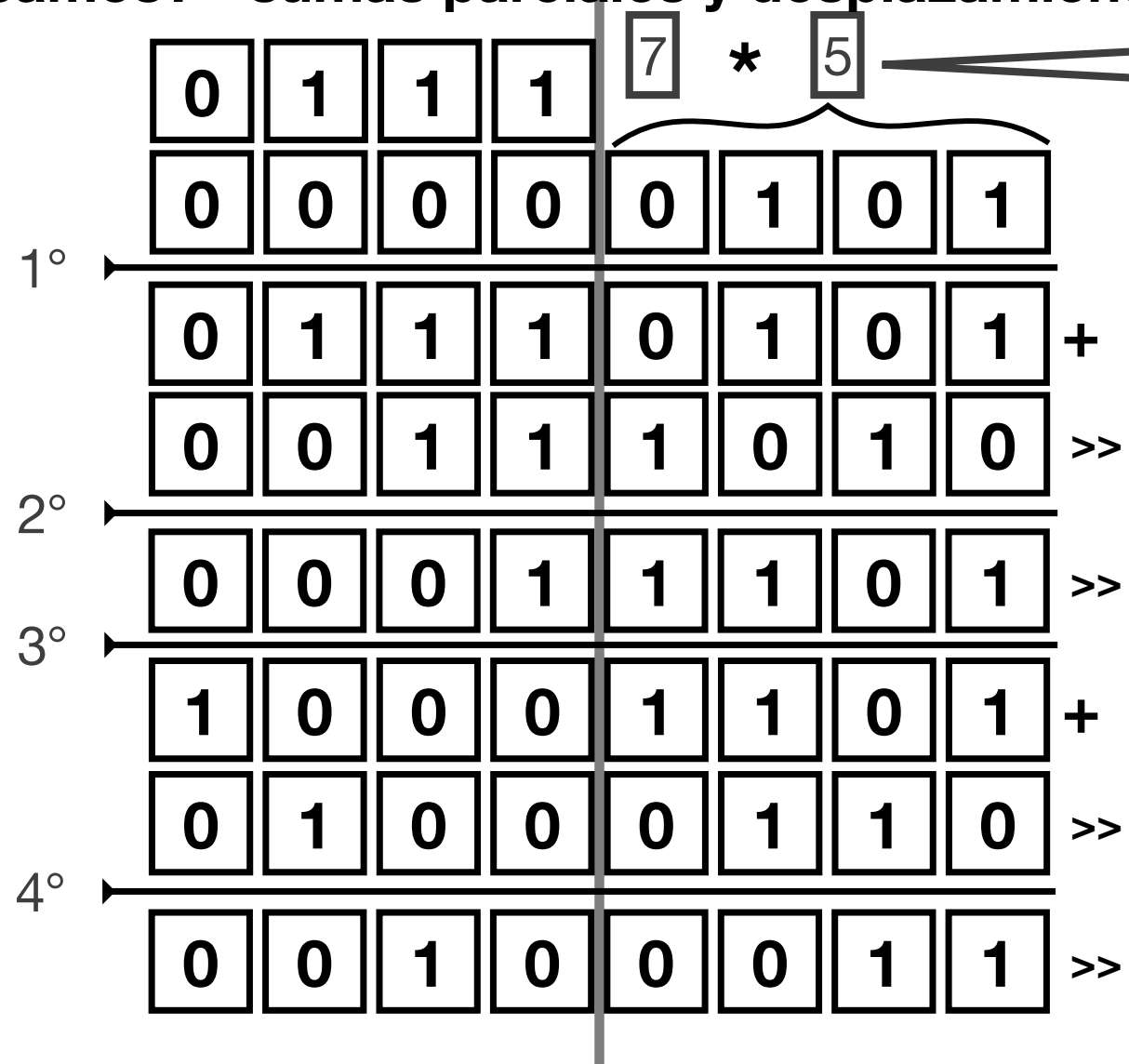
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



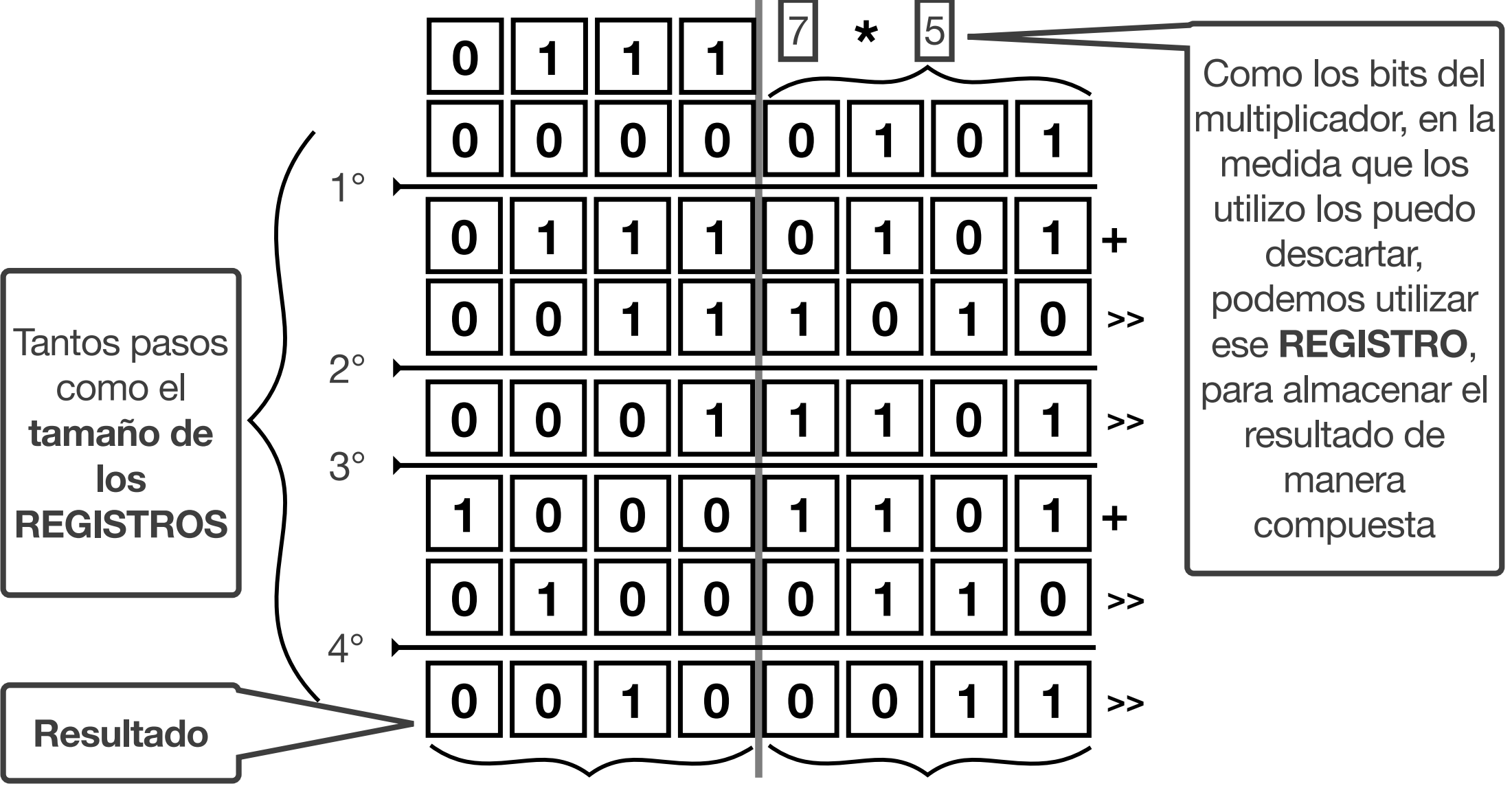
Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



Como los bits del multiplicador, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado de manera compuesta

¿Cómo multiplicamos? - sumas parciales y desplazamiento



Algoritmo de la multiplicación

Multiplicación de enteros general (positivos y negativos):

- El resultado necesita un tamaño de bits equivalente a la suma de los tamaños de los operandos.
- El proceso calcula productos parciales y depende de los siguientes casos:
 - (a)+ **Multiplicando * + Multiplicador** : Se comienza con el producto parcial igual a 0. Se recorre el multiplicador de derecha a izquierda. Si es un 0 se *shiftea con el carry (igual al valor del primer bit del registro) a derecha* el producto parcial. Si es un 1, al producto parcial se le suma el multiplicando y luego se *shiftea con el carry out a derecha* el producto parcial.
 - (b)- **Multiplicando * + Multiplicador** : Igual al proceso anterior.
 - (c)+ **Multiplicando * - Multiplicador** : se complementan ambos números y se procede como en (b)
 - (d)- **Multiplicando * - Multiplicador** : se complementan ambos números y se procede como en (a)

Algoritmo de la multiplicación - Booth

La idea del algoritmo de Booth es la de no operar por casos y la de tratar de realizar la menor cantidad posible de sumas (es más costoso que *shiftear*).

Para ello el multiplicador que indica sumas (cuando es 1) o shirteos (cuando es 0) tiene un tratamiento especial siguiendo la siguiente fundamentación: cuando tenemos una secuencia de sumas (unos), su significado es equivalente a considerar la próxima posición a la más significativa menos la menos significativa.

Ejemplo: 15

0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	0	0	0	-1
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Ejemplo: 60

0	0	0	0	1	1	1	1	0	0
0	0	0	1	0	0	0	-1	0	0
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Ejemplo: 462

0	1	1	1	0	0	1	1	1	0
1	0	0	-1	0	1	0	0	-1	0
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Algoritmo de la multiplicación - Booth

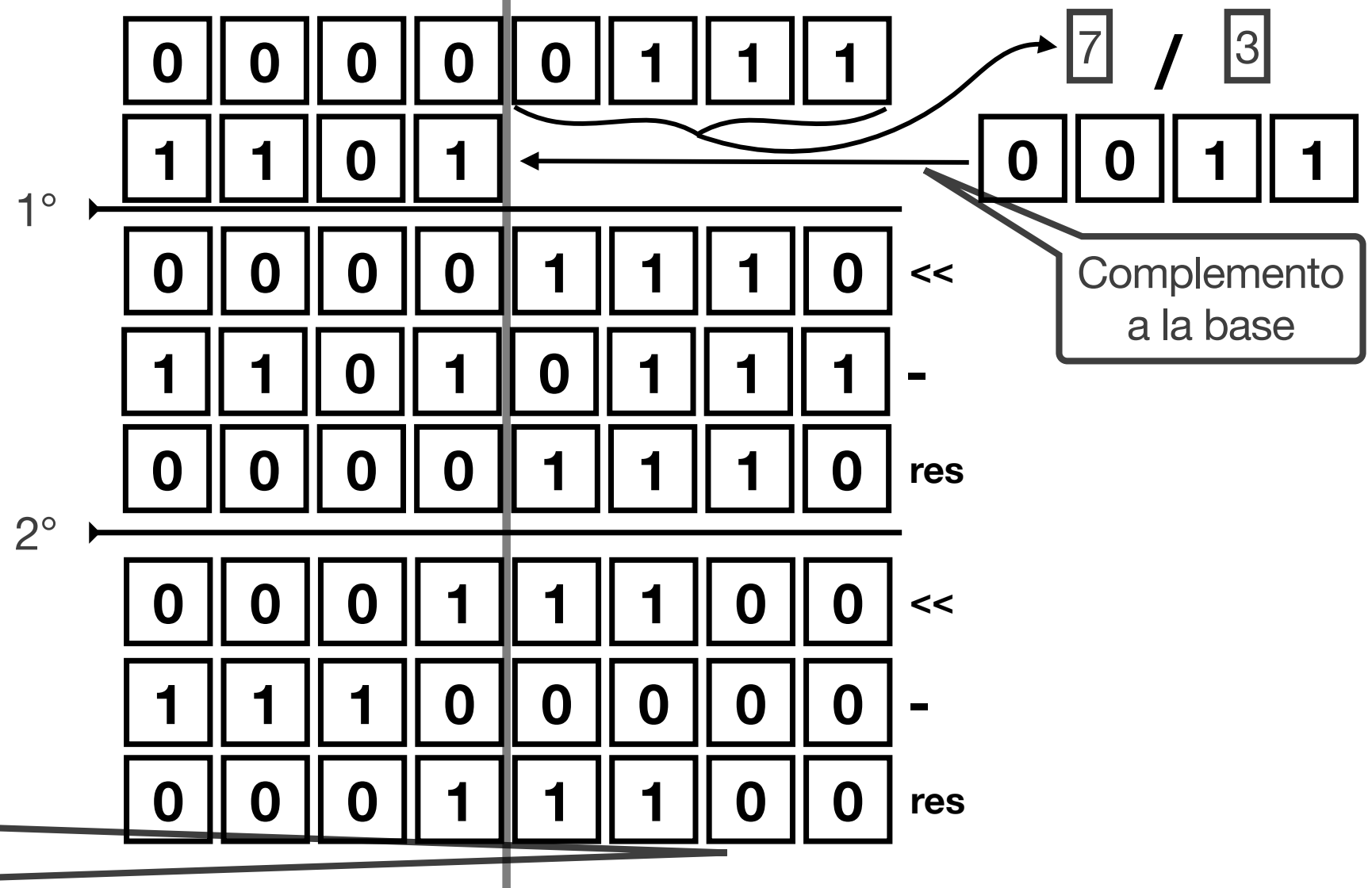
El resultado necesita un tamaño de bits equivalente a la suma del tamaño de los operandos.

El proceso calcula productos parciales de manera similar que el algoritmo general pero en vez de mirar el último bit del **multiplicador** tiene en cuenta **los últimos 2 bits** y de acuerdo a los siguientes casos opera sobre el producto parcial:

0	0	→	Se shiftea a derecha
0	1	→	Se suma el multiplicando
1	0	→	Se suma el complemento a la base (negativo) del multiplicando
1	1	→	Se shiftea a derecha respetando la extensión de signo

¿Cómo dividimos? - restas sucesivas y desplazamiento

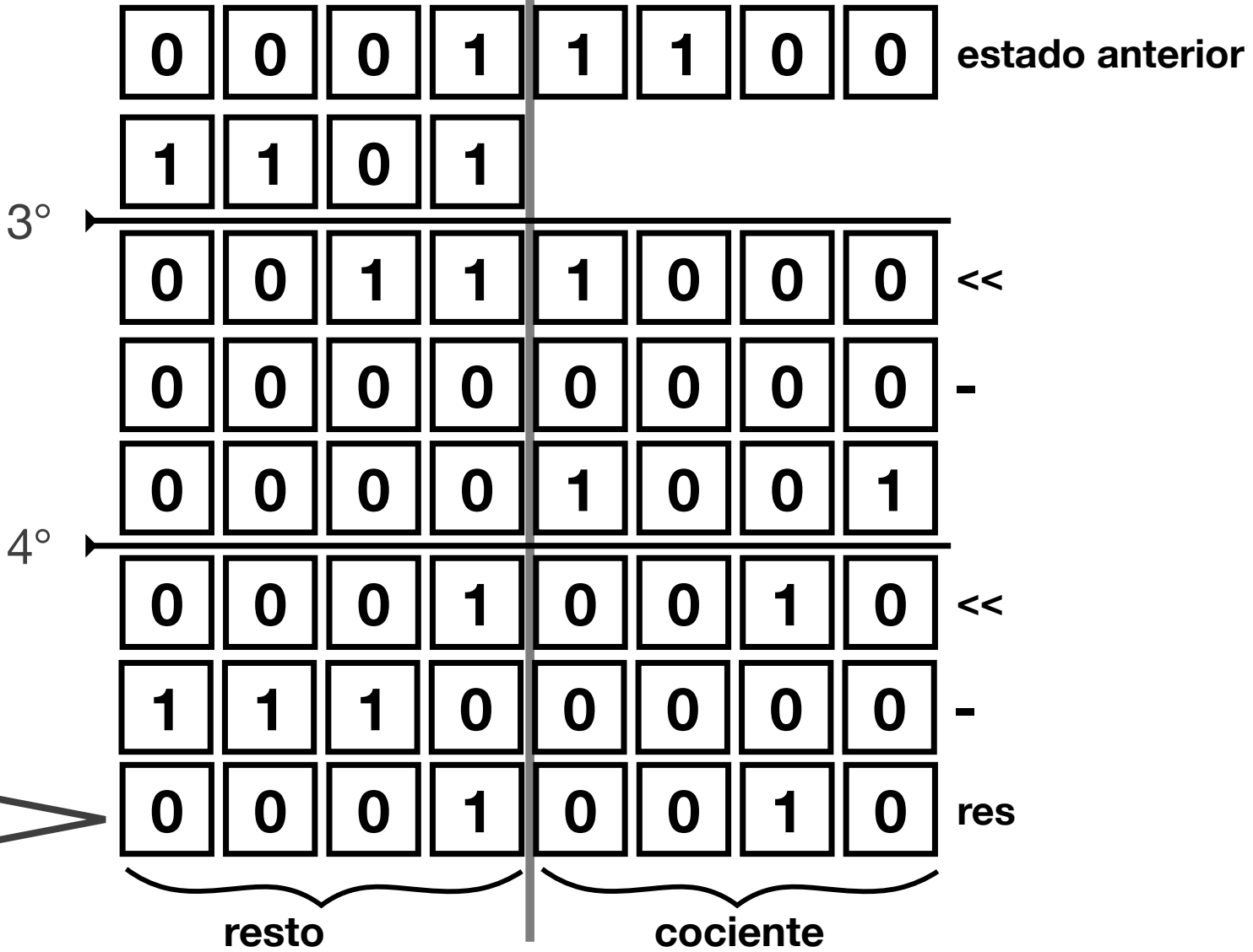
Como el los bits del **dividendo**, en la medida que los utilizo los puedo descartar, podemos utilizar ese **REGISTRO**, para almacenar el resultado (cociente) de la división



¿Cómo dividimos? - restas sucesivas y desplazamiento

Tantos pasos
como el
tamaño de
los
REGISTROS

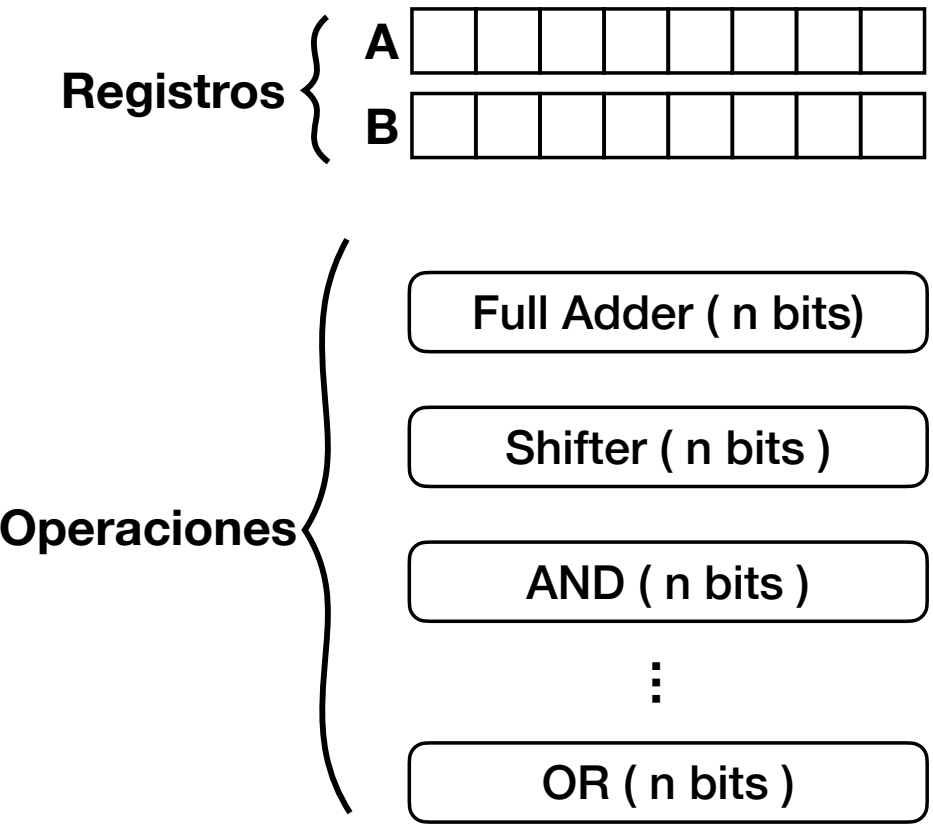
Resultados



Algoritmo de la división (con restauración)

- (1) *Shiftear a izquierda* una vez los registro correspondientes al **Dividendo** (Q) con **0** y el **registro auxiliar** (R) con el bit más significativo de Q.
- (2) Dejar en el registro auxiliar el resultado de restarle al mismo el **Divisor** (S). Notar que se suma el Comp.Base de (S)
- (3) Si el signo (*bit más* significativo del **registro auxiliar** R) es 1 (negativo), cambiar el *bit menos* significativo de **Dividendo** (Q_0) a 0 y restaurar el **registro auxiliar** sumando el **Divisor** (S), sino cambiar el *bit menos* significativo de **Dividendo** (Q_0) a 1.

Qué tenemos hasta ahora



Automatizar una sumatoria de n números

¿ Dónde guardamos
“la tarea”?

17
4
23
9
3
46
13
18
64
11

Automatizar una sumatoria de n números

CADA CELDA ALMACENA 8 BITS = 1 BYTE

¿ Dónde guardamos
“la tarea”?

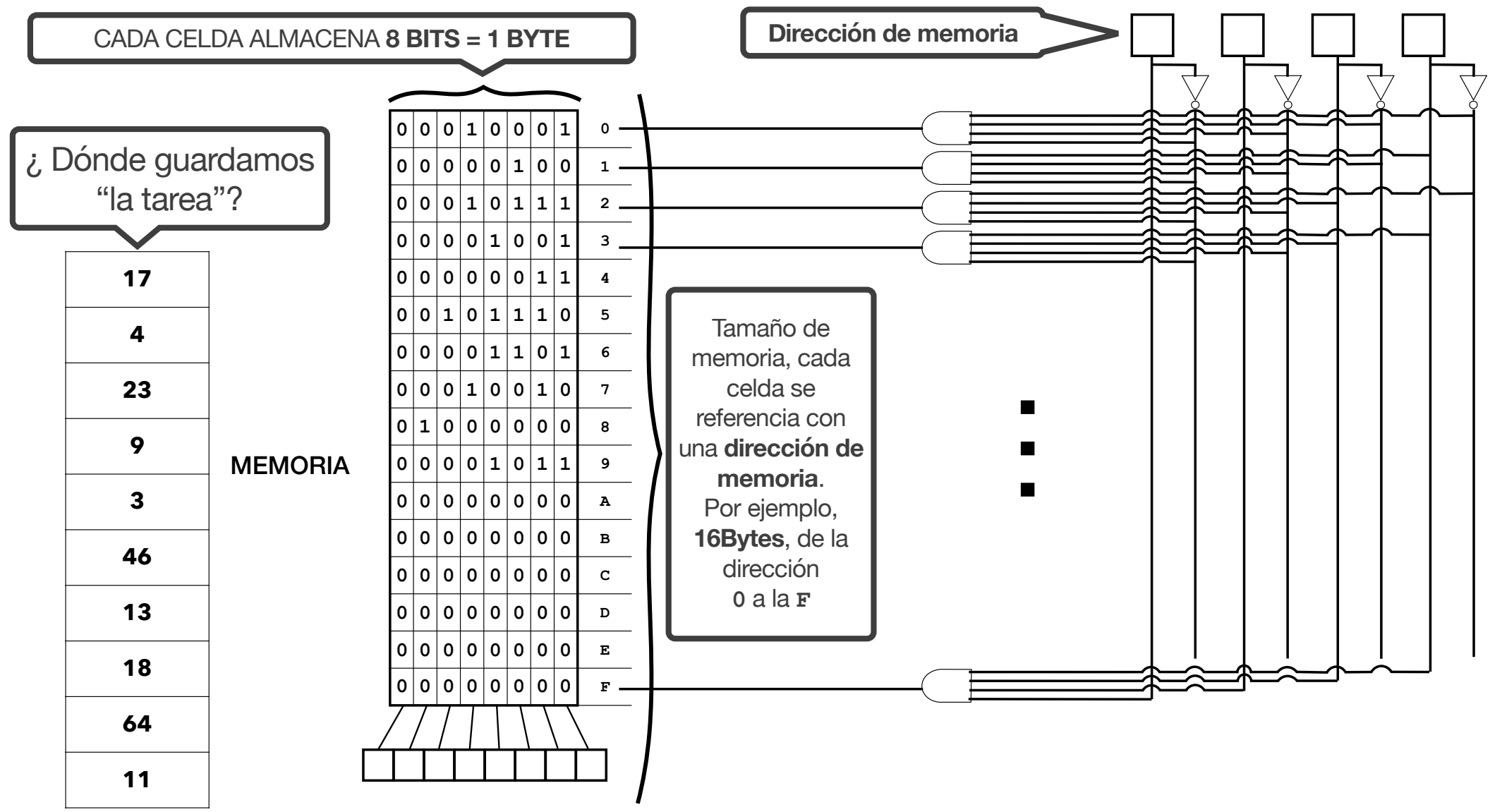
17
4
23
9
3
46
13
18
64
11

MEMORIA

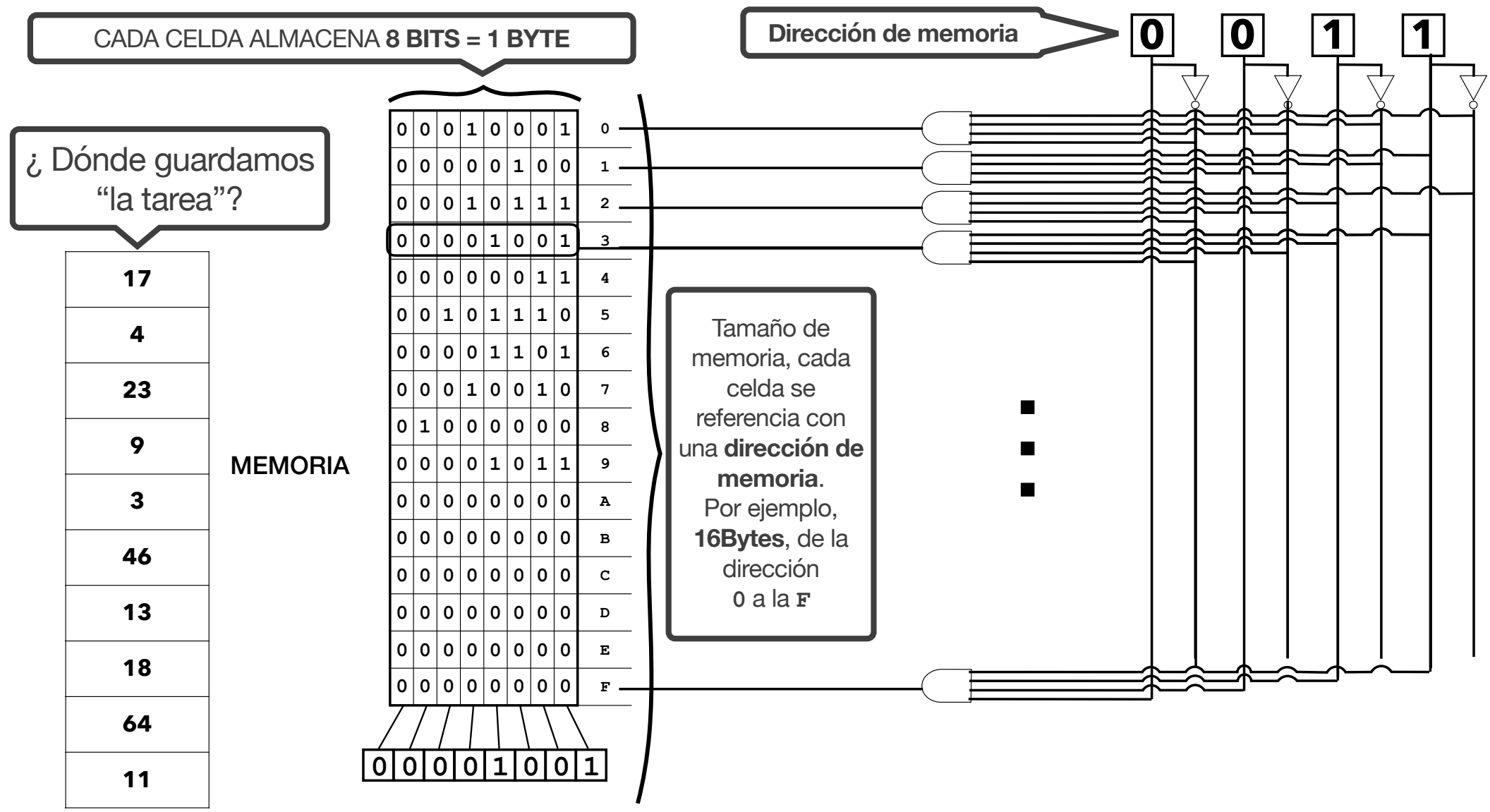
0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	0	0	0	1	0	0	1	3
0	0	0	0	0	0	1	1	4
0	0	1	0	1	1	1	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	1	0	0	0	0	0	0	8
0	0	0	0	1	0	1	1	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

Tamaño de memoria, cada celda se referencia con una **dirección de memoria**.
Por ejemplo, **16Bytes**, de la dirección 0 a la F

Automatizar una sumatoria de n números



Automatizar una sumatoria de n números



Automatizar una sumatoria de n números

CADA CELDA ALMACENA 8 BITS = 1 BYTE

¿ Dónde guardamos “la tarea”?

17
4
23
9
3
46
13
18
64
11

MEMORIA

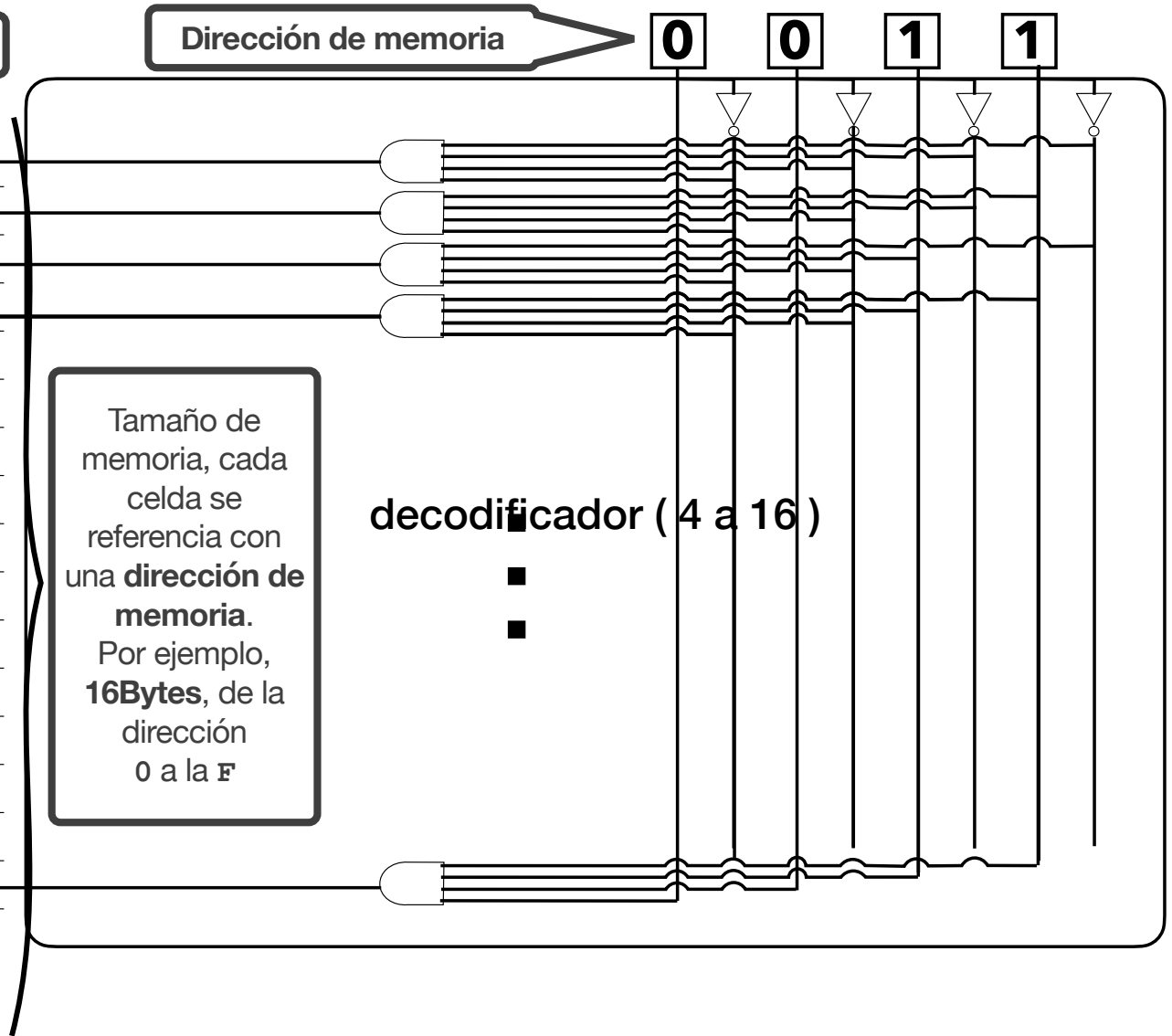
0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	0
0	0	0	0	1	0	0	1	1
0	0	0	0	0	0	1	1	0
0	0	1	0	1	1	1	0	0
0	0	0	0	1	1	0	1	0
0	0	0	1	0	0	1	0	0
0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

Dirección de memoria

0 0 1 1

Tamaño de memoria, cada celda se referencia con una **dirección de memoria**.
Por ejemplo, **16Bytes**, de la dirección 0 a la F

decodificador (4 a 16)



Automatizar una sumatoria de *n* números

0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	0	0	0	1	0	0	1	3
0	0	0	0	0	0	1	1	4
0	0	1	0	1	1	1	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	1	0	0	0	0	0	0	8
0	0	0	0	1	0	1	1	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

¿Cómo implementamos la noción de etapa o paso?, ¿cómo podemos hacer un contador que recorra las direcciones?

A

--	--	--	--	--	--	--	--

B

--	--	--	--	--	--	--	--

Full Adder (n bits)

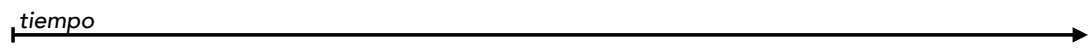
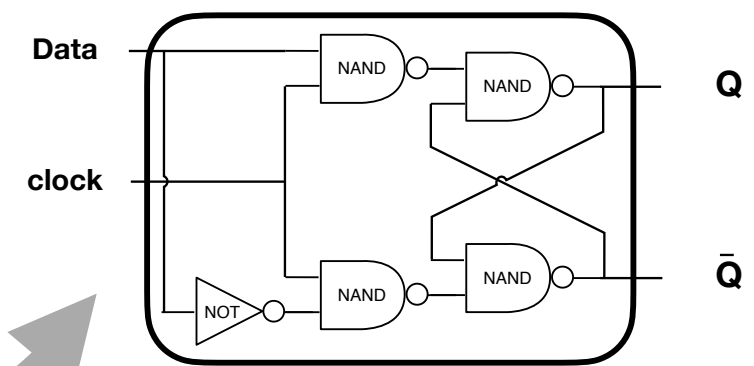
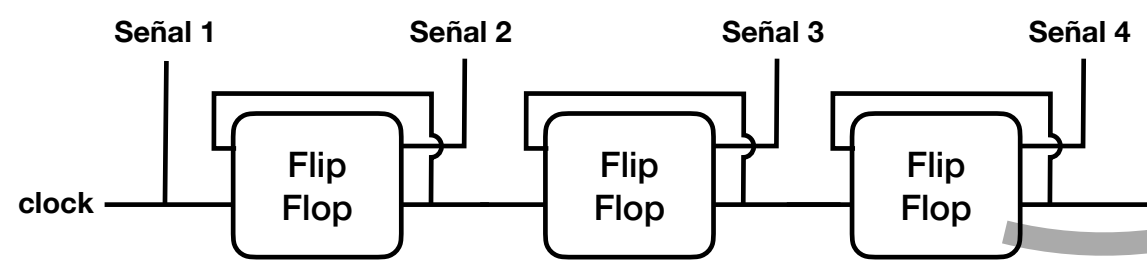
Dirección

--	--	--	--	--	--	--	--

Dato

--	--	--	--	--	--	--	--

Contador



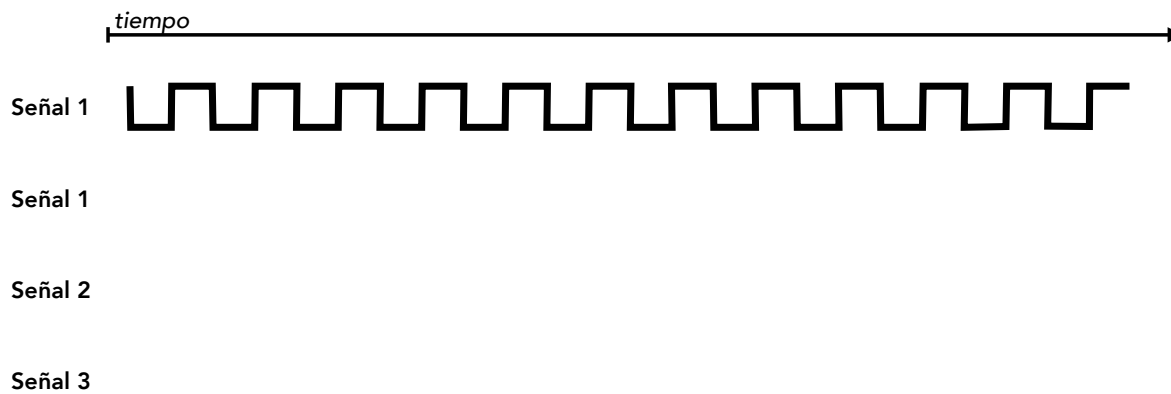
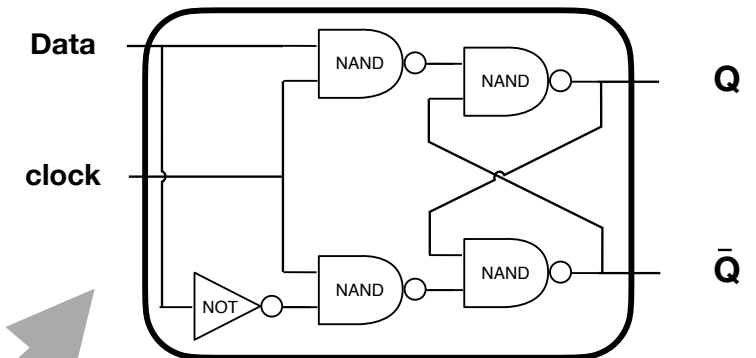
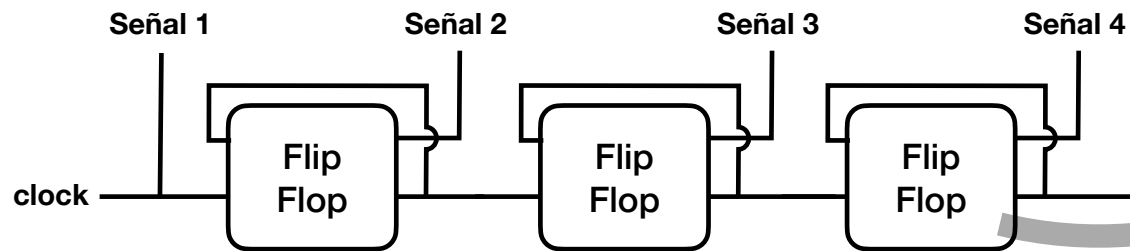
Señal 1

Señal 1

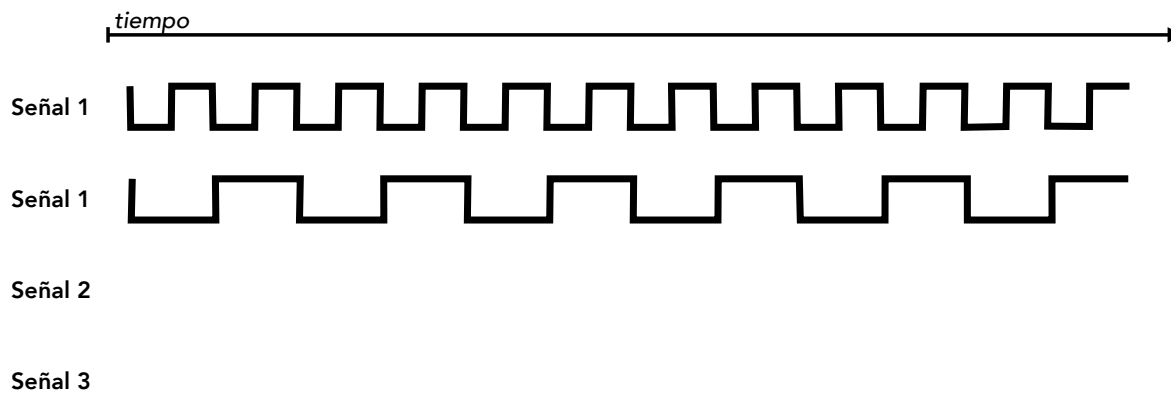
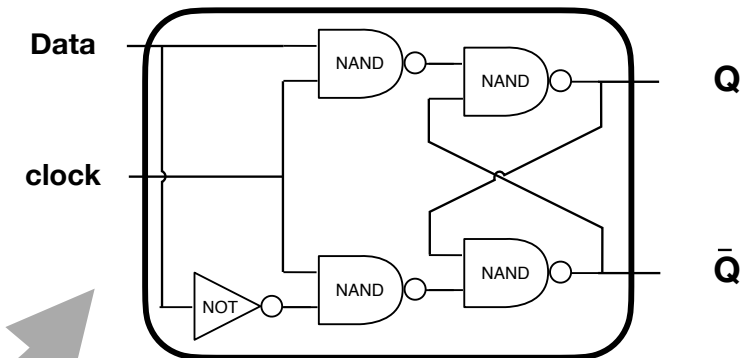
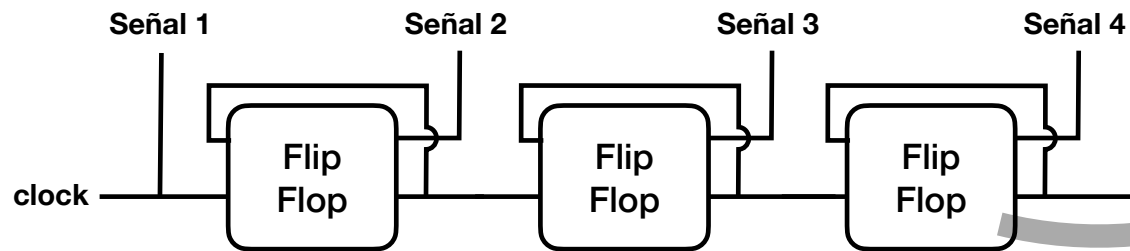
Señal 2

Señal 3

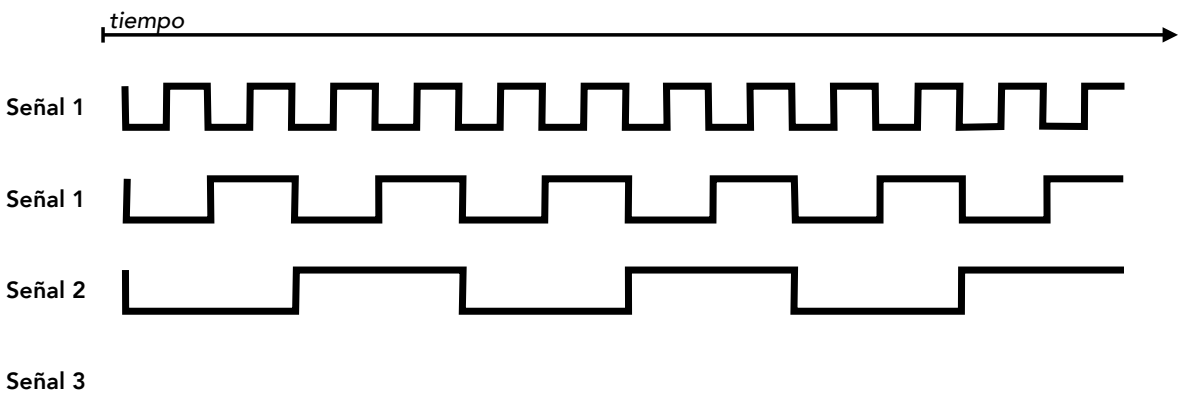
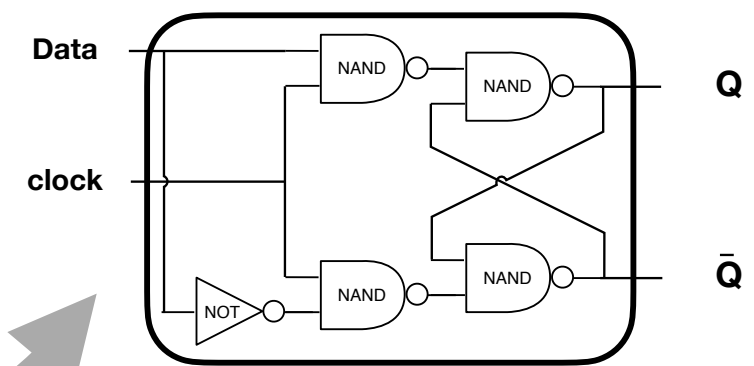
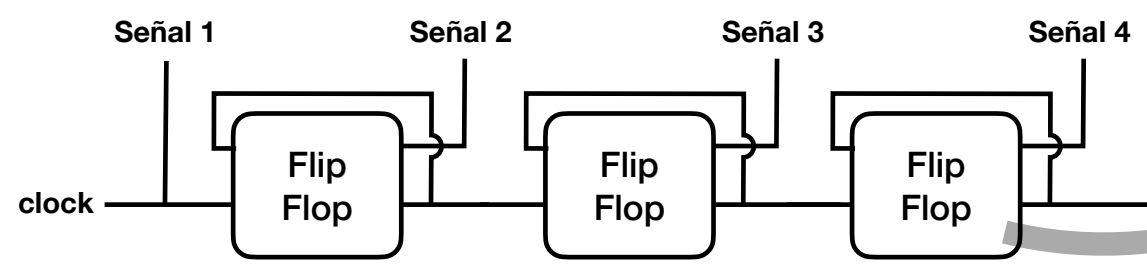
Contador



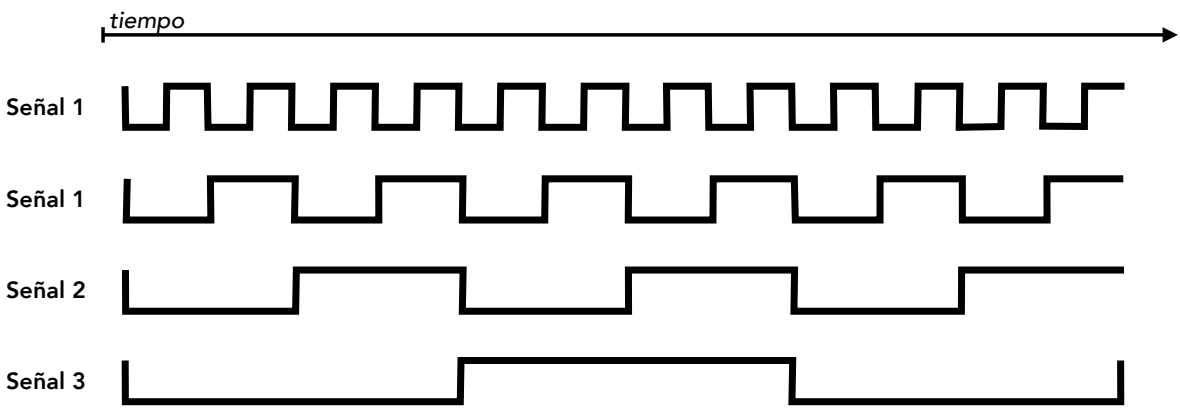
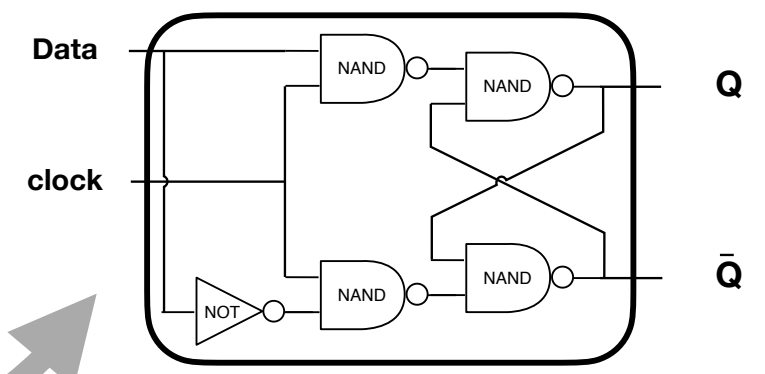
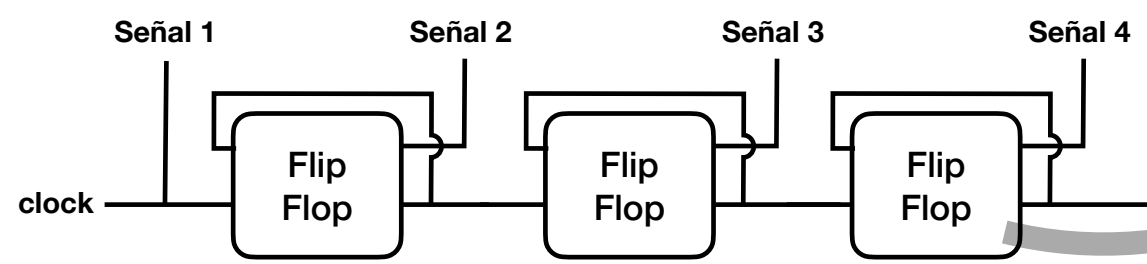
Contador



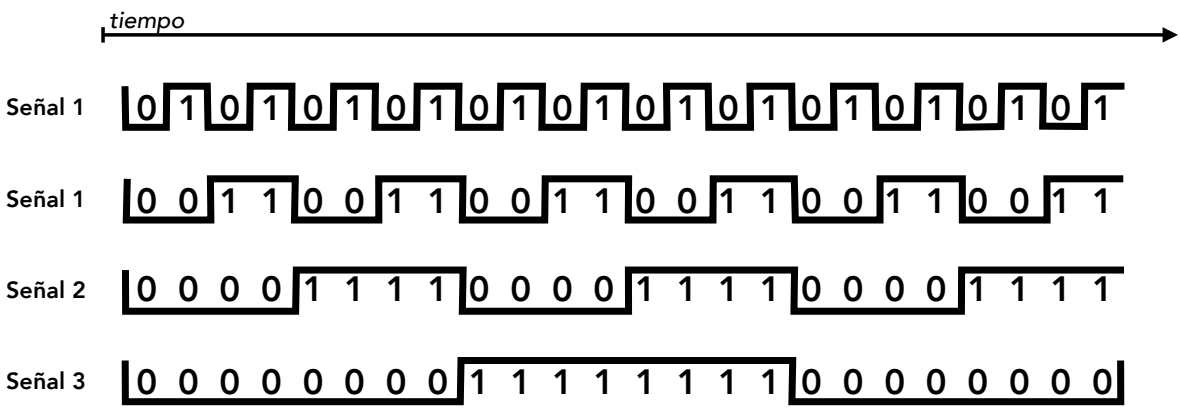
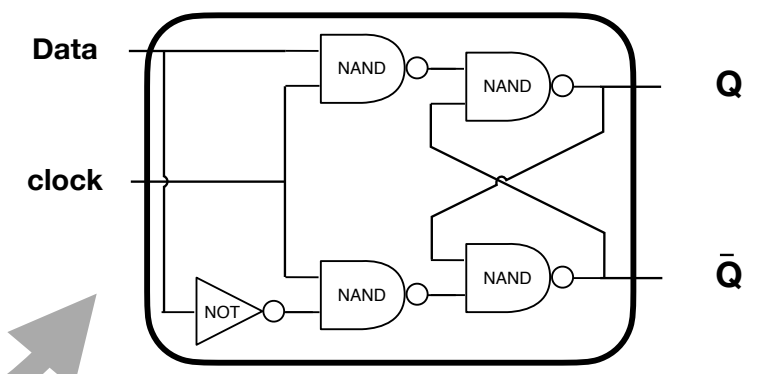
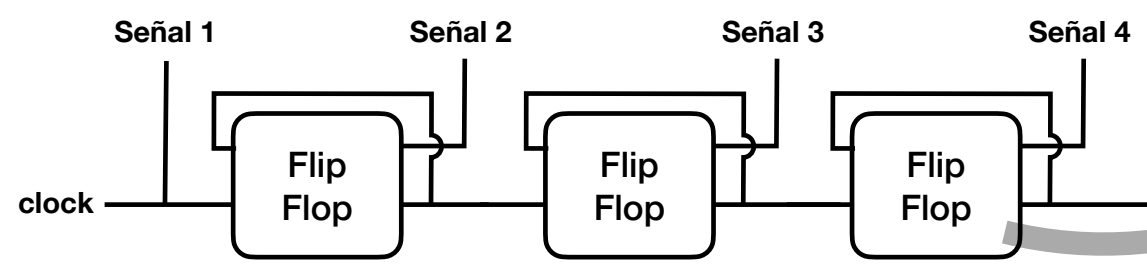
Contador



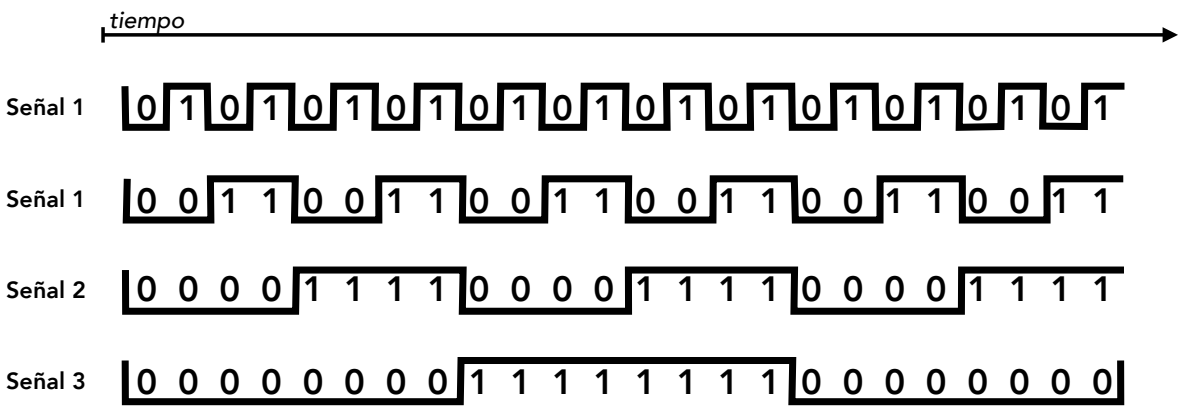
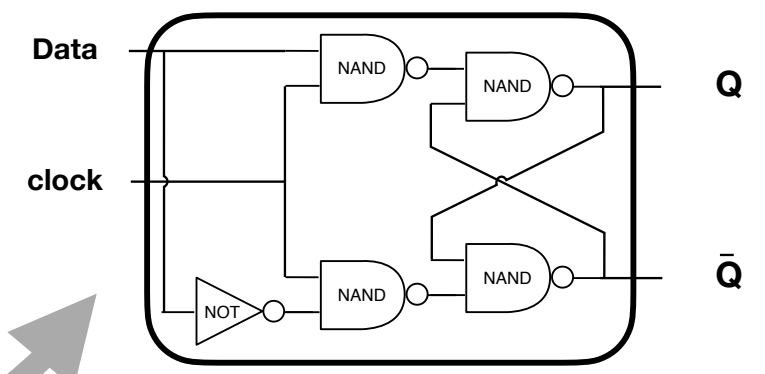
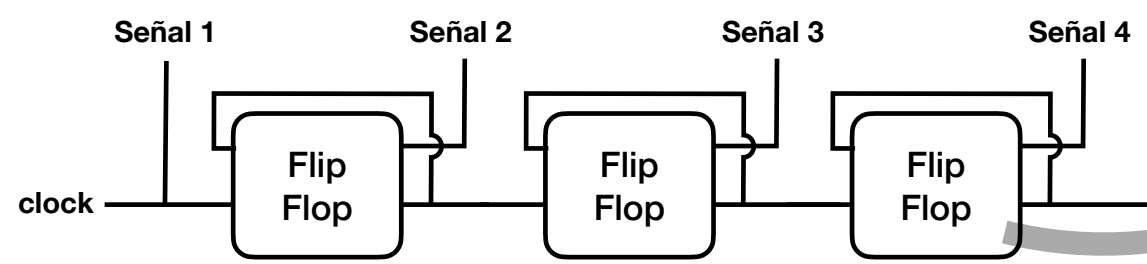
Contador



Contador

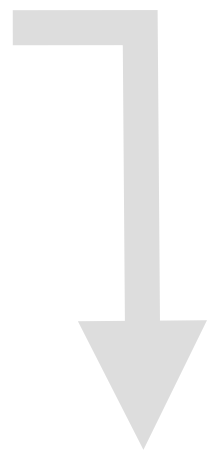
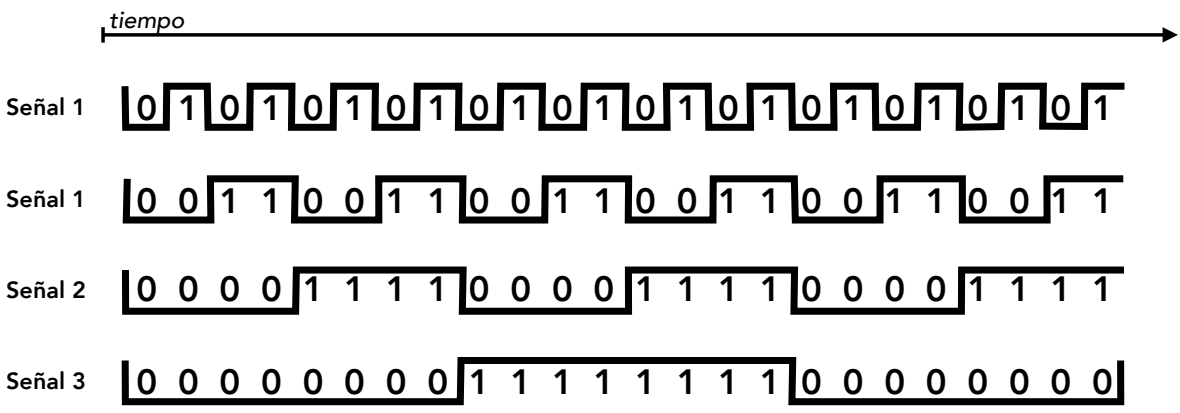
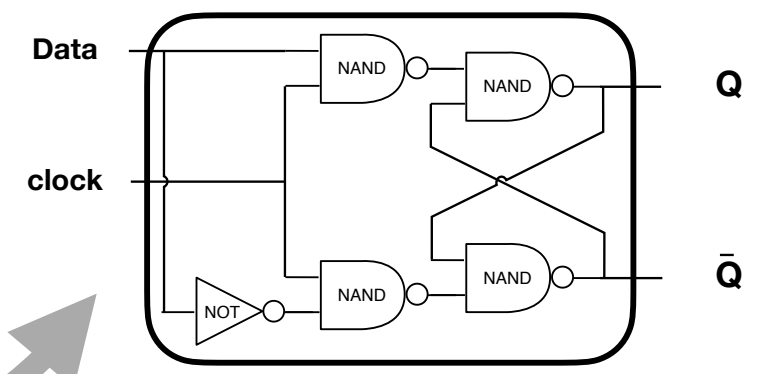
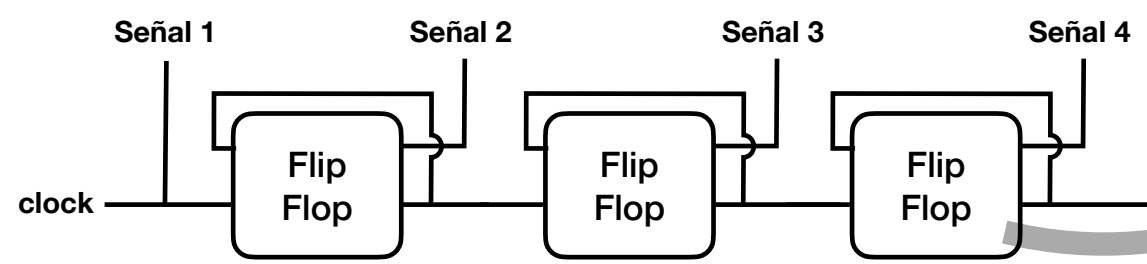


Contador



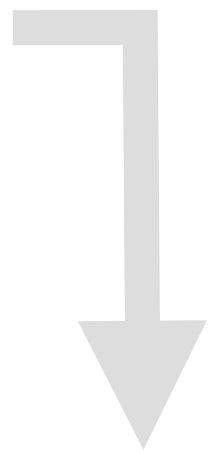
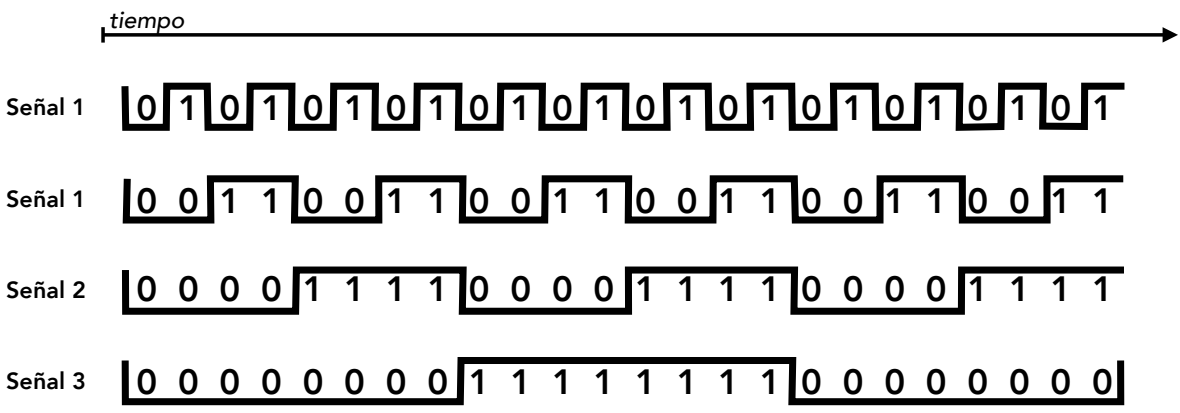
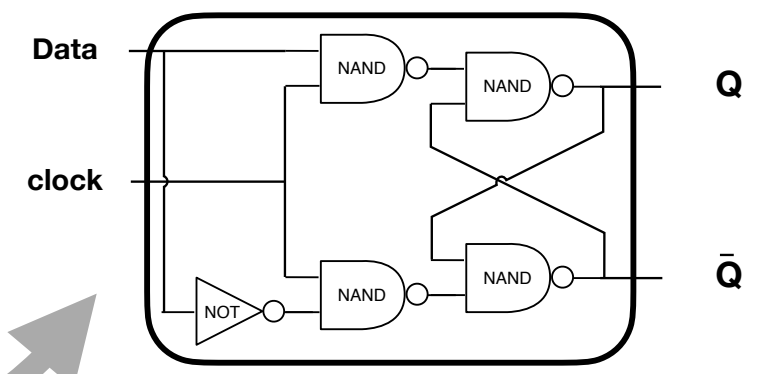
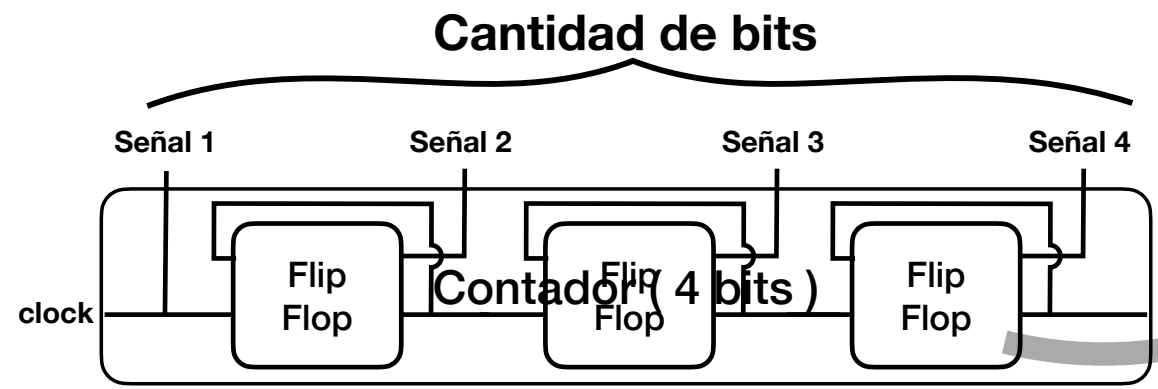
tiempo	Señal 1	Señal 2	Señal 3	Señal 4
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Contador



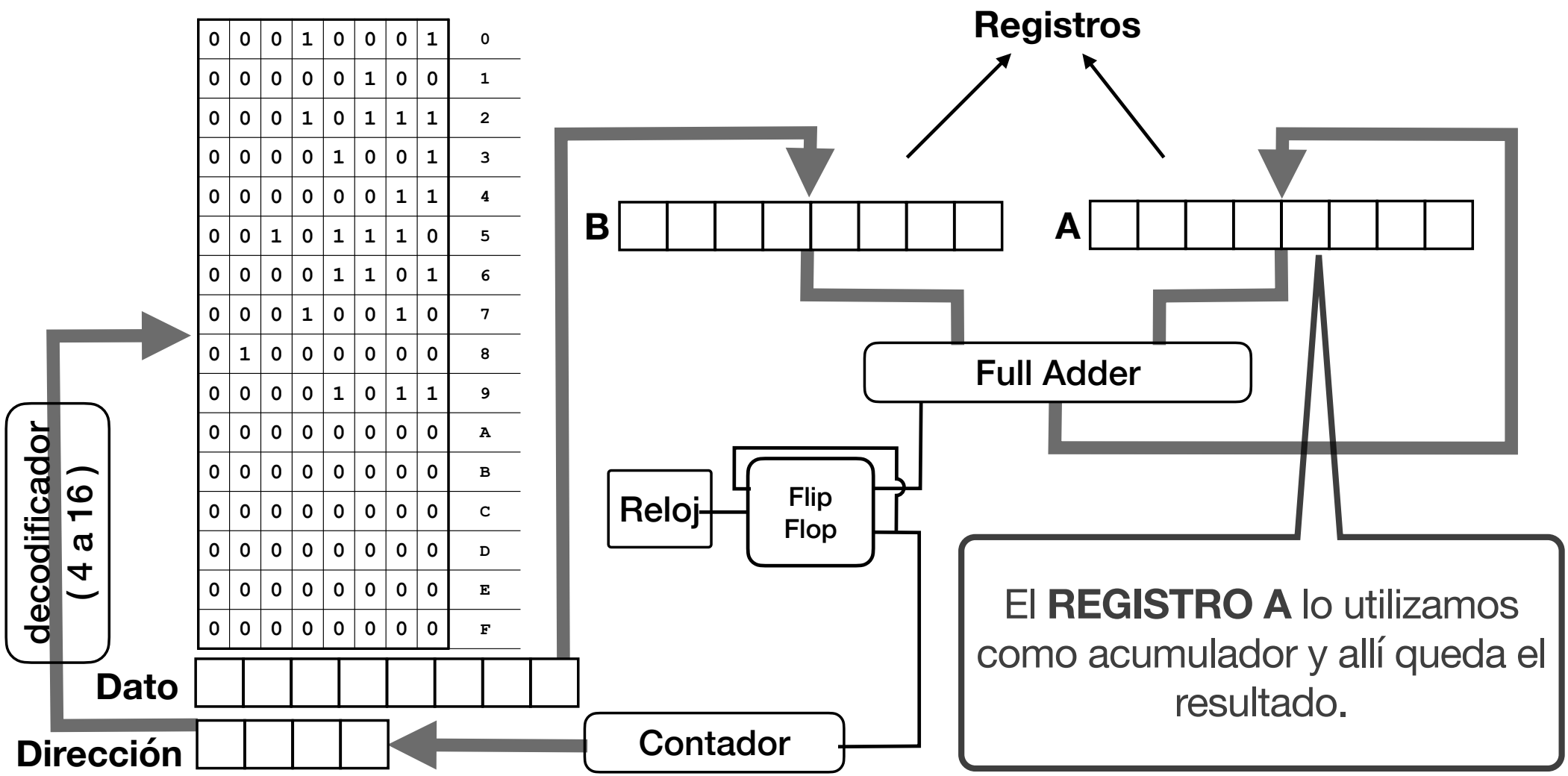
0	0	0	0	→	0
0	0	0	1	→	1
0	0	1	0	→	2
0	0	1	1	→	3
0	1	0	0	→	4
0	1	0	1	→	5
0	1	1	0	→	6
0	1	1	1	→	7
1	0	0	0	→	8
1	0	0	1	→	9
1	0	1	0	→	10
1	0	1	1	→	11
1	1	0	0	→	12
1	1	0	1	→	13
1	1	1	0	→	14
1	1	1	1	→	15

Contador



0	0	0	0	→	0
0	0	0	1	→	1
0	0	1	0	→	2
0	0	1	1	→	3
0	1	0	0	→	4
0	1	0	1	→	5
0	1	1	0	→	6
0	1	1	1	→	7
1	0	0	0	→	8
1	0	0	1	→	9
1	0	1	0	→	10
1	0	1	1	→	11
1	1	0	0	→	12
1	1	0	1	→	13
1	1	1	0	→	14
1	1	1	1	→	15

Automatizar una sumatoria de n números

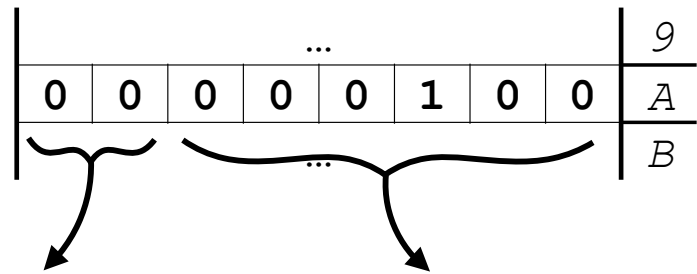


Automatizar una secuencia de operaciones: sumas, restas o *shift*

Debemos indicar qué operación y el correspondiente valor (si corresponde)

Programa

+ 17
- 4
+ 23
- 9
<<
<<
+ 13
+ 18
+ 11
- 64

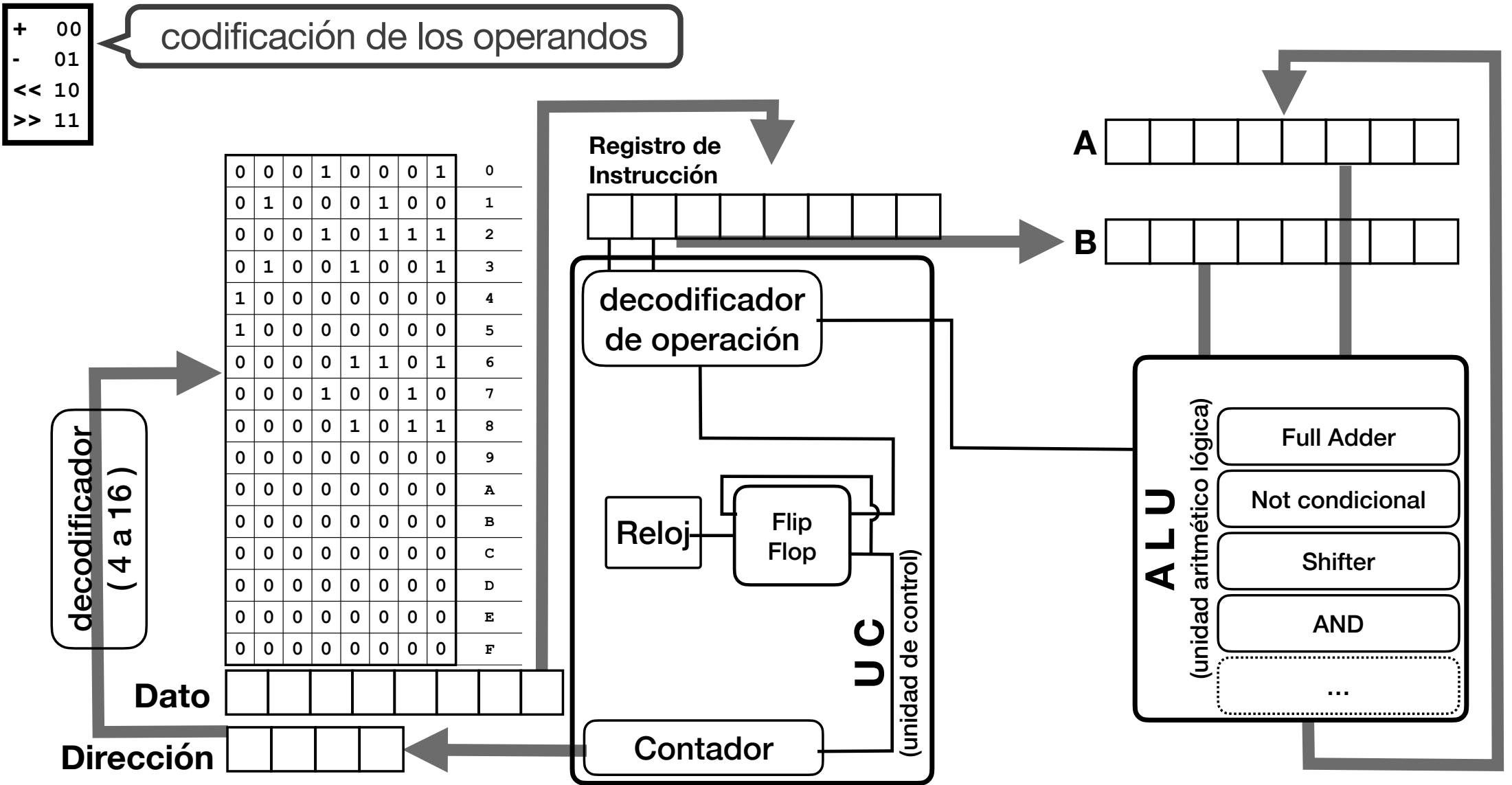


OP OPERANDO

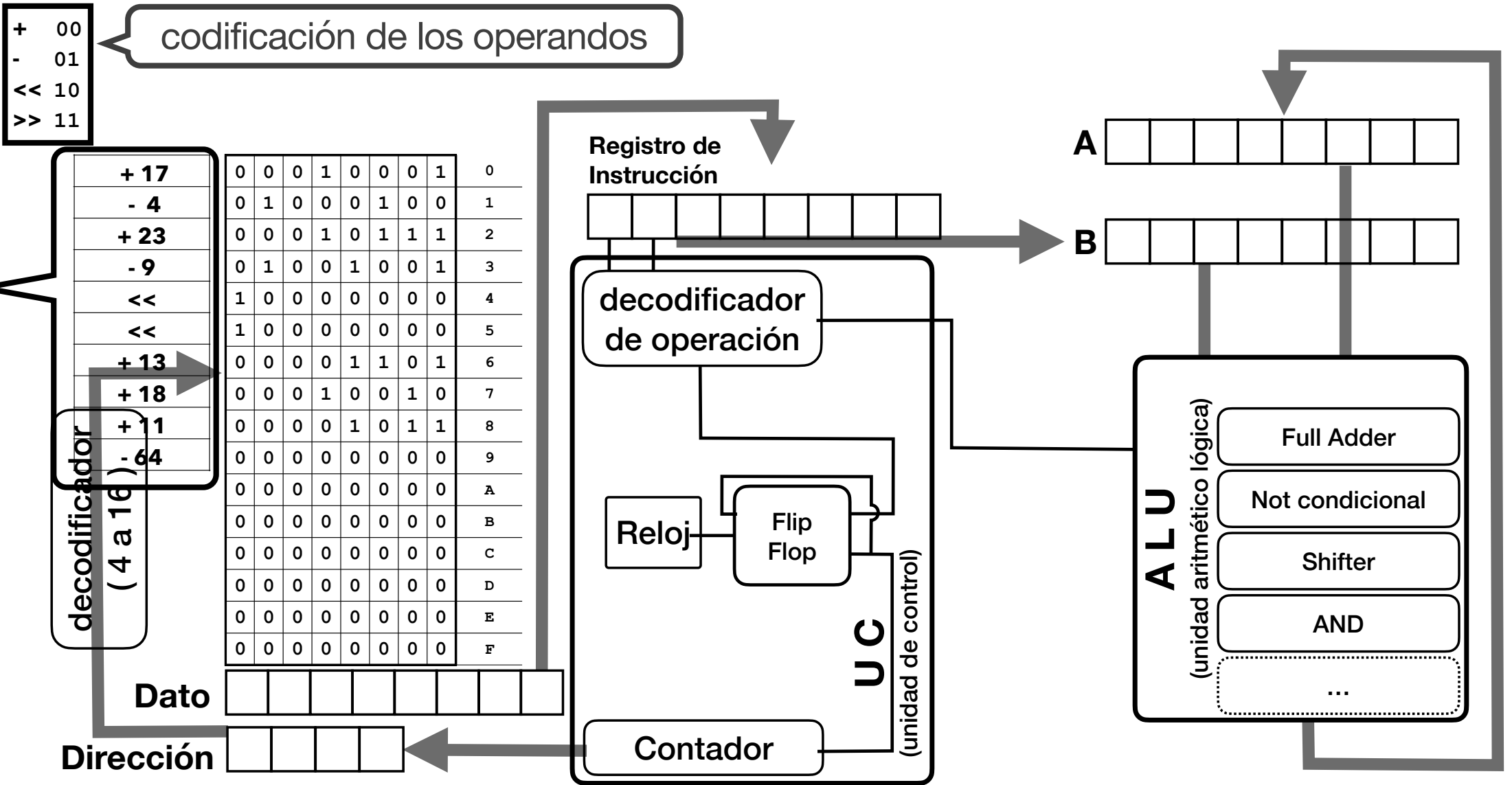
Según la cantidad de bits
puedo identificar 2^n
operaciones **diferentes**
de mi **lenguaje**

Dado que resignamos bits para la
identificación del operador, tenemos
menos bits para representar los
valores. ¿ Dejamos algunos de la lista
con esta configuración ?

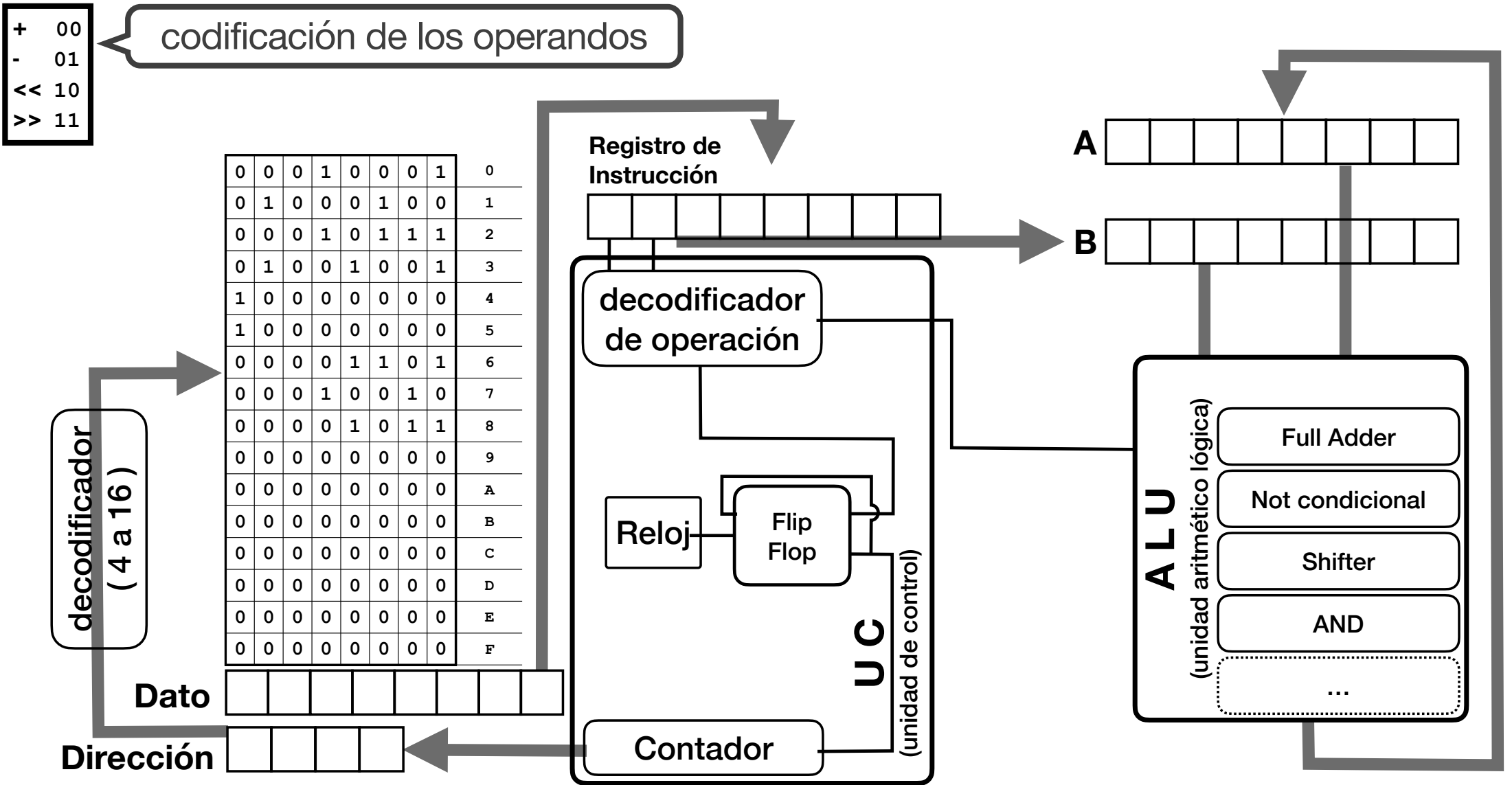
Automatizar una secuencia de operaciones: sumas, restas o *shift*



Automatizar una secuencia de operaciones: sumas, restas o *shift*



Automatizar una secuencia de operaciones: sumas, restas o *shift*



¿Cómo mejorar el espacio de trabajo?

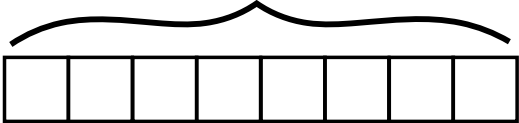
CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

OP OPERANDO

8 BITS = tamaño de
arquitectura

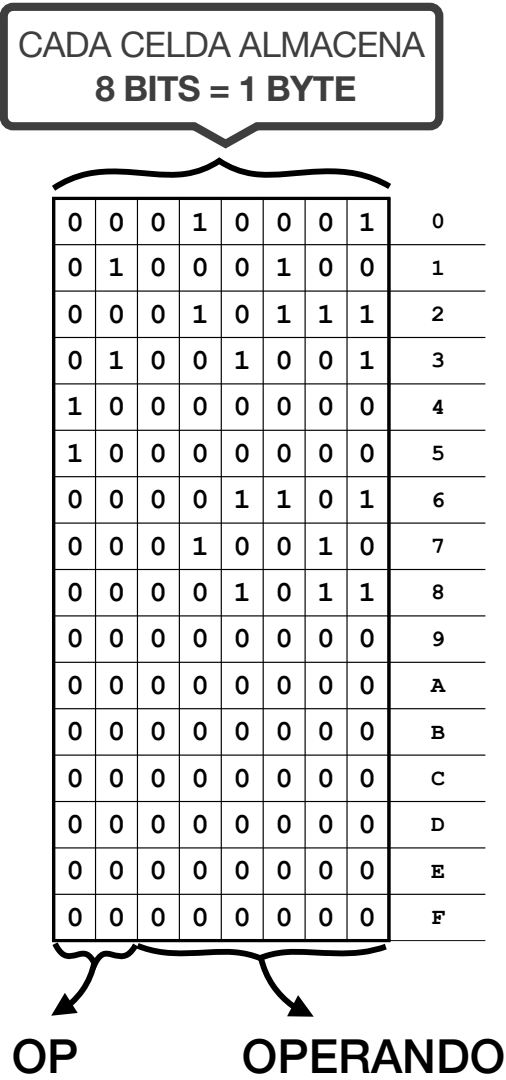
Registro
de instrucción



Registros de
uso general



¿Cómo mejorar el espacio de trabajo?



8 BITS = tamaño de arquitectura

Registro de instrucción

¿Qué sucede si queremos tener más operaciones, una mejor representación de datos o (guardar DATOS en la memoria)?

Teniendo en cuenta que

Registros de uso general

~~Cada celda tiene 1byte = 8bits~~

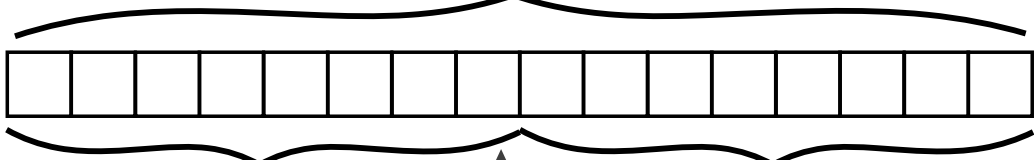
¿Cómo mejorar el espacio de trabajo?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

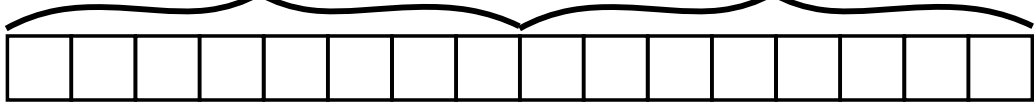
16 BITS = tamaño de
arquitectura

Registro
de instrucción



Teniendo registros de más bits y utilizando 2 celdas de memoria para cada instrucción (o para almacenar datos). Esta arquitectura nos permite configurar, por ej. operaciones con 2 operandos.

Registros de
uso general



Little endian vs Big endian

¿Cómo mejorar el espacio de trabajo?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

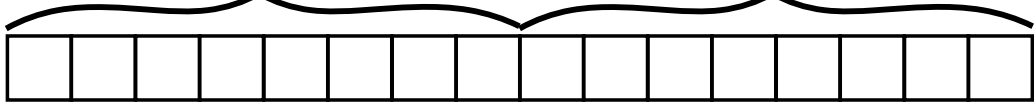
0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

Registro
de instrucción



Teniendo registros de más bits y utilizando 2 celdas de memoria para cada instrucción (o para almacenar datos). Esta arquitectura nos permite configurar, por ej. operaciones con 2 operandos.

Registros de
uso general



Little endian vs Big endian

¿Cómo mejorar el espacio de trabajo?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

16 BITS = tamaño de
arquitectura

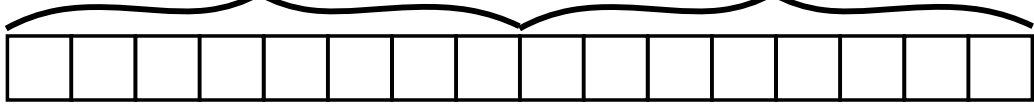
Registro
de instrucción



Teniendo registros de más bits y utilizando 2 celdas de memoria para cada instrucción (o para almacenar datos). Esta arquitectura nos permite configurar, por ej. operaciones con 2 operandos.

Para usar la memoria para guardar DATOS, puedo acceder a ellos a través de su dirección de memoria, es decir, el valor del operando es la dirección donde está almacenada la información

Registros de
uso general



Little endian vs Big endian

¿Cómo leer o guardar valores en la memoria?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

INST. COD	OPERADOR1	OPERADOR2
-----------	-----------	-----------

Ejemplo

GUARDAR	DIRRECCION C	REGISTRO A
---------	--------------	------------

0	1	1	1	0	0	1	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Registro de instrucción

...

Guardar 07 (0111)

...

...

Dirección C (001100)

...

...

RegistroA (000000)

...

0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Registros de uso general A

--	--	--	--	--	--

Dirección



Leer/Escribir

¿Cómo leer o guardar valores en la memoria?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

--	--	--	--	--	--

Dirección

1

Leer/Escribir

INST. COD	OPERADOR1	OPERADOR2
-----------	-----------	-----------

Ejemplo

GUARDAR	DIRRECCION C	REGISTRO A
---------	--------------	------------

0	1	1	1	0	0	1	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Registro de instrucción

...
Guardar 07 (0111)
...

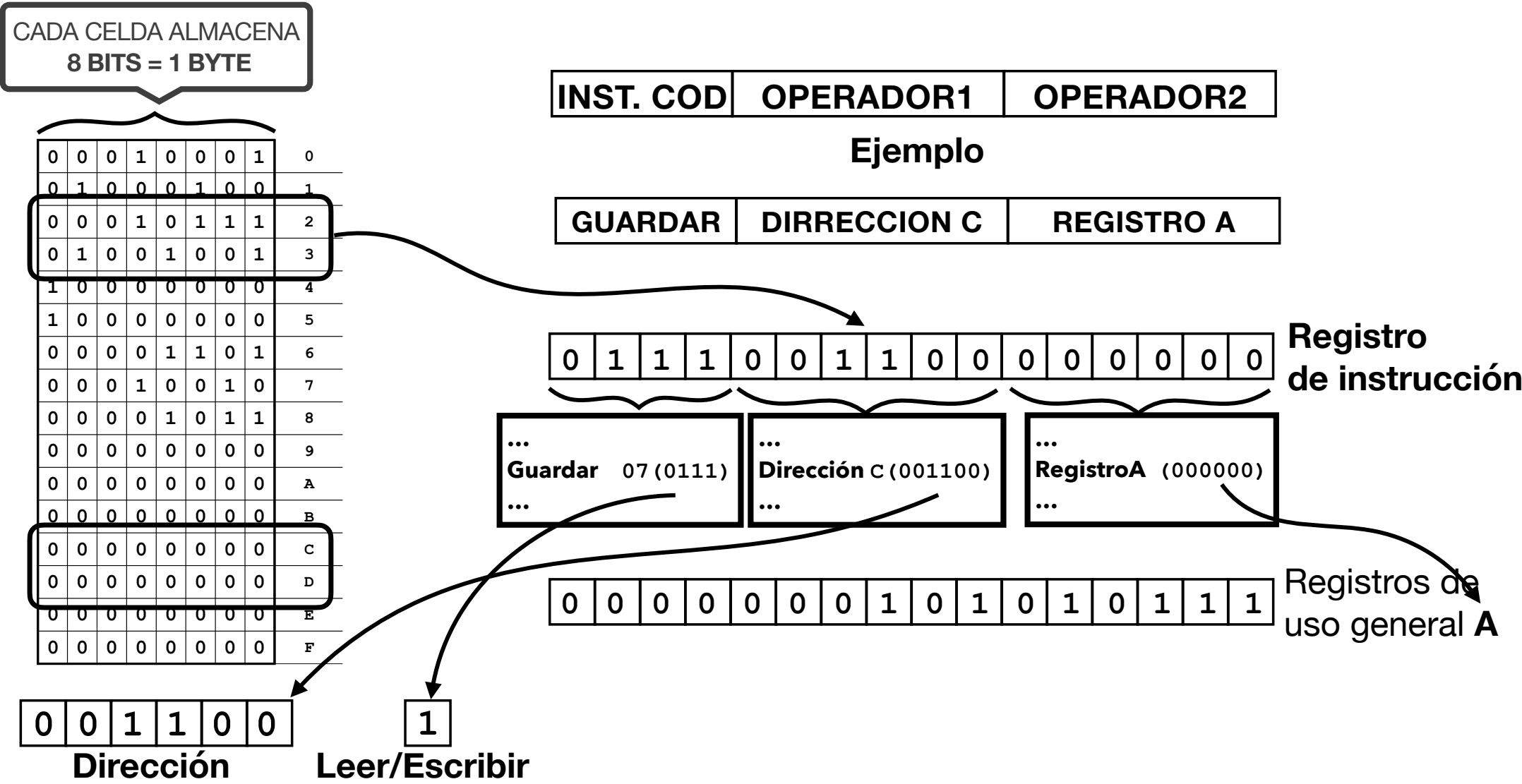
...
Dirección C (001100)
...

...
RegistroA (000000)
...

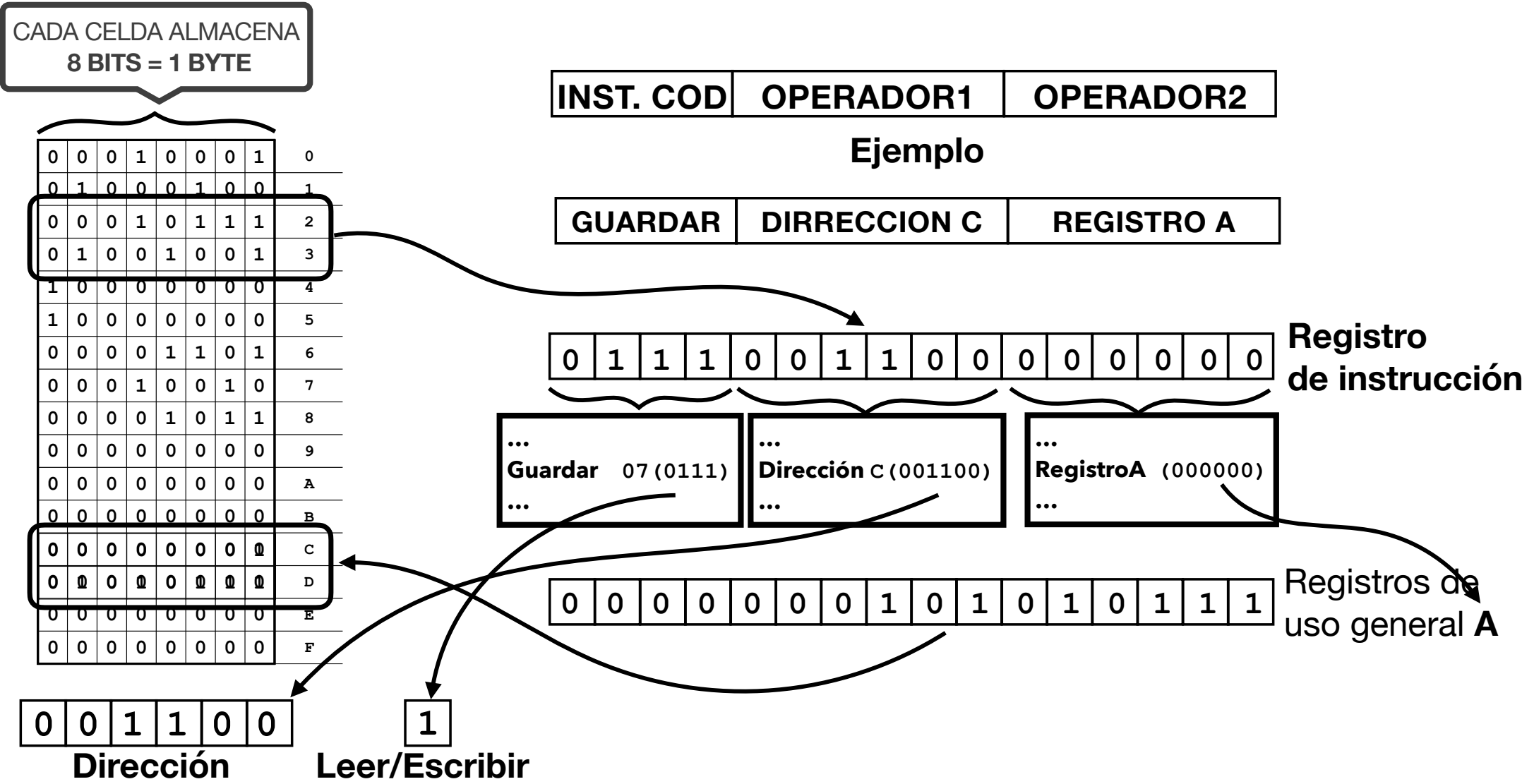
0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Registros de uso general A

¿Cómo leer o guardar valores en la memoria?



¿Cómo leer o guardar valores en la memoria?



¿Cómo podemos controlar el orden de ejecución de instrucciones?

CADA CELDA ALMACENA
8 BITS = 1 BYTE

0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	0	0	1	0	1	1	1	2
0	1	0	0	1	0	0	1	3
1	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5
0	0	0	0	1	1	0	1	6
0	0	0	1	0	0	1	0	7
0	0	0	0	1	0	1	1	8
0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	0	0	A
0	0	0	0	0	0	0	0	B
0	0	0	0	0	0	0	0	C
0	0	0	0	0	0	0	0	D
0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	F

Program Counter
(Contador de Programa)

--	--	--	--	--	--

El Program Counter contiene la **dirección** de la próxima instrucción a ejecutar

Registro de Instrucción

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

A

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

B

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

U C

(unidad de control)

A L U

(unidad aritmético lógica)

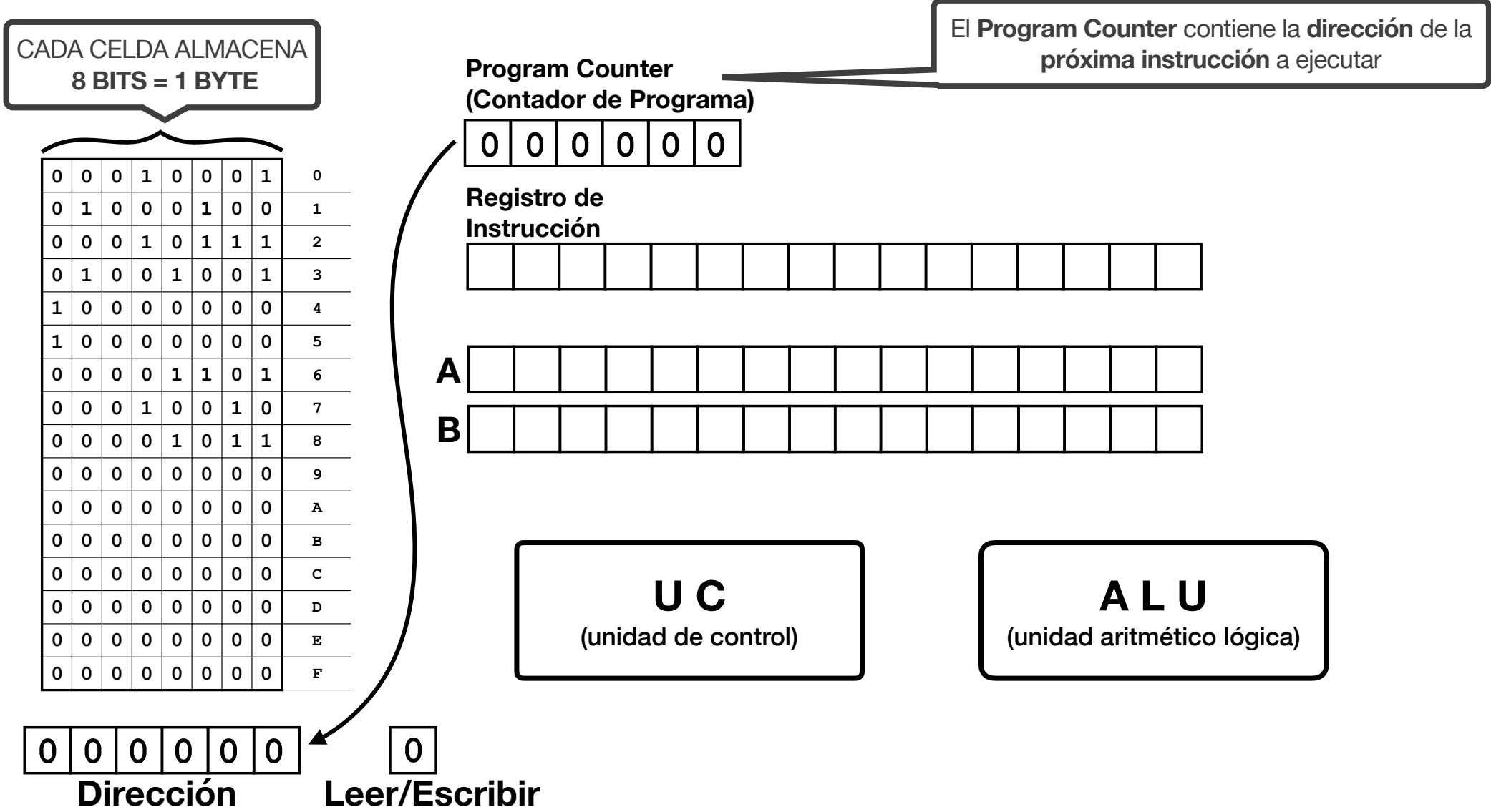
--	--	--	--	--	--

Dirección

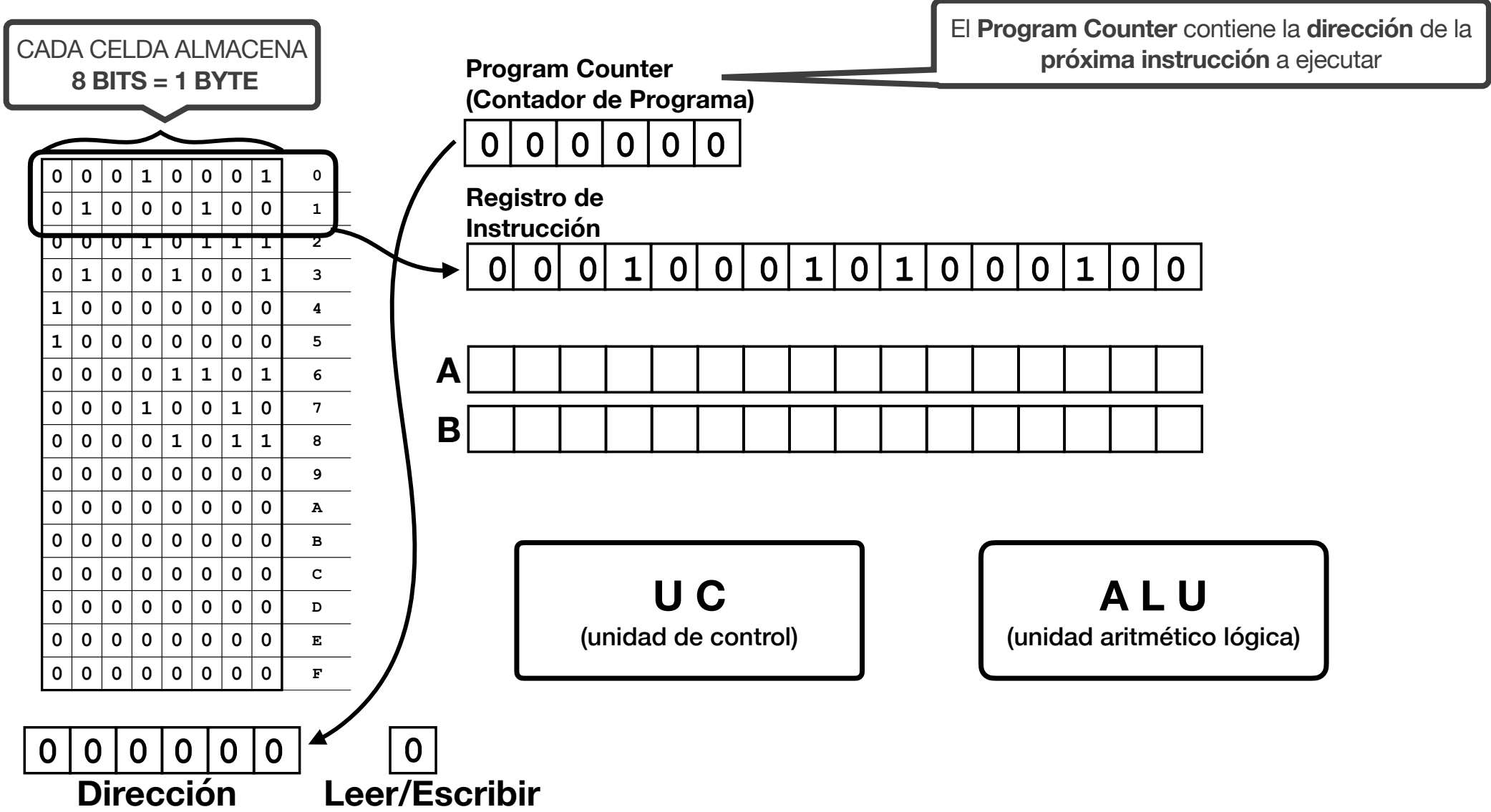
0

Leer/Escribir

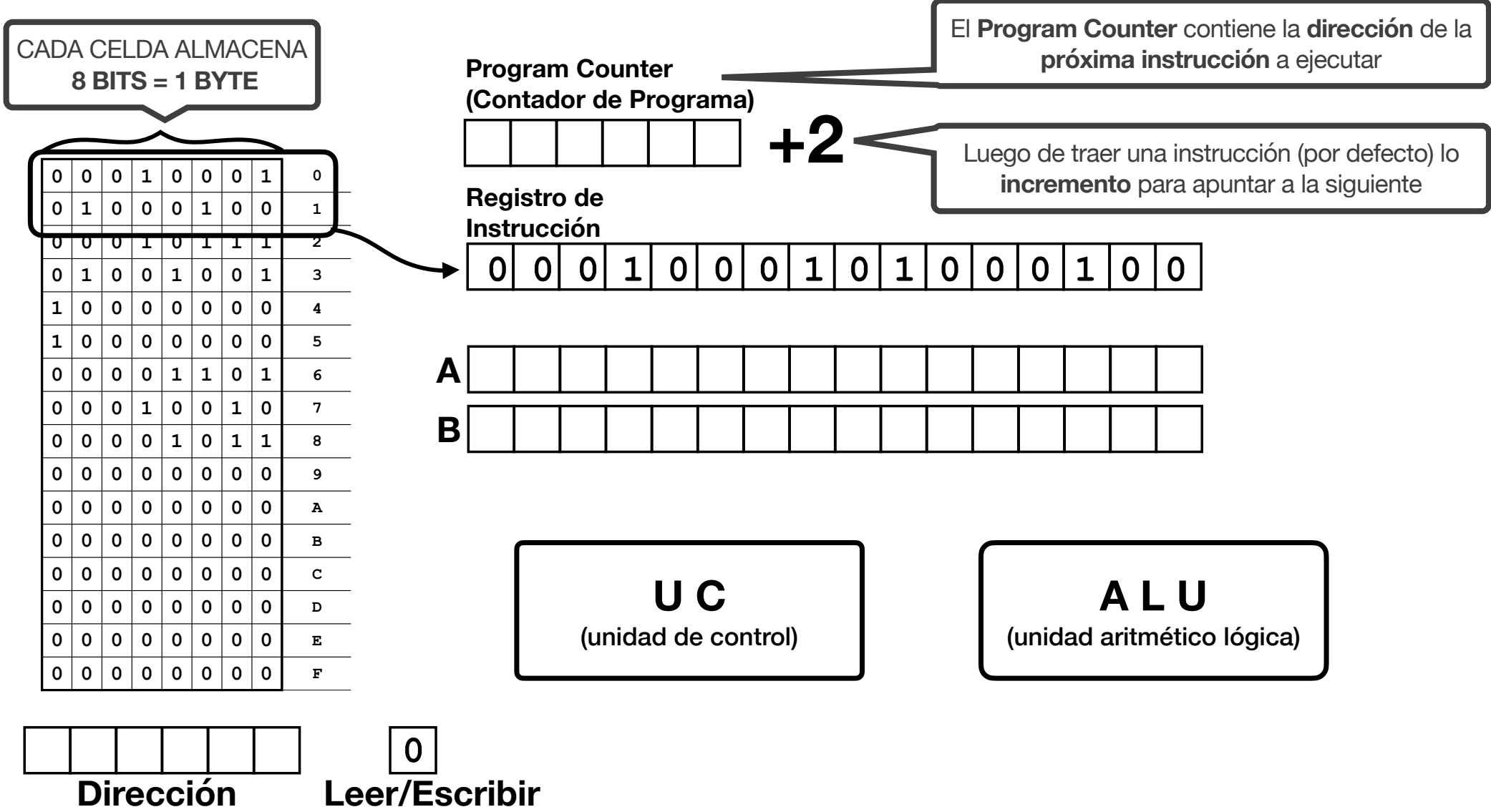
¿Cómo podemos controlar el orden de ejecución de instrucciones?



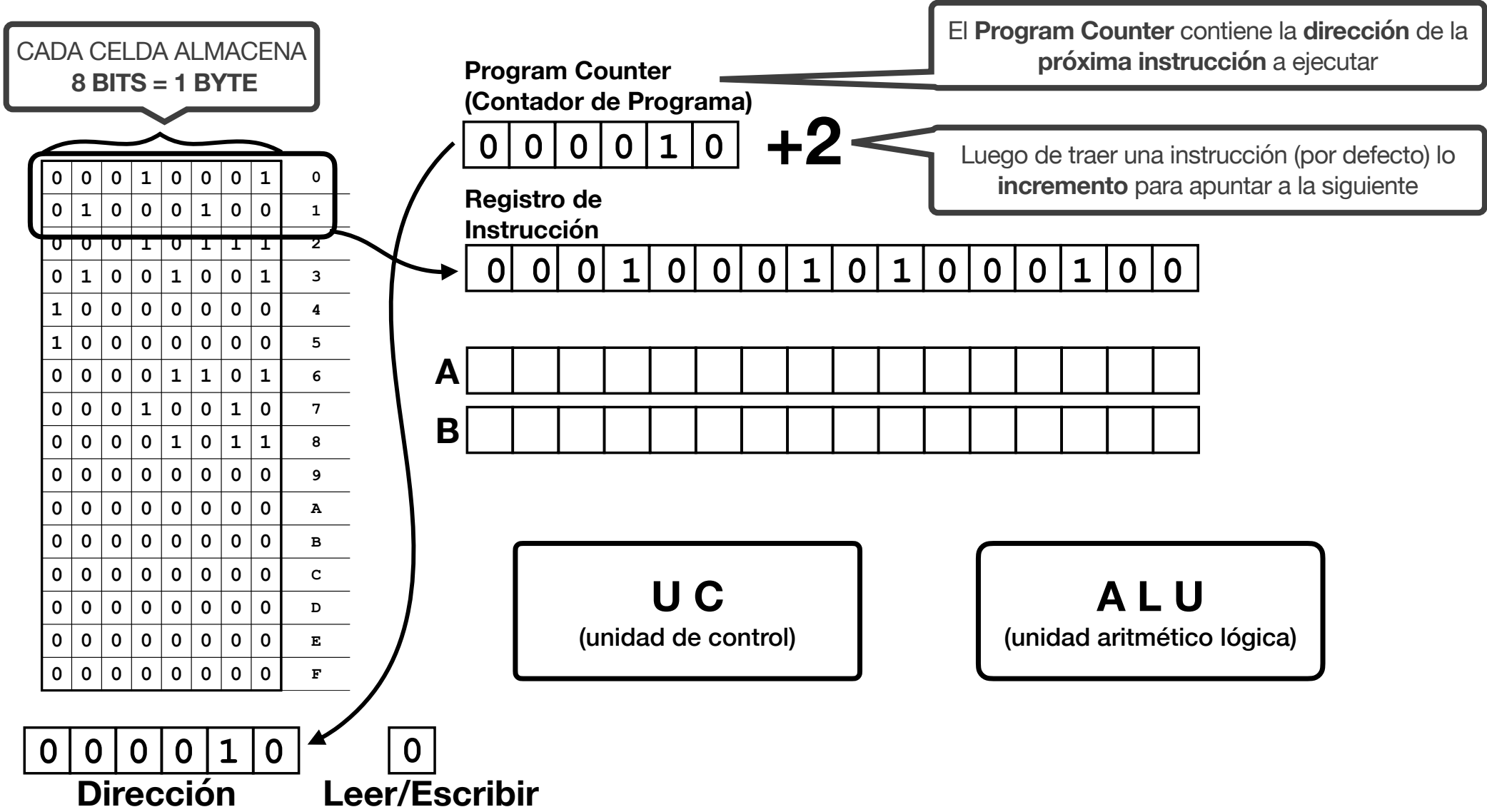
¿Cómo podemos controlar el orden de ejecución de instrucciones?



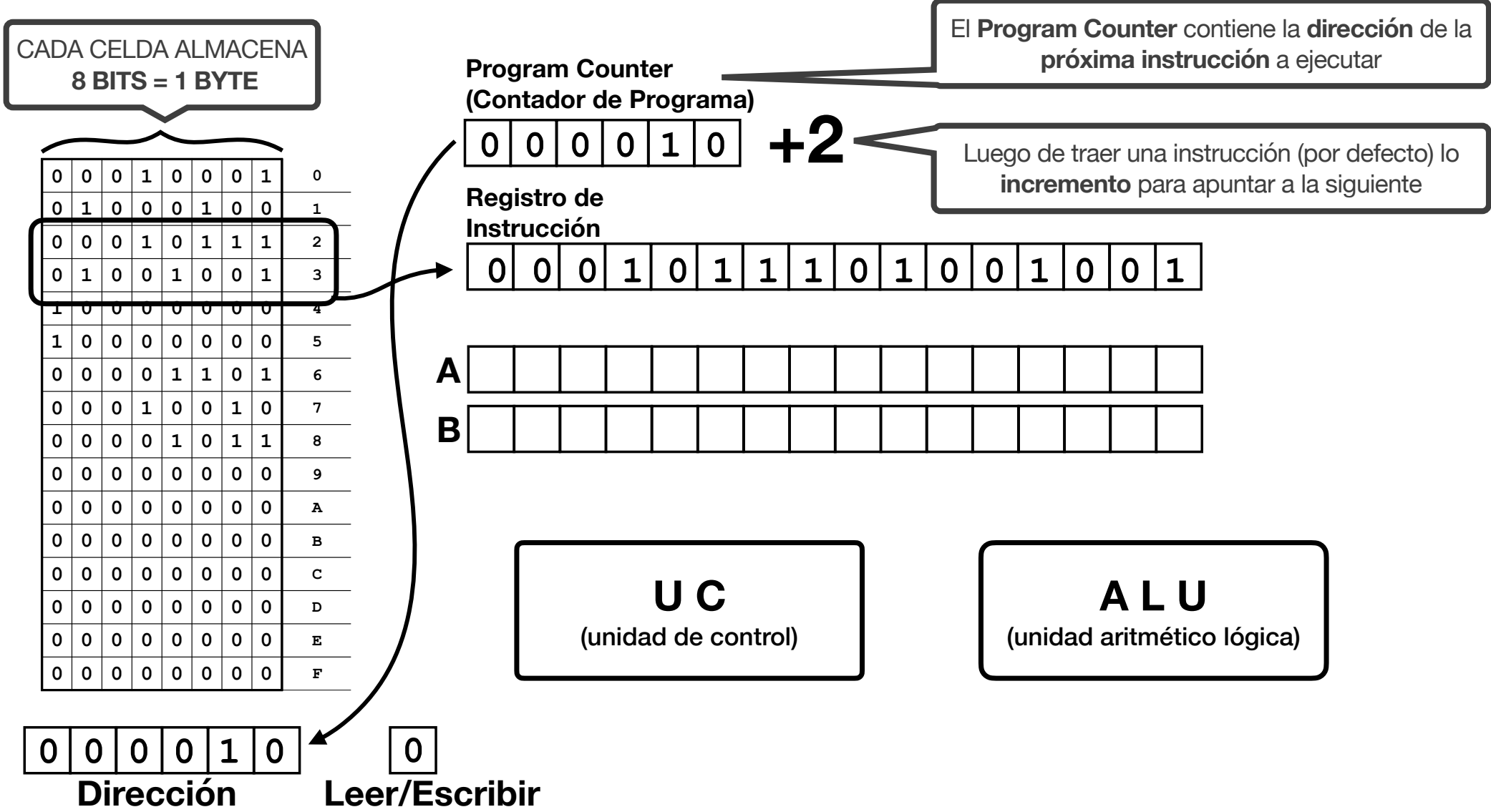
¿Cómo podemos controlar el orden de ejecución de instrucciones?



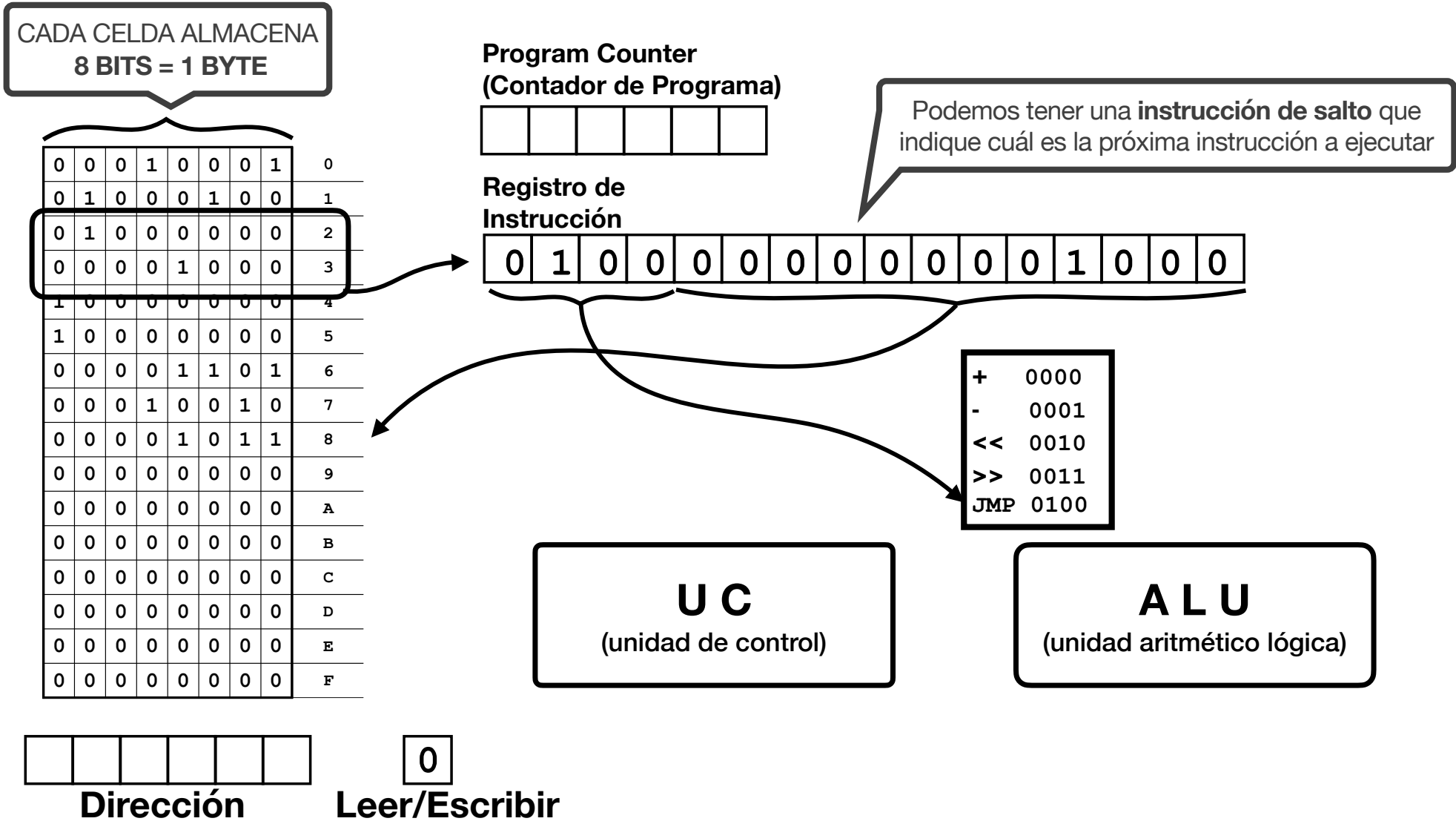
¿Cómo podemos controlar el orden de ejecución de instrucciones?



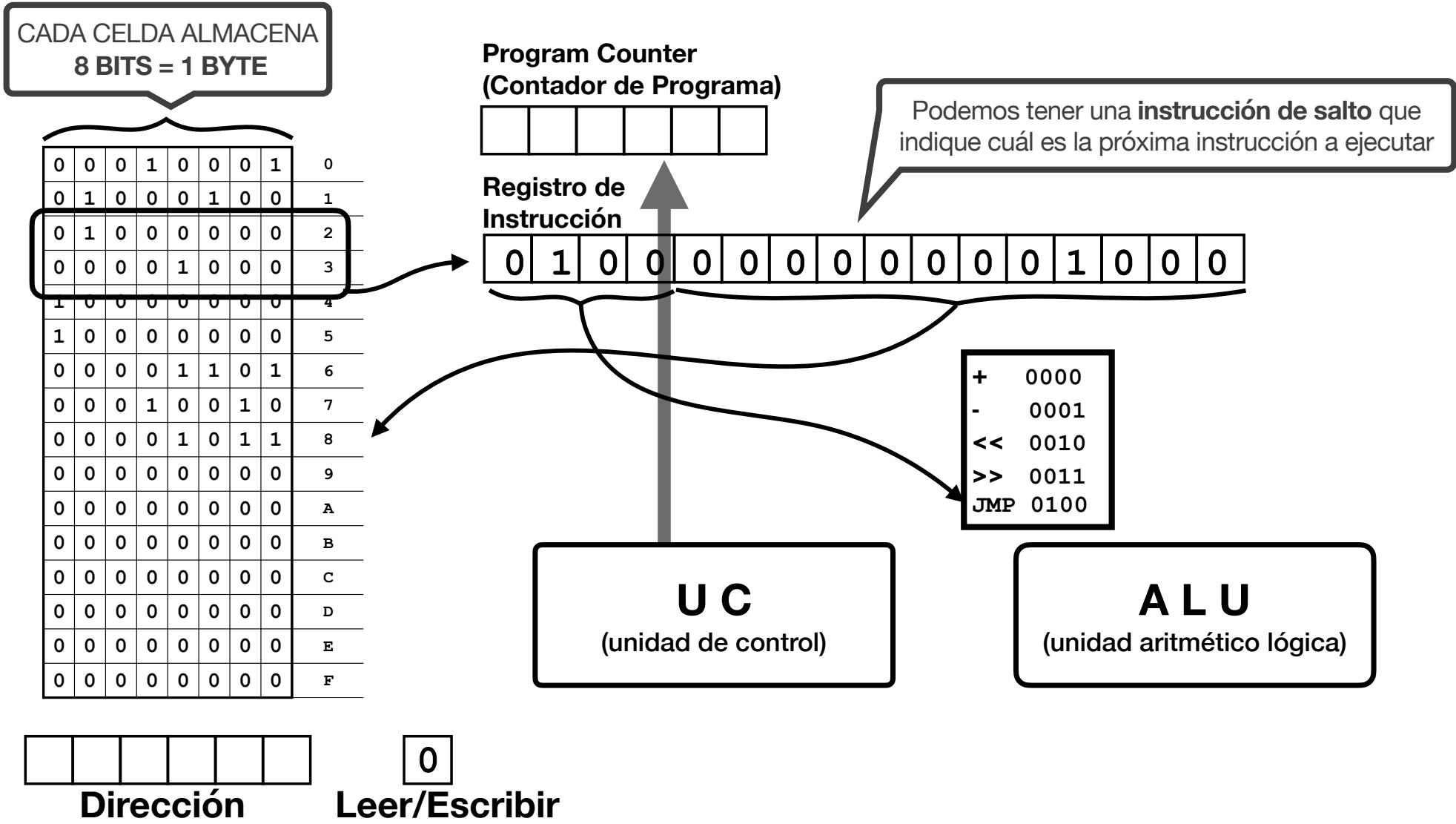
¿Cómo podemos controlar el orden de ejecución de instrucciones?



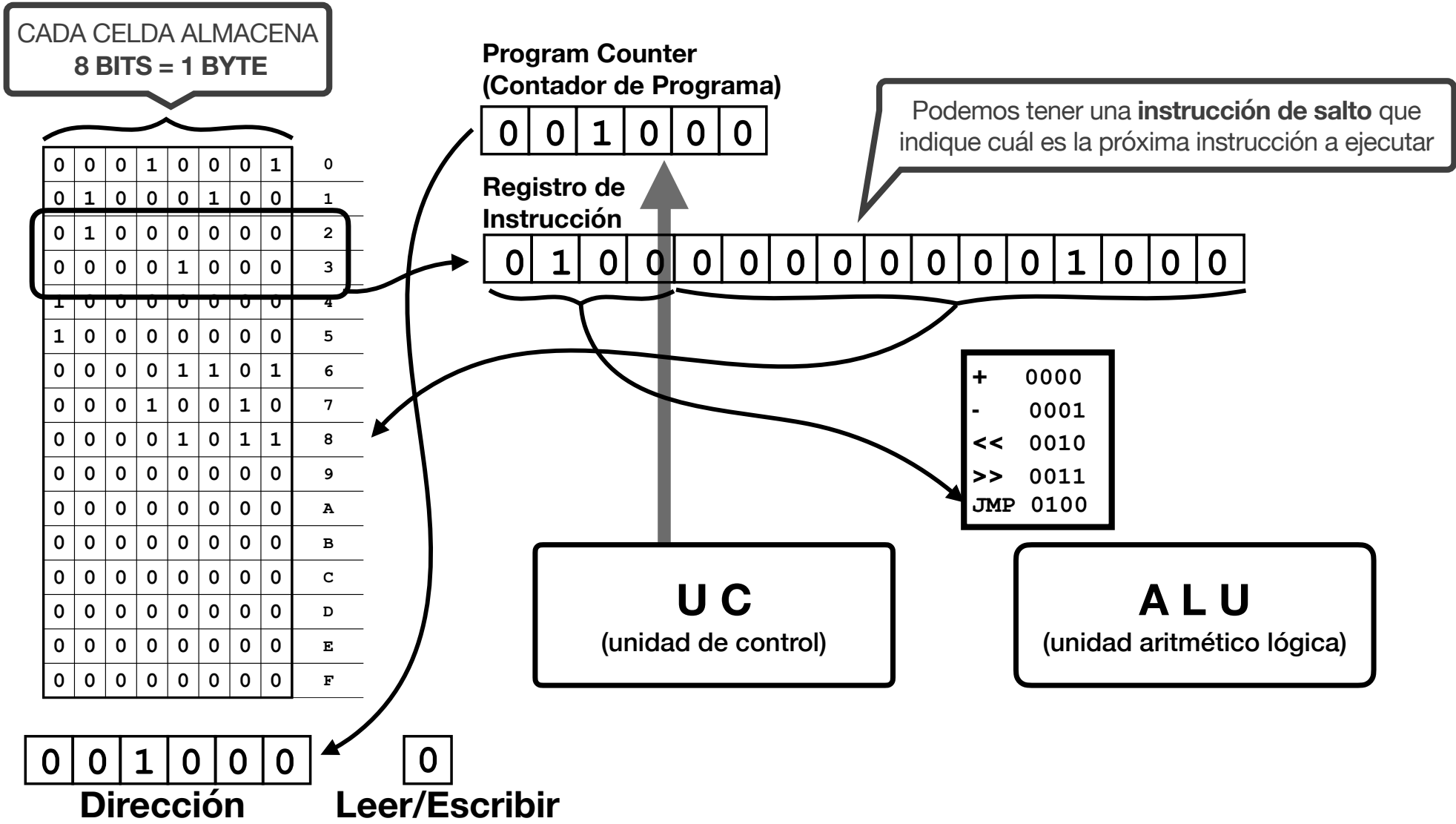
¿Cómo podemos *alterar* el orden secuencial?



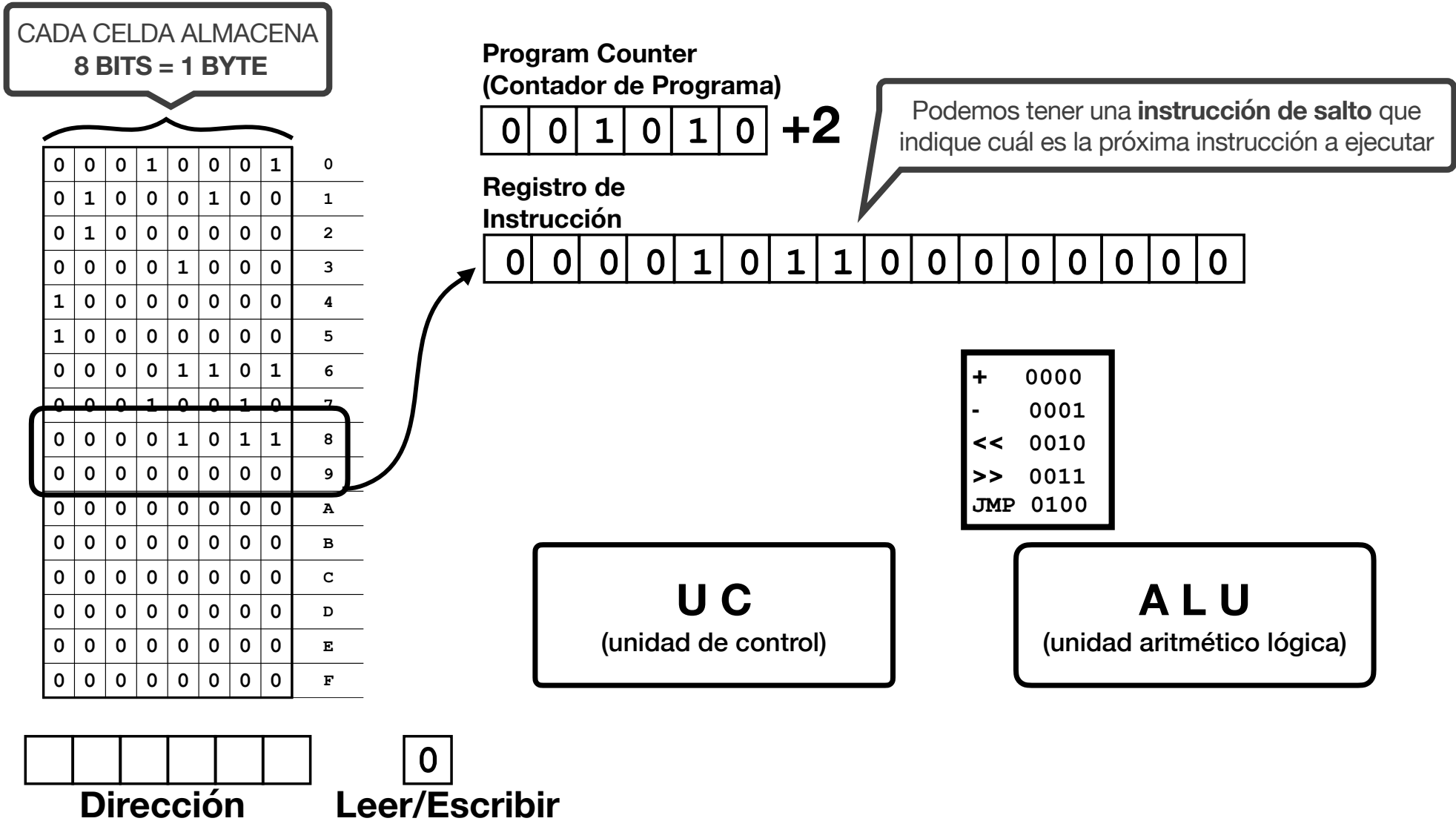
¿Cómo podemos *alterar* el orden secuencial?



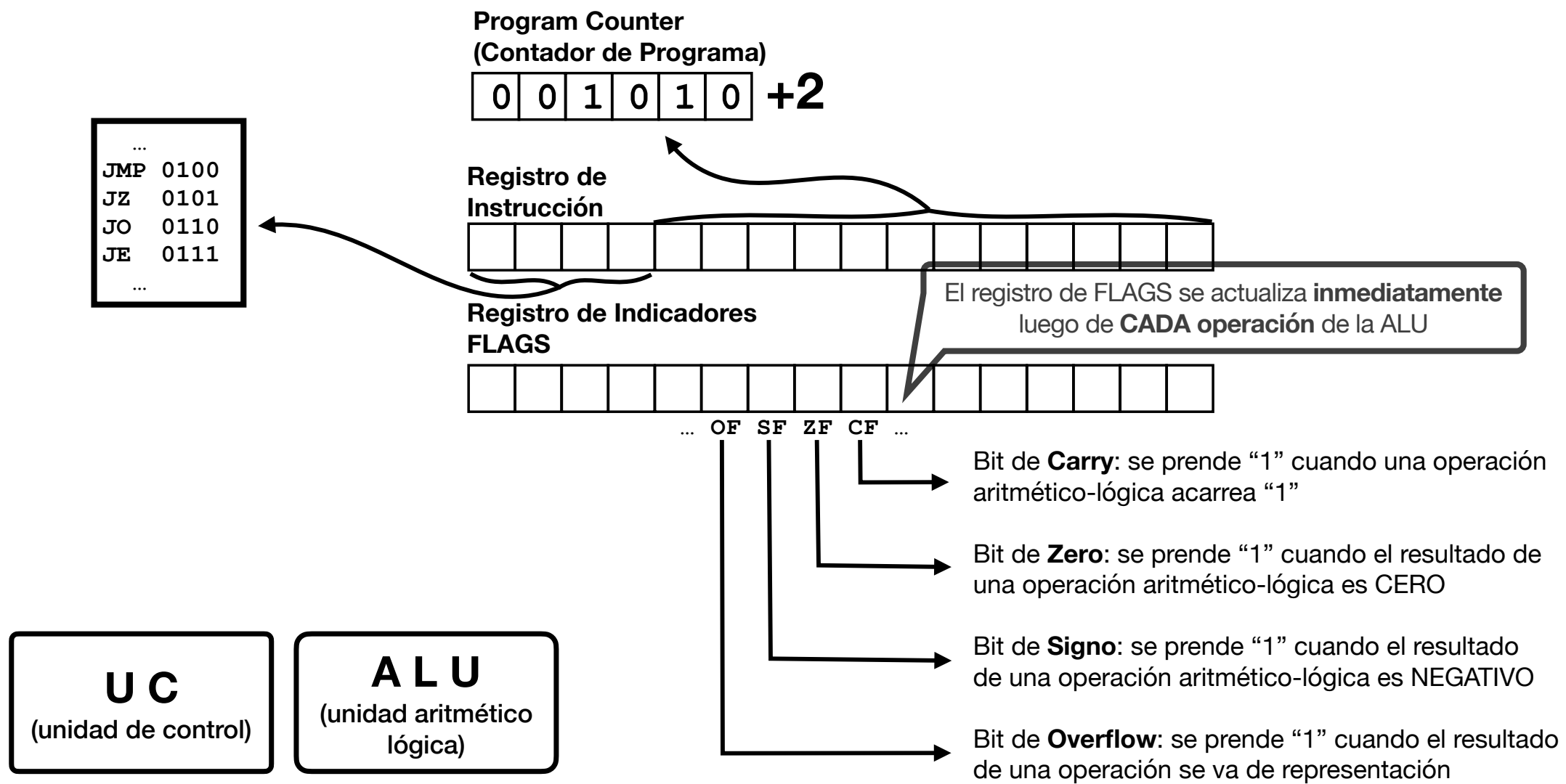
¿Cómo podemos *alterar* el orden secuencial?



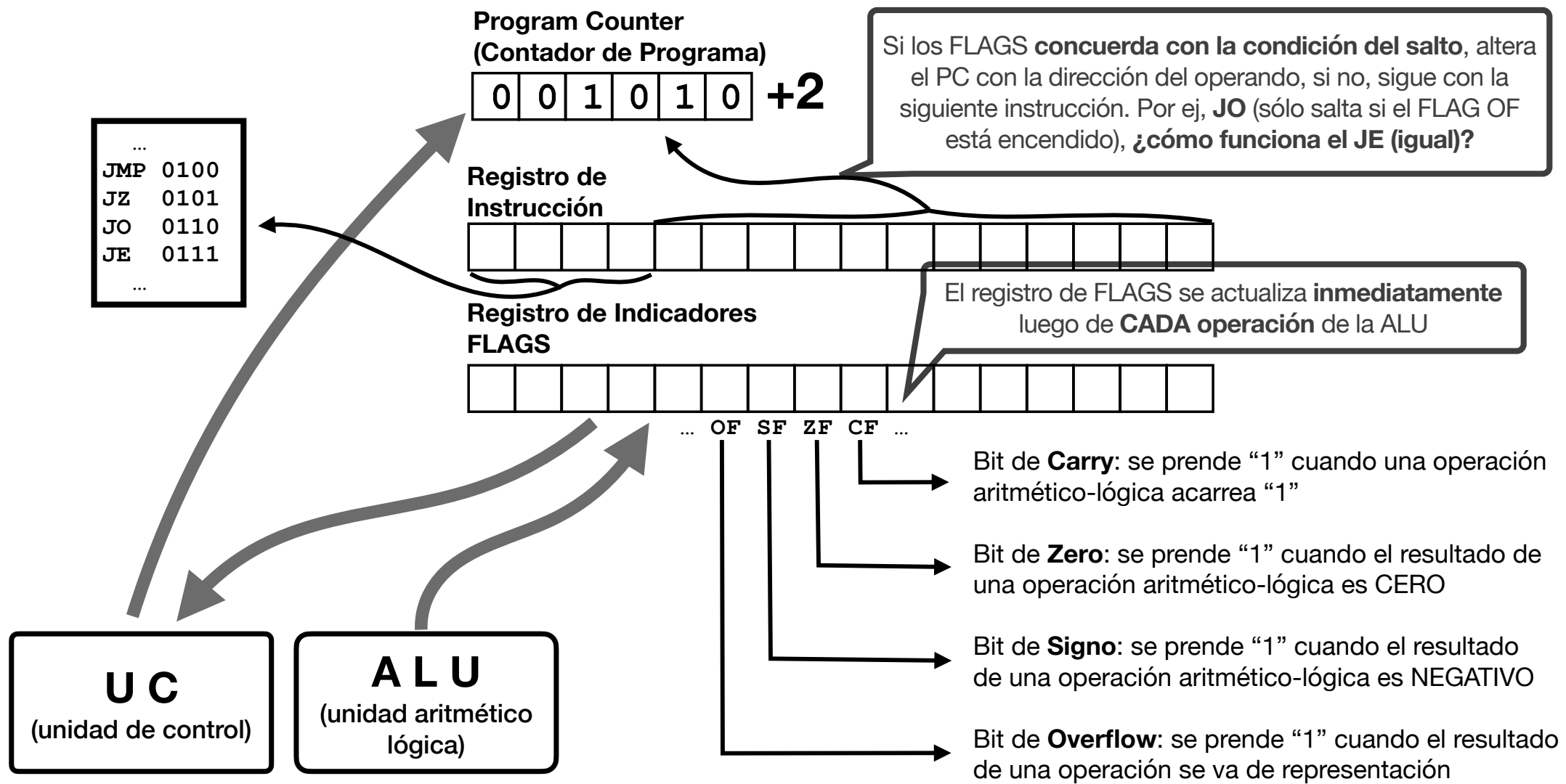
¿Cómo podemos *alterar* el orden secuencial?



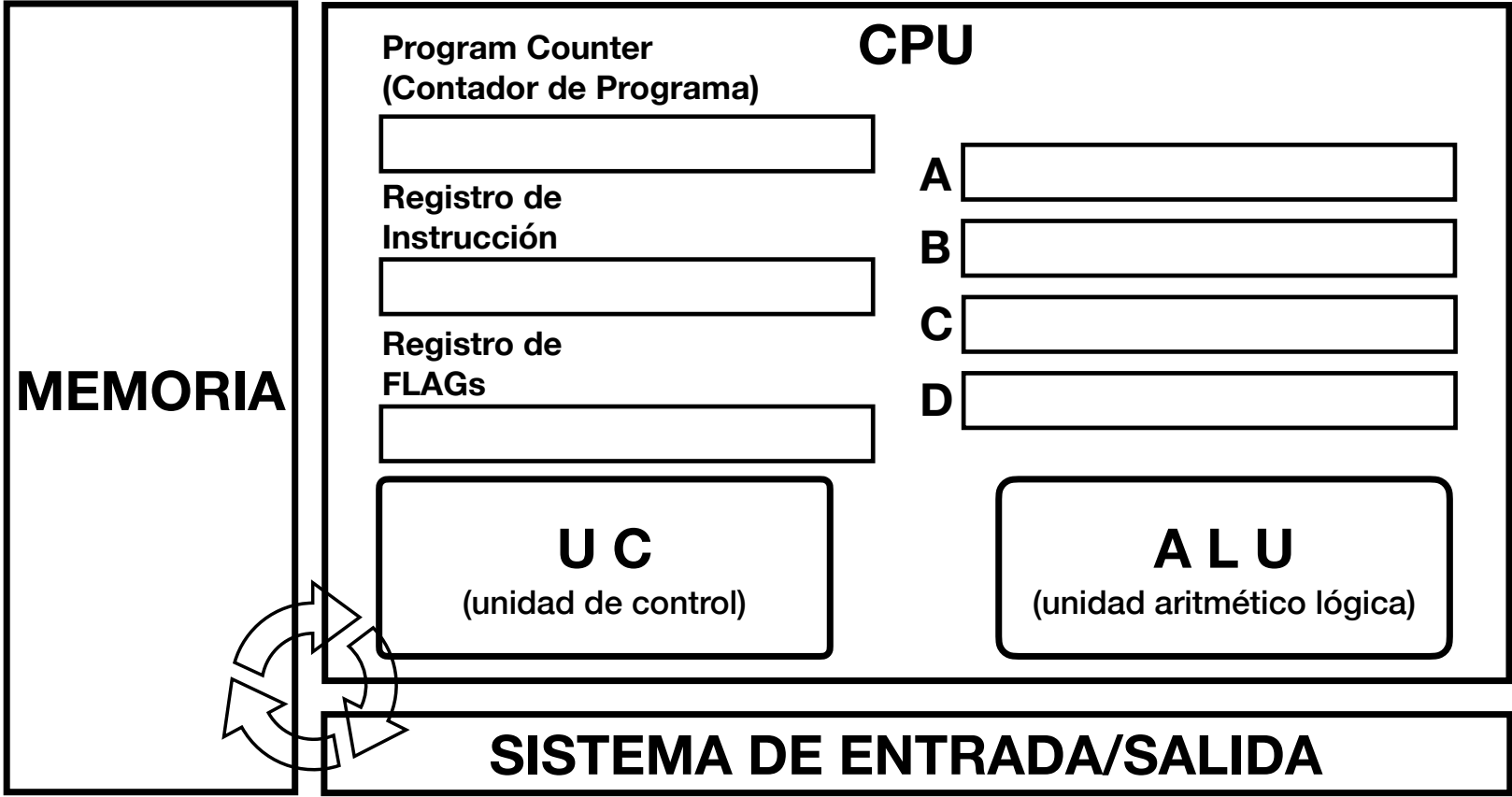
¿Cómo podemos *alterar* el orden secuencial bajo **CONDICIONES**?



¿Cómo podemos *alterar* el orden secuencial bajo **CONDICIONES**?

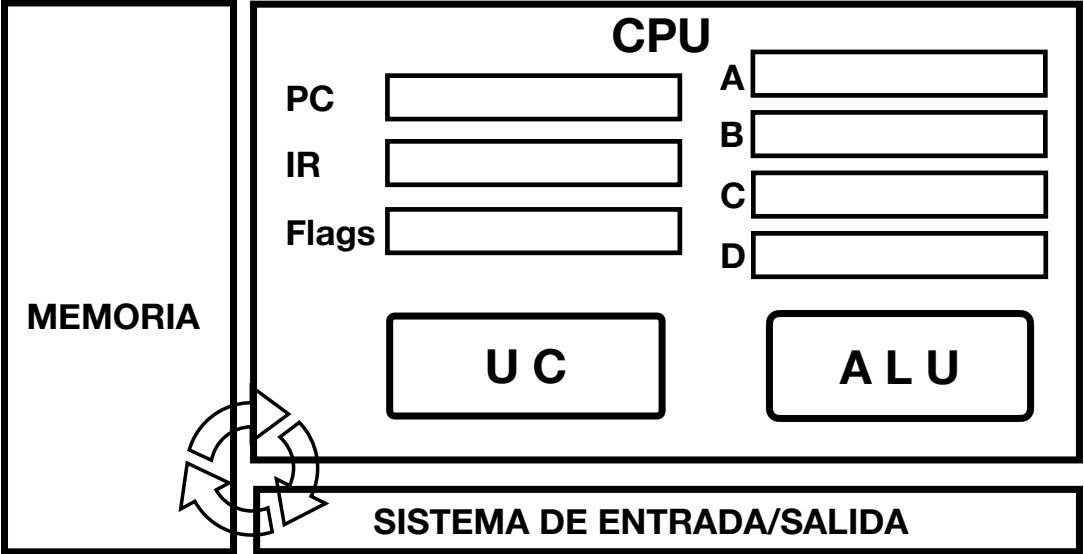


Arquitectura von Neumann



John von Neumann: (1903-1957) Físico Matemático Húngaro, Estadounidense. Participó activamente en el proyecto ENIAC y en el desarrollo de la EDVAC, donde propuso el concepto de almacenamiento de programas y datos en la memoria de la computadora, sentando las bases de la arquitectura de von Neumann, que se convirtió en el modelo predominante en las computadoras modernas.

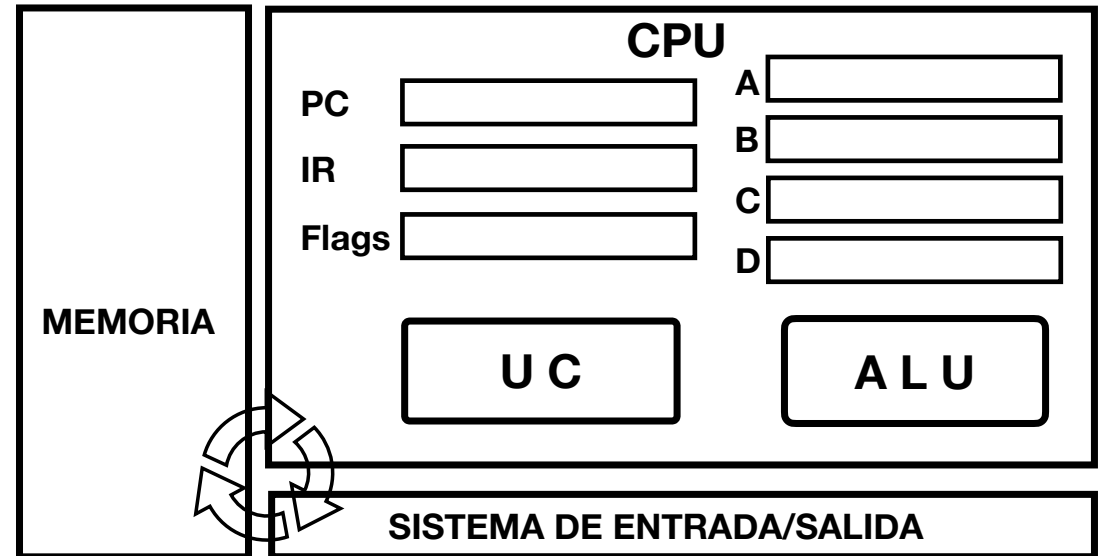
Arquitectura von Neumann



Arquitectura von Neumann

Fetch: La Unidad de control obtiene de la memoria la próxima instrucción que indica el *contador de programa* (**PC**) y la almacena en el Registro de Instrucción (**IR**). Finalmente actualiza el **PC** indicando la dirección de la próxima instrucción.

Decode: La Unidad de control decodifica la instrucción y obtiene de la memoria (si fuere necesario) la información que involucra dicha instrucción.



Execute: La ALU ejecuta (calcula) el resultado de la operación y lo almacena en un registro o memoria.