Lógica digital II

Arquitectura de Computadores – IIC2343

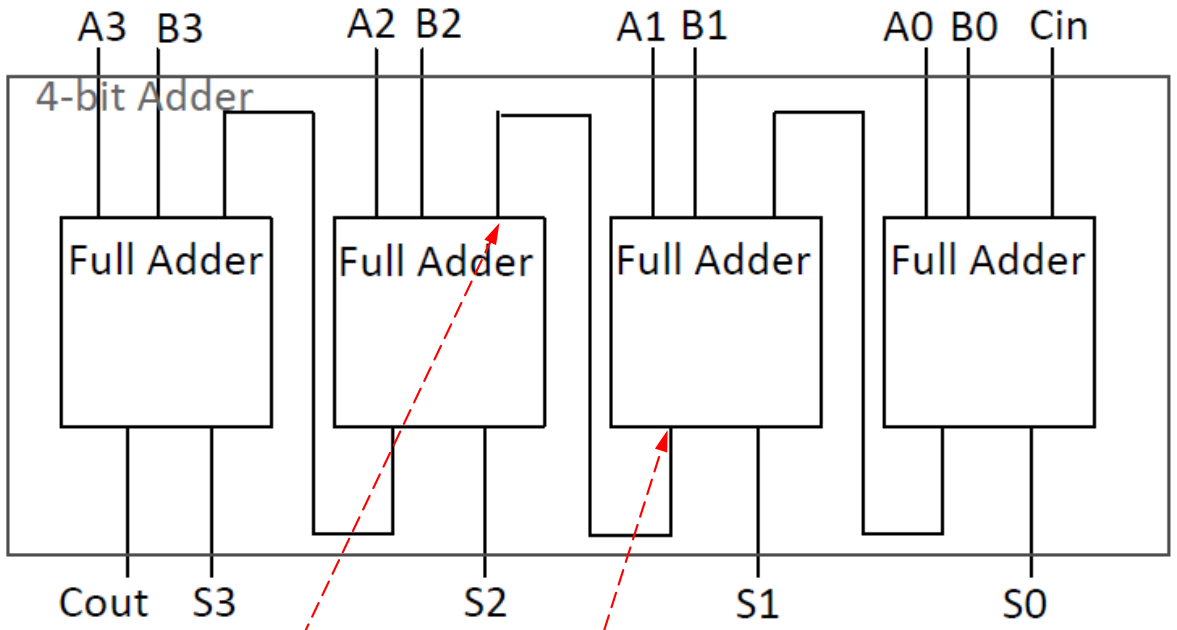
Yadran Eterovic S. (yadran@uc.cl)

2025-2

Un (circuito) sumador de sumandos de 4 bits

$$A_3A_2A_1A_0 + B_3B_2B_1B_0$$

... necesita 4 *full adders*, conectados como se muestra a continuación



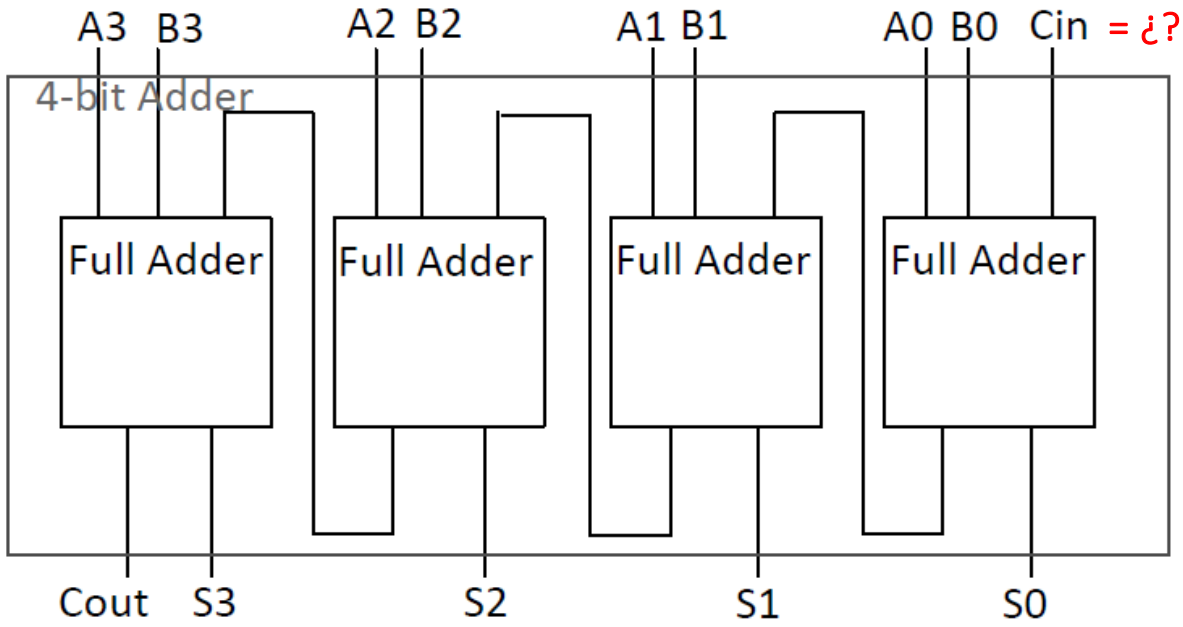
el C_{out} del *full adder* para el bit *i*-ésimo se conecta ...

... al C_{in} del *full adder* para el bit (*i*+1)-ésimo

Un (circuito) sumador de sumandos de 4 bits

$$A_3A_2A_1A_0 + B_3B_2B_1B_0$$

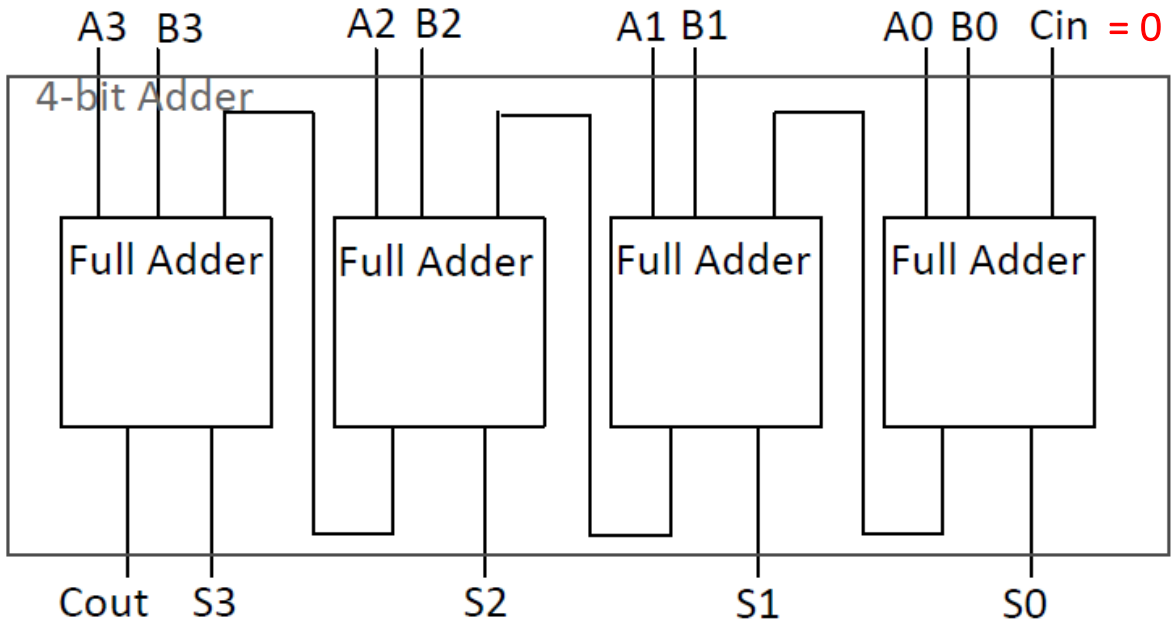
… necesita 4 *full adders*, conectados como se muestra a continuación



Un (circuito) sumador de sumandos de 4 bits

$$A_3A_2A_1A_0 + B_3B_2B_1B_0$$

… necesita 4 *full adders*, conectados como se muestra a continuación



¿ Y qué pasa si queremos restar números de 4 bits ?

¿ Podemos aprovechar el sumador ?

¿ Y qué pasa si queremos restar números de 4 bits ?

¿ Podemos aprovechar el sumador ?

Sabemos que $A - B = A + (-B)$

$\Longrightarrow$ en lugar de restar el sustraendo (B) al minuendo (A)

… podemos, equivalentemente, sumar al minuendo

… el inverso aditivo del sustraendo ($-B$)

¿ Y qué pasa si queremos restar números de 4 bits ?

¿ Podemos aprovechar el sumador ?

Sabemos que $A - B = A + (-B)$

$\Longrightarrow$ en lugar de restar el sustraendo (B) al minuendo (A)

… podemos, equivalentemente, sumar al minuendo

… el inverso aditivo del sustraendo ($-B$)

¿ Por qué esto ayuda ?

Recordemos: en *complemento de 2* el inverso aditivo $-B$ se obtiene

… primero invirtiendo cada 0 de B a 1 y cada 1 de B a 0

… y luego sumando 1 al patrón de bits resultante

P.ej., si queremos restar:

	0	1	0	1	1	0	*minuendo A*
–	0	0	1	1	0	1	***sustraendo B***

… primero calculamos el inverso aditivo del sustraendo *B* (en *complemento de 2*), en dos pasos:

paso 1)	1	1	0	0	1	0	*invertimos cada dígito de B*
paso 2)	1	1	0	0	1	1	*… y sumamos 1 al resultado*

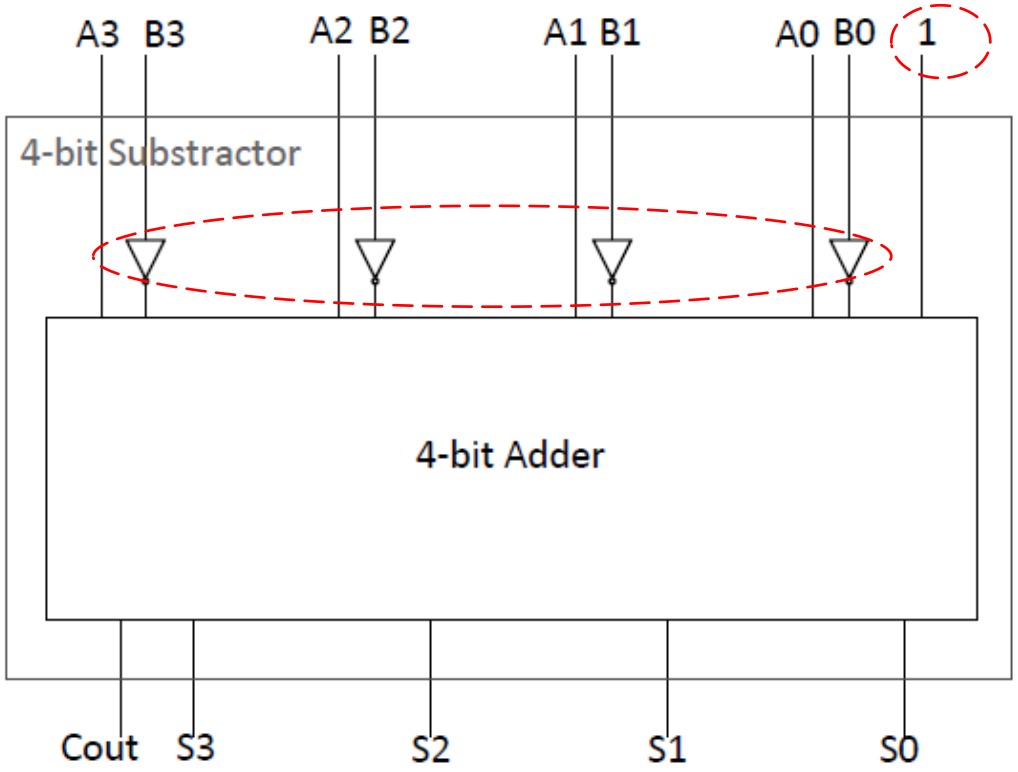
… y así convertimos la resta anterior en una suma:

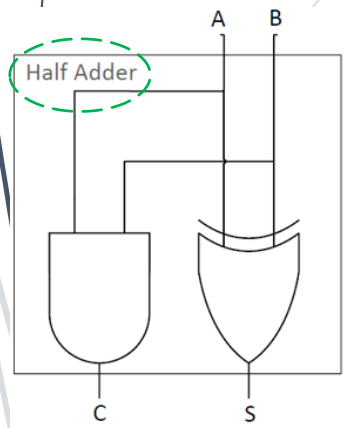
	0	1	0	1	1	0	*A*
+	1	1	0	0	1	1	*–B*

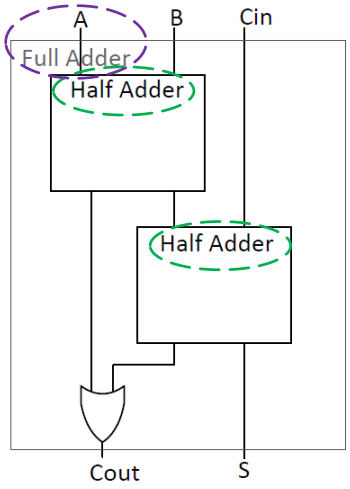
Por lo tanto, un (circuito) **restador** de operandos de 4 bits

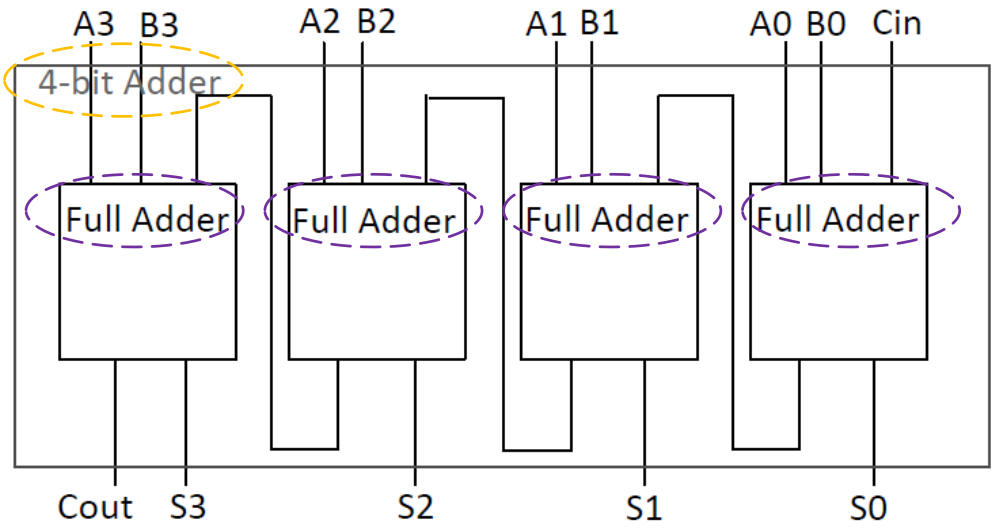
$$A_3A_2A_1A_0 - B_3B_2B_1B_0$$

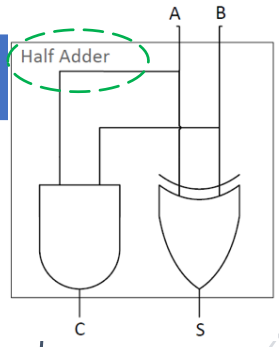
... es como se muestra a continuación:

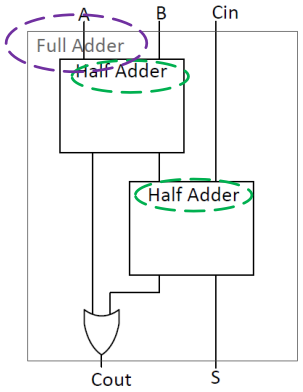


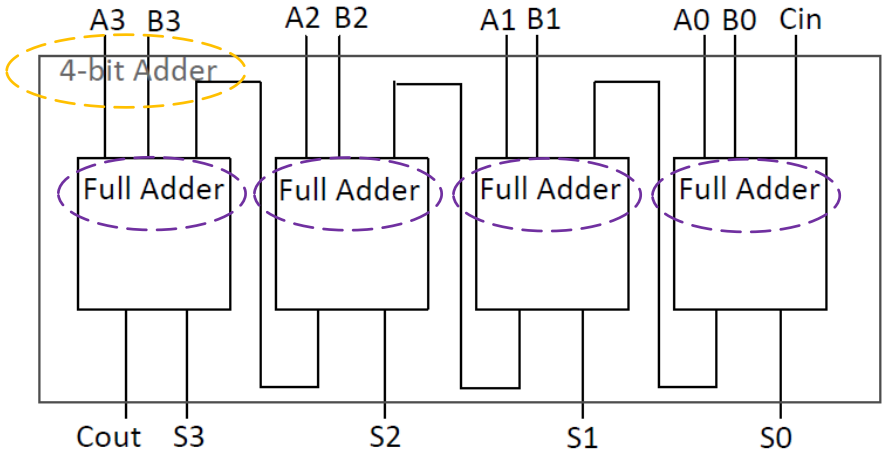

A
B
Half Adder
C
S

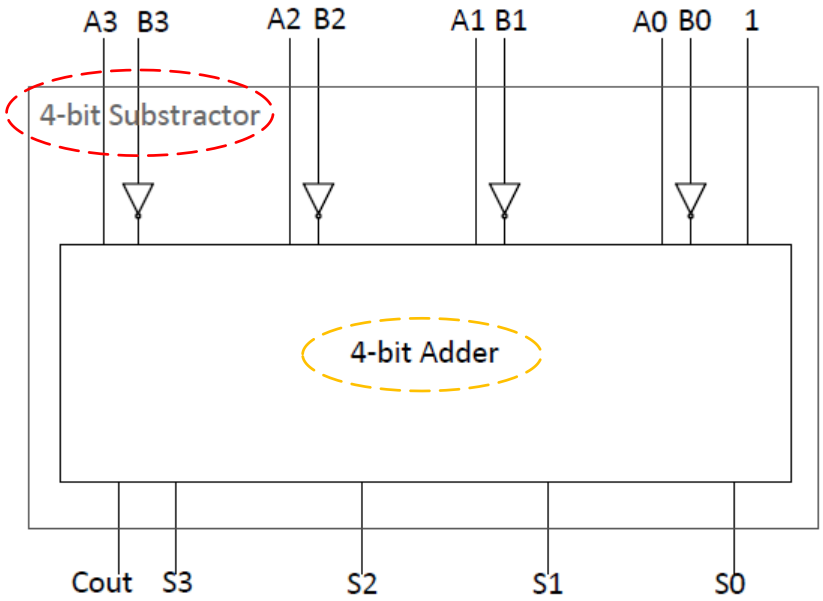

A
B
Cin
Full Adder
Half Adder
Half Adder
Cout
S


A3 B3
A2 B2
A1 B1
A0 B0
Cin
4-bit Adder
Full Adder
Full Adder
Full Adder
Full Adder
Cout
S3
S2
S1
S0


A
B
Half Adder
C
S


A
B
Cin
Full Adder
Half Adder
Half Adder
Cout
S


A3 B3
A2 B2
A1 B1
A0 B0
Cin
4-bit Adder
Full Adder
Full Adder
Full Adder
Full Adder
Cout
S3
S2
S1
S0


A3 B3
A2 B2
A1 B1
A0 B0
1
4-bit Substractor
4-bit Adder
Cout
S3
S2
S1
S0

¿ Cómo construimos un circuito sumador/restador?

- ¿ podemos hacerlo usando sólo los elementos vistos hasta ahora ?

El circuito va a tener una **entrada *A***, otra **entrada *B*** y una **salida *S***, todas de de 4 bits

… pero de alguna manera tenemos que decirle si queremos que sume, $S = A + B$, o que reste, $S = A - B$

… ⟹ tenemos que *controlar la operación* del circuito

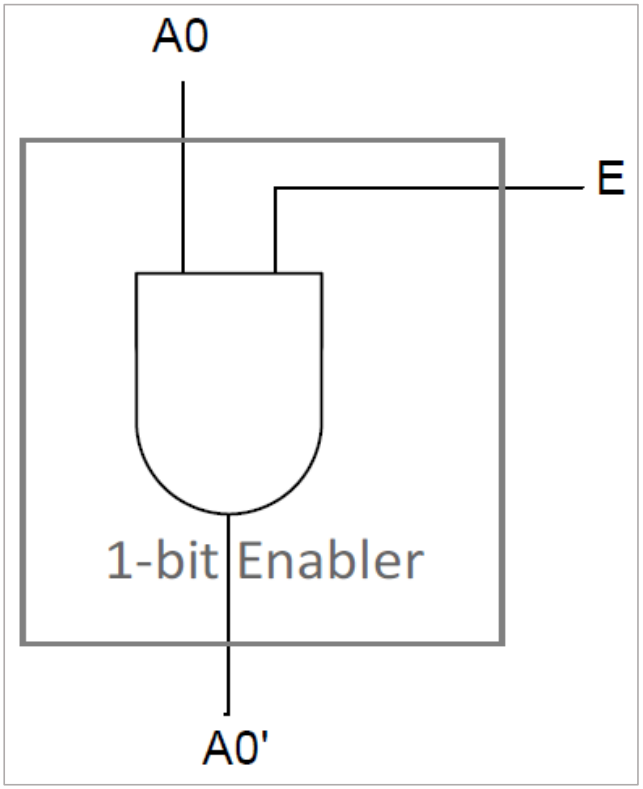
… ⟹ tenemos que poder **programar** el circuito:

… ⟹ además de *inputs* de datos, el circuito tiene que tener ***inputs*** **de control**

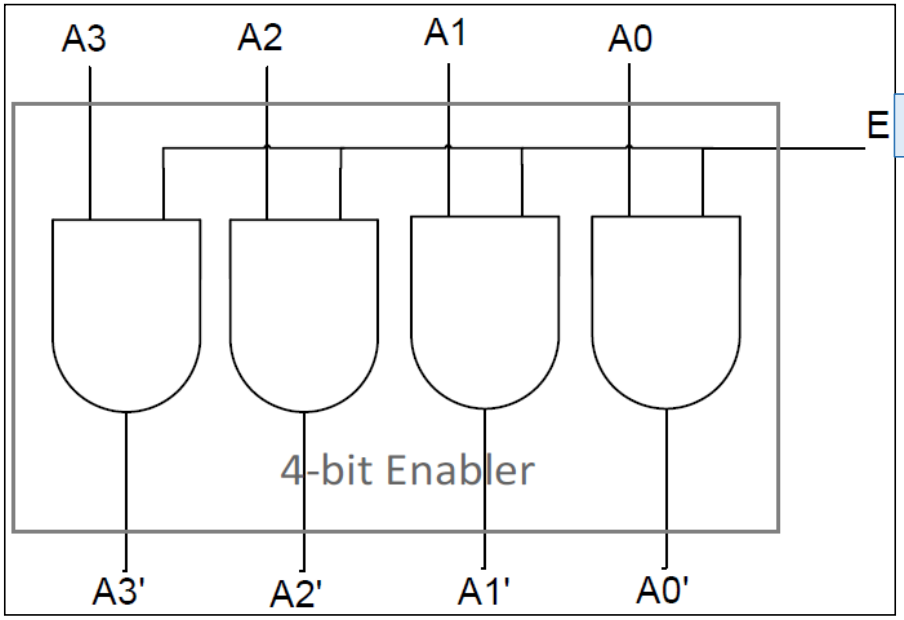
P.ej., este es un circuito muy simple —***1-bit Enabler***— con un input de control, *E*, además de un input de datos de un bit, A_0 :

- dependiendo del valor de *E*, el output A_0' será 0 (si $E = 0$) o exactamente igual a A_0 (si $E = 1$)

E	A0'
0	0
1	A0



P.ej., esta es la versión del circuito anterior para un input de datos de cuatro bits, $A_3A_2A_1A_0$ –***4-bit Enabler***



si $E = 0 \Longrightarrow 0\ 0\ 0\ 0$
si $E = 1 \Longrightarrow A_3A_2A_1A_0$

Un **multiplexor** (**mux**) es un circuito con 2^n inputs de datos (de m bits c/u)

… un output (de m bits)

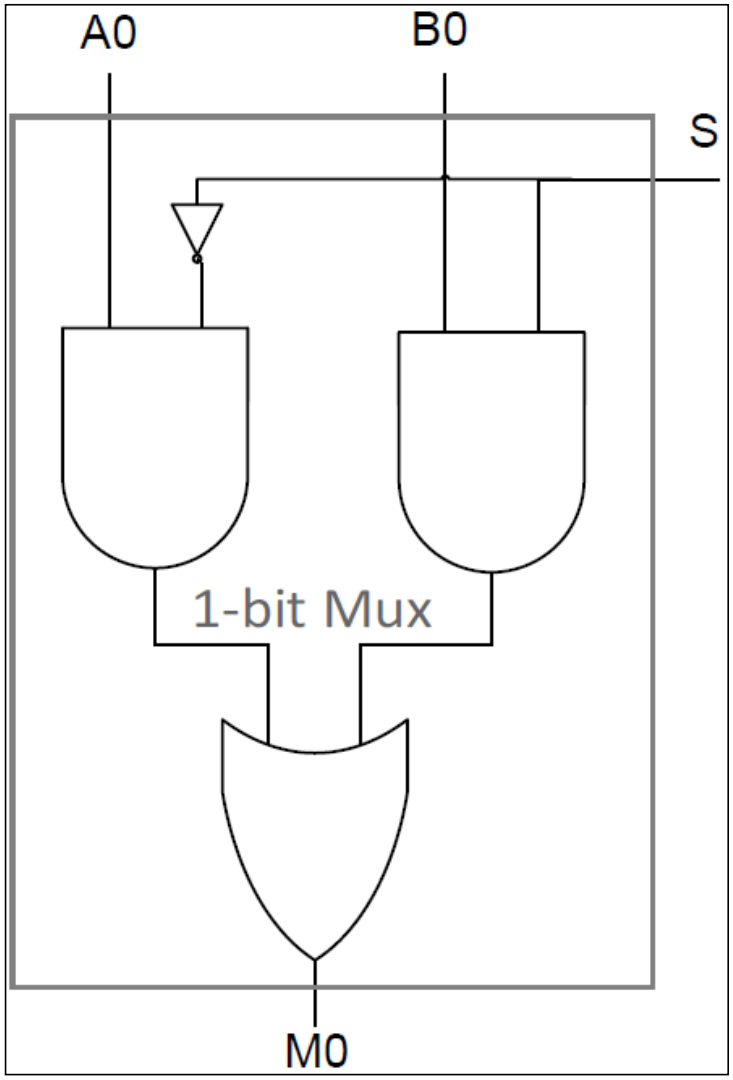
… y n inputs de control (de un bit c/u) que seleccionan exactamente uno de los inputs de datos y lo ponen en el output

P.ej., para $n = 1$ e inputs de un bit ($m = 1$), hay dos inputs de datos, A_0 y B_0, un input de control, S, y un output, M_0:

- dependiendo del valor de S, M_0 será exactamente igual a A_0 (si $S = 0$)

  … o a B_0 (si $S = 1$)

S	M0
0	A0
1	B0

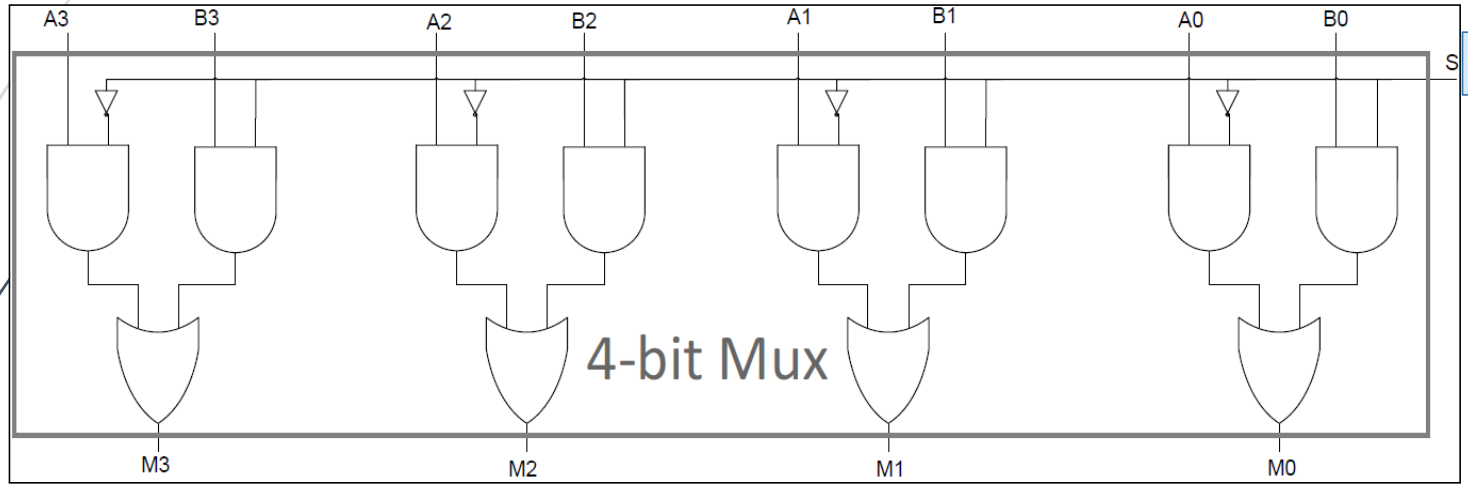


P.ej., para $n = 1$ e inputs de 4 bits ($m = 4$)

… hay dos inputs de datos, $A_3A_2A_1A_0$ y $B_3B_2B_1B_0$,

… un input de control, S,

… y un output, $M_3M_2M_1M_0$



$S = 0 / 1$

si $S = 0 \Longrightarrow A_3A_2A_1A_0$
si $S = 1 \Longrightarrow B_3B_2B_1B_0$

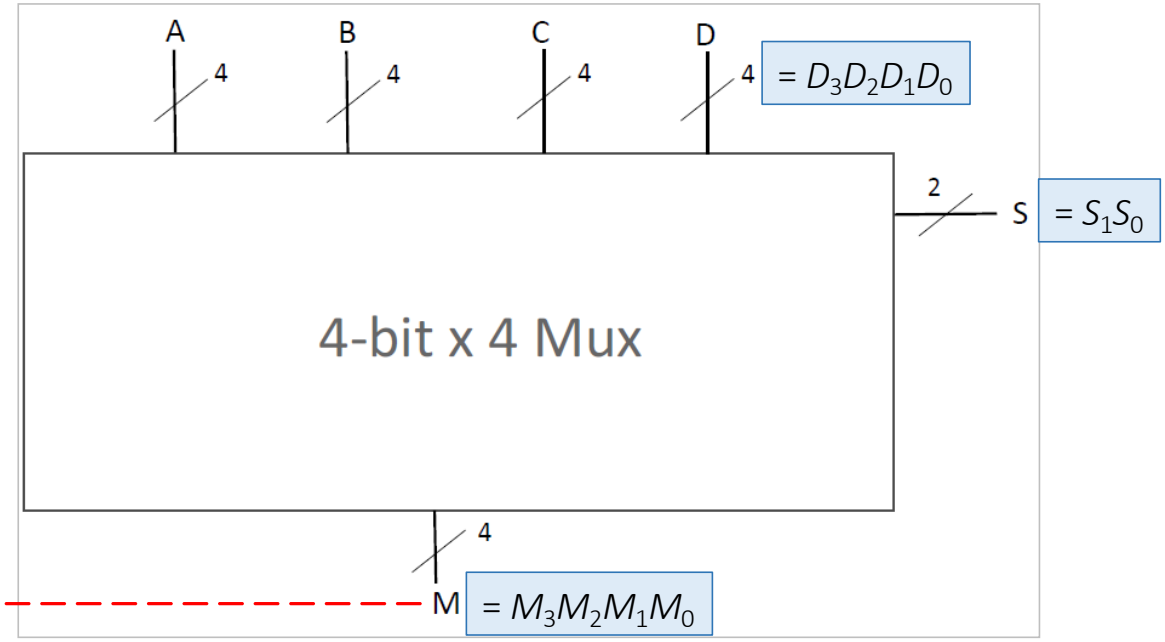
P.ej., para $n = 2$ e inputs de 4 bits ($m = 4$), hay cuatro inputs de datos, A, B, C y D, dos inputs de control, S de dos bits ($S = S_1S_0$), y un output, M:

- dependiendo del valor de S,

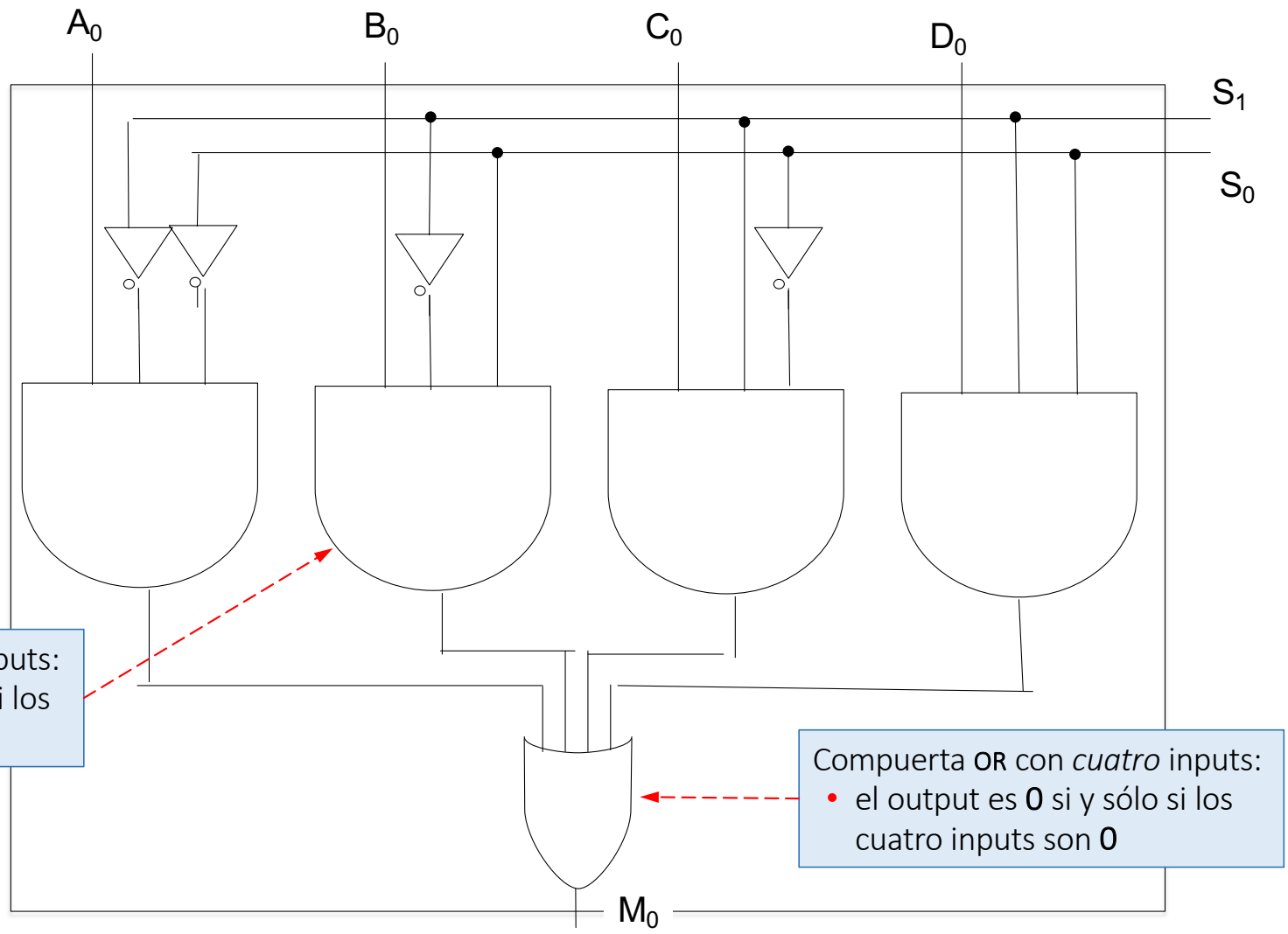
  … M será igual a A (si $S = 00$) o B (si $S = 01$) o C (si $S = 10$) o D (si $S = 11$)

- la próx. diapo. muestra el circuito para un bit

S_1	S_0	M
0	0	A
0	1	B
1	0	C
1	1	D



"doble click" al *mux* anterior para el caso del bit 0



Compuerta **AND** con *tres* inputs:
- el output es **1** si y sólo si los tres inputs son **1**

Compuerta **OR** con *cuatro* inputs:
- el output es **0** si y sólo si los cuatro inputs son **0**

Los circuitos anteriores:

- *enabler*
- multiplexor (*mux* o selector)

… son circuitos en que además de los inputs típicos *A*, *B*, … (los datos que estamos manejando)

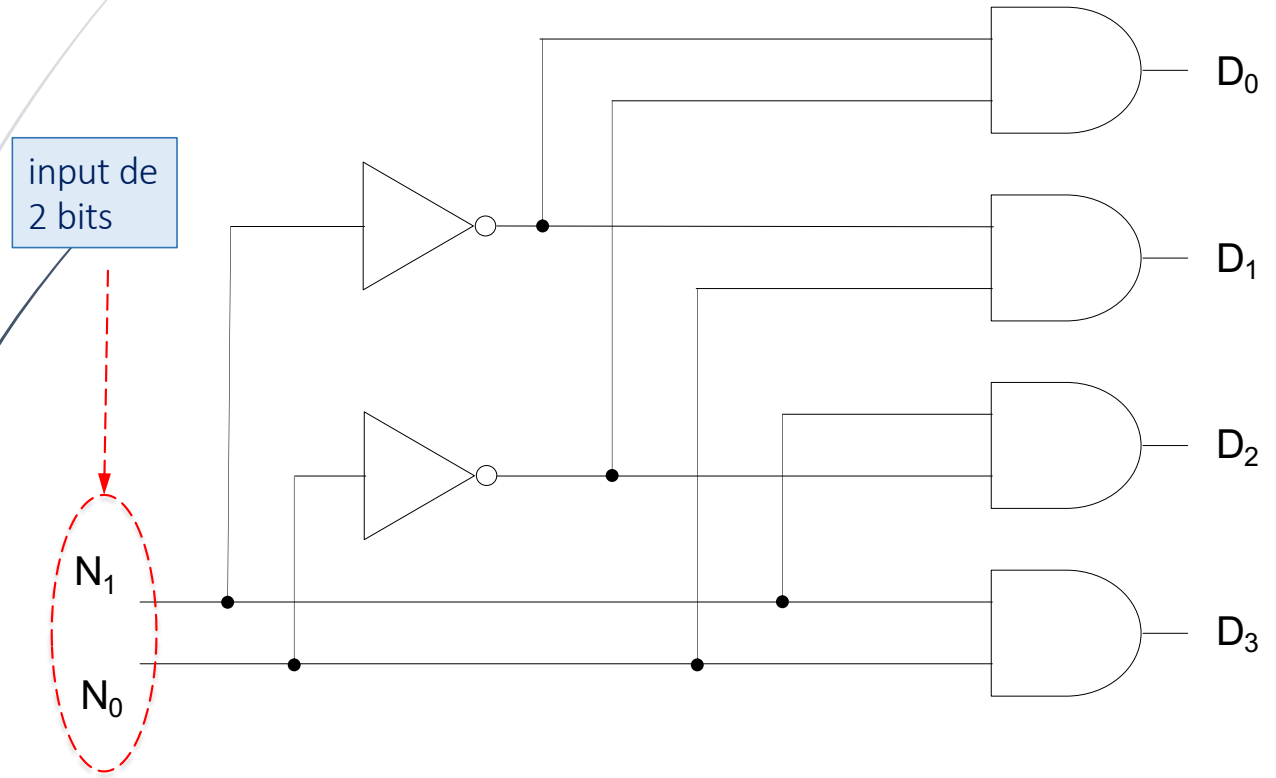
… hay **inputs** (o señales) **de control S**, que especifican cómo queremos manejar esos datos

(Esto nos sirve como introducción a la *ALU* —un circuito con varias señales de control— y, un poco más en general, a la CPU —varios circuitos, cada uno con sus propias señales de control)

Un **decodificador** recibe como input un número de *n* bits,

… y lo usa para poner en 1 exactamente una de 2^n líneas de salida:

- p.ej., para $n = 2$

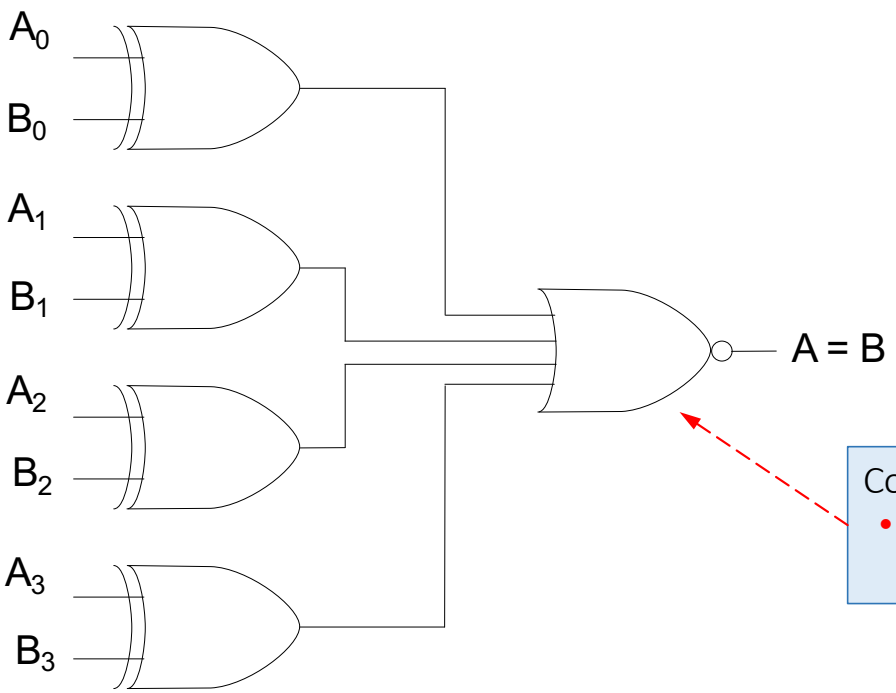


4 (= 2^2) líneas de salida:

- sólo una vale 1, dependiendo del valor de N_1N_0

Un **comparador** compara dos patrones de bits del mismo largo (los inputs) y produce un 1 (el output) si y sólo si los patrones son exactamente iguales (bit a bit); de lo contrario, produce un 0:

- p.ej., para el caso de patrones de 4 bits cada uno



Compuerta **NOR** con *cuatro* inputs:

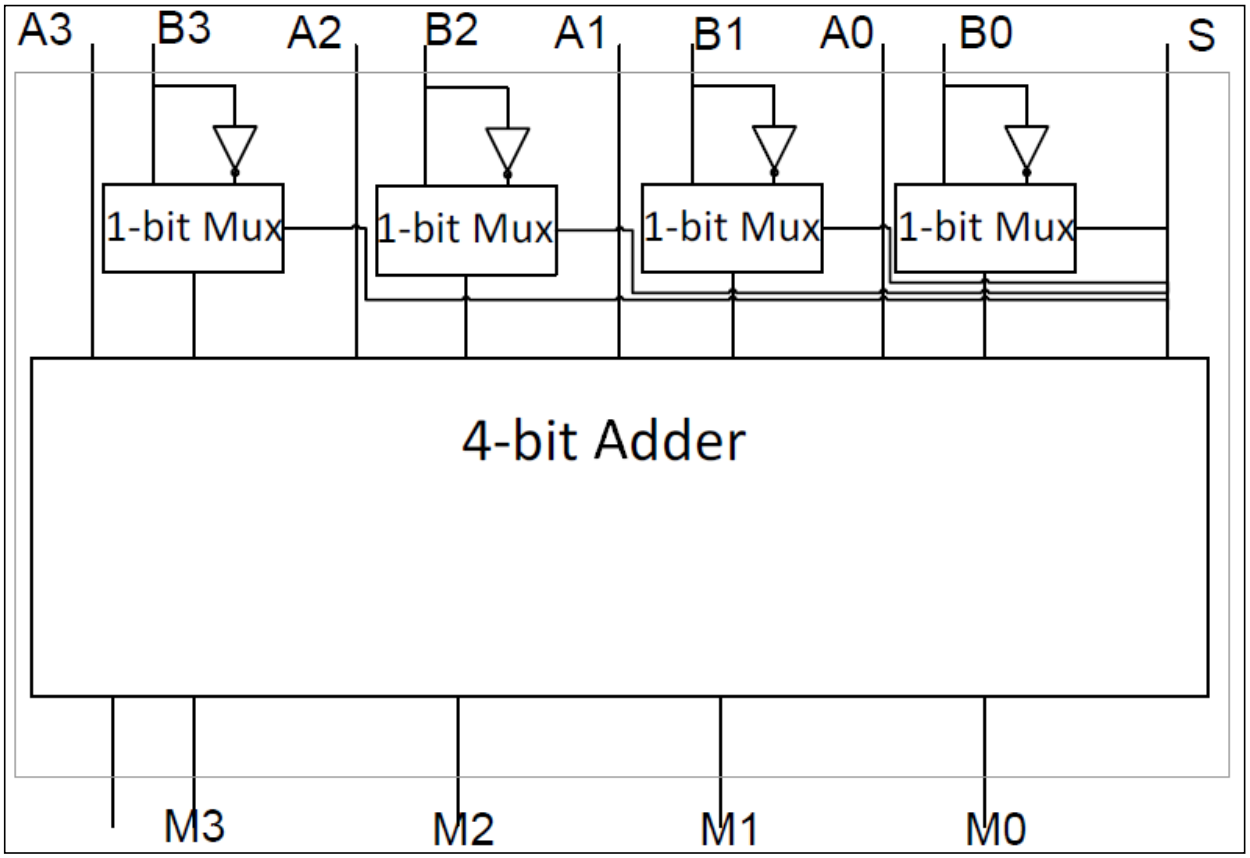
- el output es **1** si y sólo si los cuatro inputs son **0**

Nuestro circuito sumador/restador de 4 bits (próx. diap.) aprovecha el hecho de que el circuito restador es esencialmente el mismo que el circuito sumador,

- los inputs de datos, *A* y *B*, de 4 bits c/u
- 4 negadores (inversores), uno para c/u de los bits de B
- el sumador de 4 bits (diap. 2)
- 4 multiplexores de 1 bit para seleccionar entre el bit de *B* o el bit invertido de *B* (diap. 15)
- un input de control, *S*, que controla los multiplexores y es al mismo tiempo el input C_{in} del sumador del bit 0

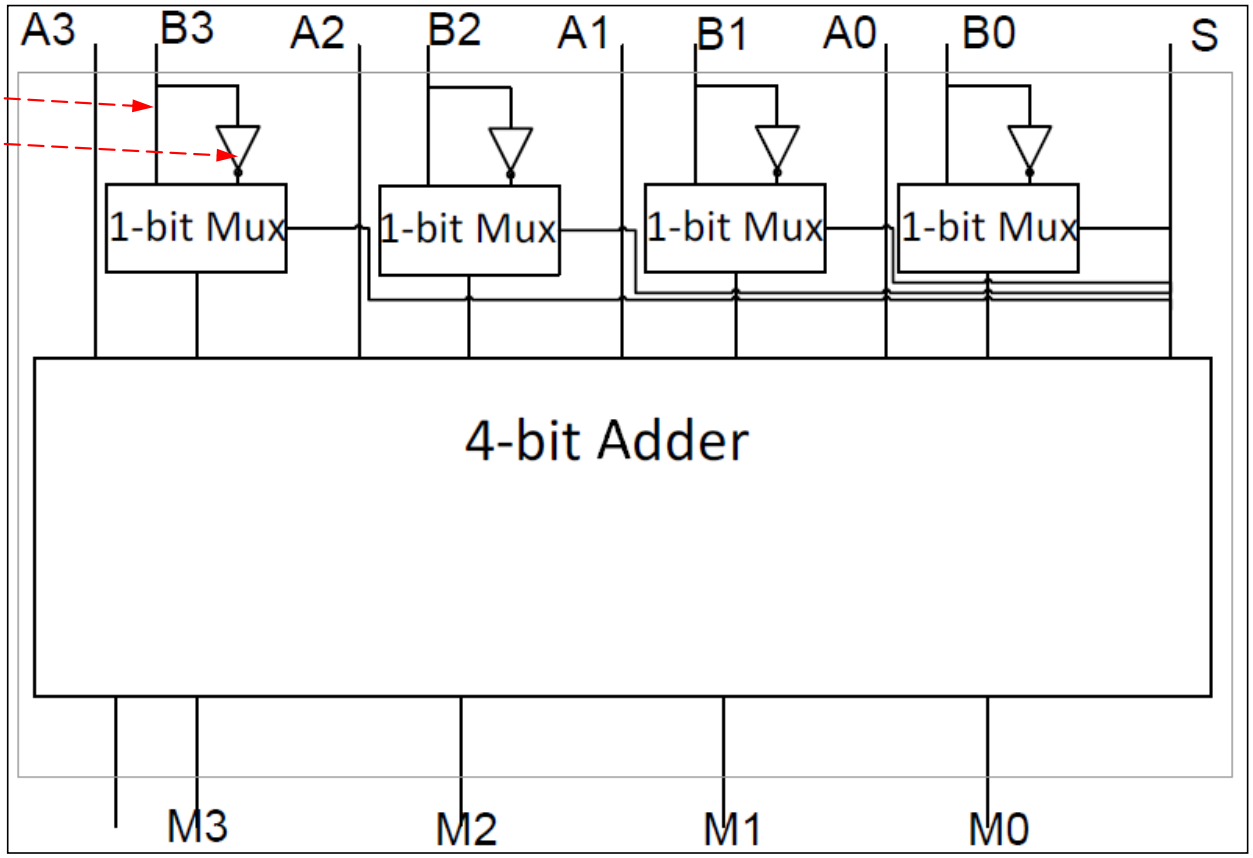
… sólo que recibe como uno de sus inputs el inverso aditivo del sustraendo *B* (en vez de *B* propiamente tal)

Sumador/restador
de 4 bits



Sumador/restador de 4 bits

El input *B* puede entrar
- "directo" –en el caso de una suma
- invertido –en el caso de una resta

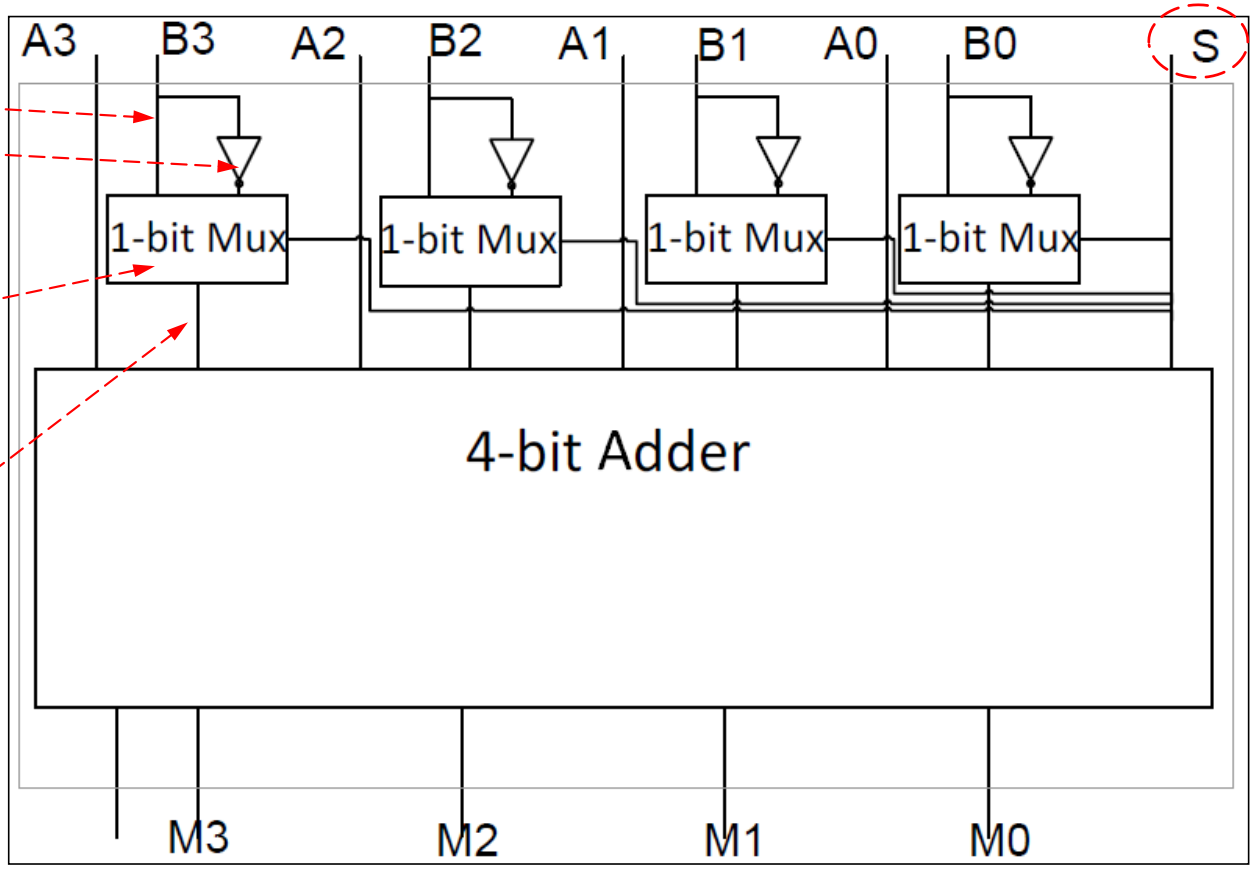


Sumador/restador de 4 bits

El input *B* puede entrar
- "directo" –en el caso de una suma
- invertido –en el caso de una resta

… por lo que ambas versiones pasan por un multiplexor controlado por el input de control *S* antes de entrar al circuito sumador:

- $S = 0 \Longrightarrow$ el *mux* selecciona la versión "directa" de *B*
- $S = 1 \Longrightarrow$ el *mux* selecciona la versión invertida de *B*

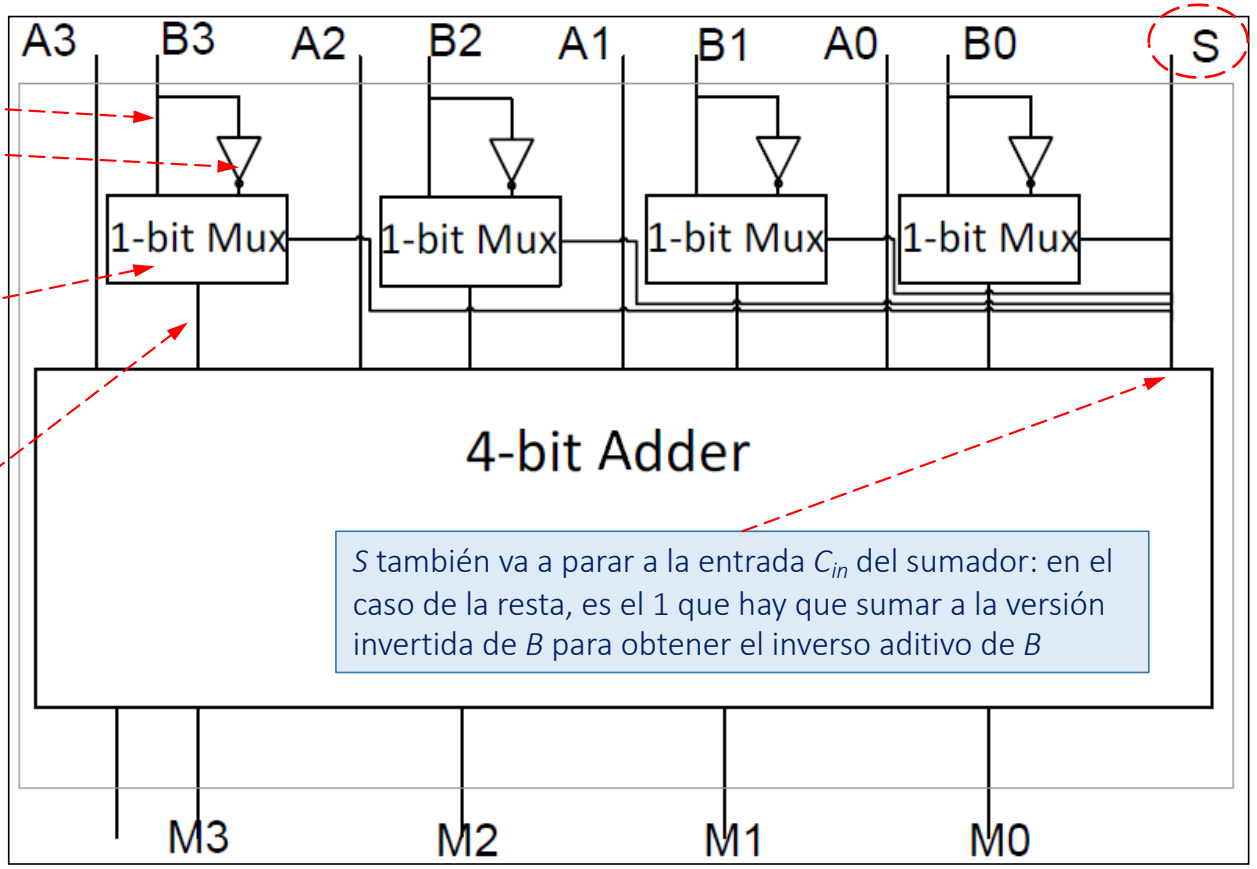


Sumador/restador de 4 bits

El input *B* puede entrar
- "directo" –en el caso de una suma
- invertido –en el caso de una resta

… por lo que ambas versiones pasan por un multiplexor controlado por el input de control *S* antes de entrar al circuito sumador:

- $S = 0 \Longrightarrow$ el *mus* selecciona la versión "directa" de *B*
- $S = 1 \Longrightarrow$ el *mux* selecciona la versión invertida de *B*

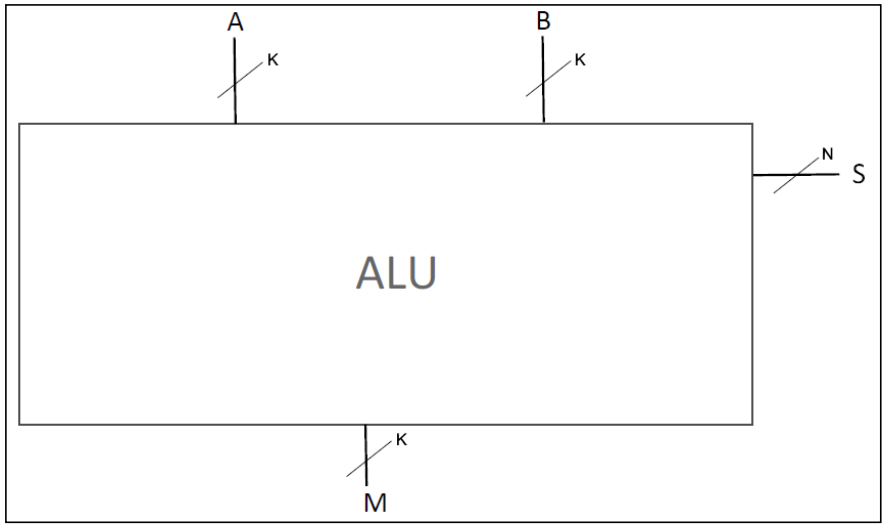


S también va a parar a la entrada C_{in} del sumador: en el caso de la resta, es el 1 que hay que sumar a la versión invertida de *B* para obtener el inverso aditivo de *B*

Todos los computadores tienen un circuito para realizar operaciones lógicas (p.ej., AND, OR) y aritméticas (p.ej., suma, resta) sobre dos operandos

→ la **unidad lógica aritmética** o ***ALU*** :

- operandos *A* y *B*, de *K* bits c/u
- resultado *M*, de *K* bits
- input de control *S*, de *N* bits, para seleccionar entre 2^N operaciones distintas



P.ej., el circuito sumador / restador de la diap. anterior correspondería a una *ALU* muy simple que sólo suma o resta operandos de $K = 4$ bits

⟹ sólo dos operaciones, por lo que *S* tiene sólo un bit ($N = 1$)

La ALU también realiza las operaciones lógicas básicas sobre sus inputs:

NOT AND OR XOR

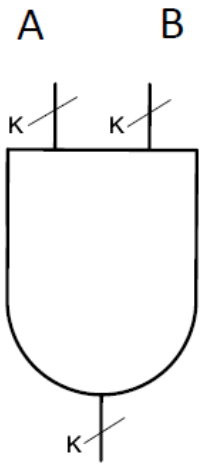
Son muy útiles por sí mismas y formando parte de otras operaciones más complejas

Se realizan bit a bit (*bitwise*)

K = 8 bits

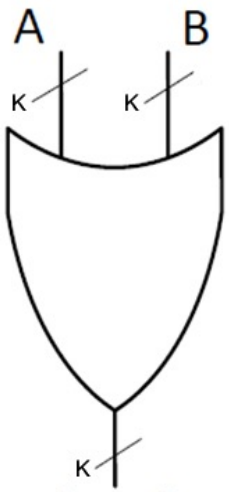
A = 0 1 1 0 0 1 0 1

B = 1 1 0 1 0 0 0 1



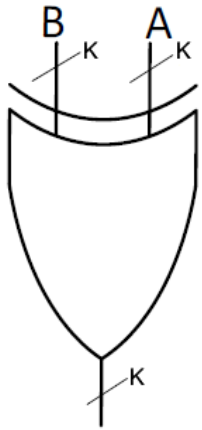
A and B

0 1 0 0 0 0 0 1



A or B

1 1 1 1 0 1 0 1



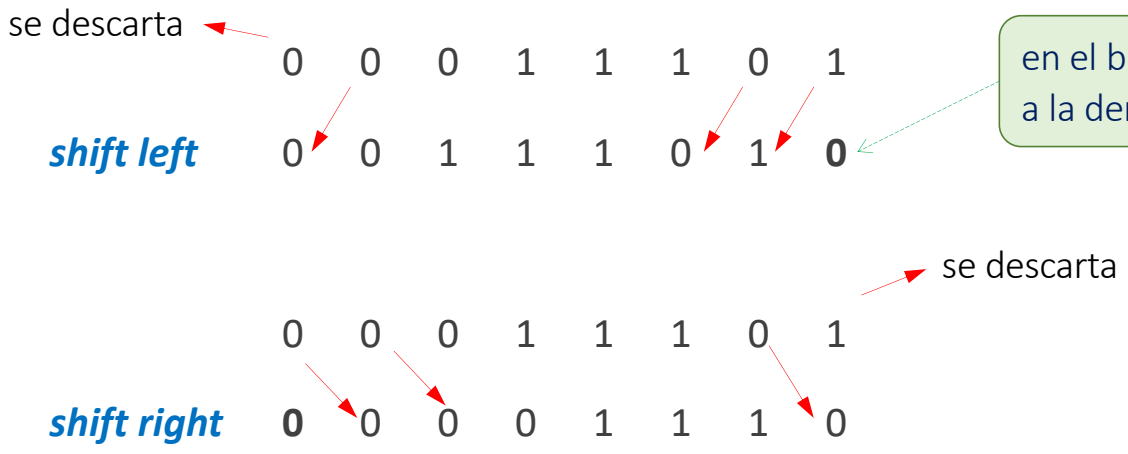
A xor B

1 0 1 1 0 1 0 0

not(A)

1 0 0 1 1 0 1 0

Las dos últimas operaciones que vamos a implementar en nuestra *ALU* básica son *shift left* y *shift right*



Las operaciones *shift* pueden parecer arbitrarias, pero son muy útiles

P.ej., podemos verlas como la **multiplicación por 2** (*shift left*)

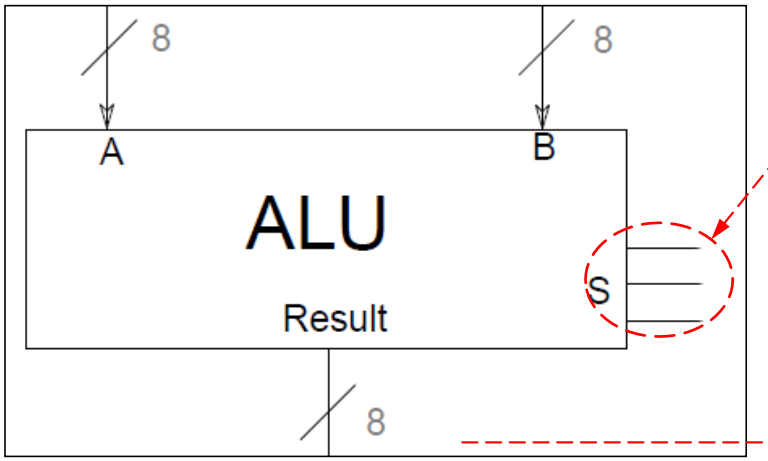
… y la **división** (entera) **por 2** (*shift right*)

… análogas, en base 10, a una multiplicación por 10 (se agrega un cero a la derecha de la secuencia de dígitos) y una división entera por 10 (se descarta el dígito de más a la derecha):

$47293 \times 10 = 472930$ $47293 / 10 = 4729$

(En *shift right*, la razón de "llenar" el bit de más a la izquierda con el mismo valor que había allí originalmente, es mantener el signo —positivo o negativo— del número representado por el patrón de bits)

ALU básica de 8 operaciones, dibujada esquemáticamente para dos inputs de datos de 8 bits (***A*** y ***B***), un input de control de 3 bits (***S***), y un output de 8 bits (***Result***)



S2	S1	S0	Result
0	0	0	Suma
0	0	1	Resta
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Xor
1	1	0	Shift left
1	1	1	Shift right

El output ***Result*** va a contener el resultado de la ejecución de una de las 8 operaciones, según la combinación de valores en los tres bits del input de control, $S_2\ S_1\ S_0$; p.ej.:

0 0 1 $\Longrightarrow$ *A* – *B* 0 1 1 $\Longrightarrow$ *A* OR *B* 1 1 1 $\Longrightarrow$ *shift right*(*A*)