Circuitos electrónicos digitales

Universidad de Sevilla

10/11/2025 16:09:35

Tema 6

Unidades aritméticas y lógicas

Usted es libre de copiar, distribuir y comunicar públicamente la obra y de hacer obras derivadas siempre que se cite la fuente y se respeten las condiciones de la licencia “Attribution-Share alike de Creative Commons”.

Texto completo de la licencia en:

http://creativecommons.org/licenses/by-sa/3.0/es

Introducción

- Los computadores pueden ejecutar operaciones aritméticas y lógicas.
- Una operación de cálculo en un computador puede ser realizado de dos formas:
  - *Hardware*: un circuito realiza completamente esa operación.
  - *Software*: un programa descompone esa operación en otras más elementales que son realizadas mediante hardware.

Introducción

- Hardware de cálculo numérico en los procesadores:
  - Todos los procesadores disponen de los componentes necesarios para sumar y restar enteros por hardware.
  - Algunos pueden realizar por hardware otras operaciones enteras más complejas como multiplicación o división entera.
  - Algunos incluso pueden realizar por hardware operaciones con números no enteros, algunas tan complejas como exponenciales, logaritmos o funciones trigonométricas.

Introducción

La aritmética de un computador digital tiene ciertas peculiaridades:

- La base del sistema de numeración suele ser 2 (binaria).
- El código empleado para representar conjuntamente enteros mayores y menores que cero (notación con signo) normalmente no es signo-magnitud.
- A menudo, el número de bits disponible para representar un número tiene una cota predeterminada, lo que limita la precisión y el rango de valores representables.

Aritmética binaria: operaciones básicas

Suma aritmética

$$
\begin{array}{r}
1\ 0\ 0\ 1\ 1\ 0 \\
+\quad\ \ 1\ 1\ 0\ 1 \\
\hline
\end{array}
$$

Aritmética binaria: operaciones básicas

Suma aritmética

0	1	1	0	0		Acarreo (carry)
1	0	0	1	1	0	Sumando 1
+		1	1	0	1	Sumando 2
1	1	0	0	1	1	Suma

Aritmética binaria: operaciones básicas

Resta

$$
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 1 & 0 \\
- & & 1 & 1 & 0 & 1 \\
\hline
\end{array}
$$

Aritmética binaria: operaciones básicas

Resta

1	0	0	1	1	0	Minuendo
-		1	1	0	1	Sustraendo
1	1	0	0	1		Préstamo (borrow)
0	1	1	0	0	1	Resta

Aritmética binaria: operaciones básicas

Multiplicación

$$
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 1 & 0 \\
X & & 1 & 1 & 0 & 1 \\
\hline
\end{array}
$$

Aritmética binaria: operaciones básicas

Multiplicación (algoritmo de sumas y desplazamientos, no óptimo)

			1	0	0	1	1	0	Multiplicando
			X		1	1	0	1	Multiplicador
			1	0	0	1	1	0	
		0	0	0	0	0	0		
	1	0	0	1	1	0			
1	0	0	1	1	0				
1	1	1	1	0	1	1	1	0	Multiplicación

Aritmética binaria: operaciones básicas

División

1 0 0 1 1 0 | 1 1 0 1

Aritmética binaria: operaciones básicas

División (algoritmo no óptimo)

```
Dividendo 1  0  0  1  1  0          | 1  1  0  1  Divisor
                         ←          |-----------------------
            -    1  1  0  1          1  0     Cociente
          ----------------
               0  1  1  0  0
          -       0  0  0  0
          ----------------
                  1  1  0  0     Resto
```

Operador módulo: definición

- Sea $x \in \mathbb{Z}$ y sea $n \in \mathbb{N}$, se define $x \bmod n$ como el número $r \in \mathbb{Z}_n$ tal que x-r es múltiplo de n.
- Es fácil demostrar que $x \bmod n$ es único.
- Ejemplos:

  23 mod 5 = 3

  -30 mod 8 = 2

  4 mod 20 = 4

  -4 mod 20 = 16

- El operador módulo suele tener la menor precedencia, es decir, $a+b \bmod m = (a+b) \bmod m$, $a-b \bmod m = (a-b) \bmod m$, $a \times b \bmod m = (a \times b) \bmod m$.

Operador módulo: propiedades

- Si $x \geq 0$, $x \bmod n$ es el resto de dividir x entre n.
- Si x es múltiplo de n, $x \bmod n=0$.
- Si $x \in \mathbb{Z}_n$, $x \bmod n=x$.
- $(f+g) \bmod n=[(f \bmod n)+g] \bmod n=[f+(g \bmod n)] \bmod n=$

  $=[(f \bmod n)+(g \bmod n)] \bmod n$
- $(f-g) \bmod n=[(f \bmod n)-g] \bmod n=[f-(g \bmod n)] \bmod n=$

  $=[(f \bmod n)-(g \bmod n)] \bmod n$
- $(f \times g) \bmod n=[(f \bmod n) \times g] \bmod n=[f \times (g \bmod n)] \bmod n=$

  $=[(f \bmod n) \times (g \bmod n)] \bmod n$

Operador módulo: ejercicios

- Calcule 83141592 mod n para los siguientes valores de n: 10^9, 10^8, 10^7, 10^6, 10^5, 10^4, 10^3, 10^2 y 10^1.
- Represente el número $101100110_{(2}$ mod n en base 2 para los siguientes valores de n: 2^{10}, 2^9, 2^8, 2^7, 2^6, 2^5, 2^4, 2^3, 2^2 y 2^1.
- Represente el número $2143157_{(8}$ mod n en base 8 para los siguientes valores de n: 8^8, 8^7, 8^6, 8^5, 8^4, 8^3, 8^2 y 8^1.
- Calcule 236813×564157 mod 10.
- Represente en base 2 el número $111001101_{(2} \times 101100111_{(2}$ mod 2^2.

notaciones con signo

Los códigos binarios para representar número enteros se dividen en dos tipos:

- Sin signo: No permiten representar valores menores que cero.
  - Base 2
  - Gray
- Con signo: Permiten representar valores menores que cero.
  - Notación Signo-Magnitud
  - Notación Complemento a 1
  - Notación Complemento a 2
  - Notación Exceso

notación signo-magnitud

- El número que representa una cadena de bits A en notación signo-magnitud lo notamos $A_{(S\text{-}M}$
- Con n bits los números representables son los enteros en el intervalo $[-(2^{n-1} – 1), 2^{n-1} – 1]$ (un total de $2^n – 1$)
- Para representar un número $X \geq 0$ se toma una cadena A tal que $A_{(2}=X$. Ejemplo: “01011010”$_{(S\text{-}M}=90$
- Para representar un número $X \leq 0$ se toma una cadena A tal que $A_{(2}=2^{n-1}+|X|$. Ejemplo: “11011010”$_{(S\text{-}M}=-90$
- Cualquier cadena con los n-1 bits menos significativos iguales a 0 representa el número cero (“0000” y “1000” para n=4)

notación signo-magnitud

Cuando se representa un número con una cadena A en notación signo magnitud, al bit más significativo de A se le llama bit de signo ya que si vale cero entonces $A_{(S\text{-}M} \geq 0$ y si vale uno entonces $A_{(S\text{-}M} \leq 0$. A los bits restantes se les denomina magnitud ya que en notación base 2 sin signo representan el valor absoluto de $A_{(S\text{-}M}$

	Signo	Magnitud						
+ 90	→ 0	1	0	1	1	0	1	0
- 90	→ 1	1	0	1	1	0	1	0

notación complemento a 1

Sea A una cadena de n bits, el complemento a 1 de A, Ca1(A), es otra cadena de n bits tal que $A_{(2}+Ca1(A)_{(2}=2^n-1$

Propiedades de Ca1:

- Ca1(Ca1(A))=A
- Ca1(A) puede obtenerse complementando los bits de A, es decir, Ca1(A)=$\overline{A}$
- Ejemplo: Ca1(“01011010”)=”10100101”

notación complemento a 1

- El número que representa una cadena de bits A en notación complemento a 1 lo notamos $A_{(Ca1}$
- Con n bits los números representables son los enteros en el intervalo $[-(2^{n-1} – 1), 2^{n-1} – 1]$ (un total de $2^n – 1$)
- Para representar un número $X \geq 0$ se toma una cadena A tal que $A_{(2}=X$. Ejemplo: “01011010”$_{(Ca1}$=90
- Para representar un número $X \leq 0$ se toma una cadena A tal que $Ca1(A)_{(2}=|X|$. Ejemplo: “10100101”$_{(Ca1}$=-90
- Cualquier cadena con todos los bits iguales representa el número cero (“0000” y “1111” para n=4)
- Al bit más significativo de A se le llama bit de signo ya que si vale 0 entonces $A_{(Ca1} \geq 0$ y si vale 1 entonces $A_{(Ca1} \leq 0$.

notación complemento a 2

Sea A una cadena de n bits, el complemento a 2 de A, Ca2(A), es otra cadena de n bits tal que

$$Ca2(A)_{(2}=(-A_{(2}) \bmod 2^n$$

Propiedades de Ca2:

- Ca2(Ca2(A))=A
- Si $A_{(2}=0$ entonces Ca2(A)=A
- Si $A_{(2}\neq 0$ entonces $A_{(2}+Ca2(A)_{(2}=2^n$

$\Big\}\ A_{(2}+Ca2(A)_{(2} \bmod 2^n=0$

- $Ca2(A)_{(2}=\overline{A}_{(2}+1 \bmod 2^n$
- Ca2(A) puede obtenerse complementando los bits a la izquierda del primer uno de A empezando por la derecha.
- Ejemplo: Ca2("01011100")="10100100"

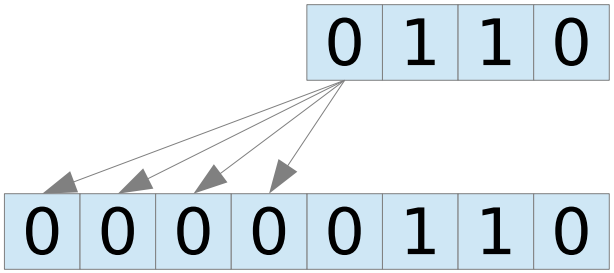
notación complemento a 2

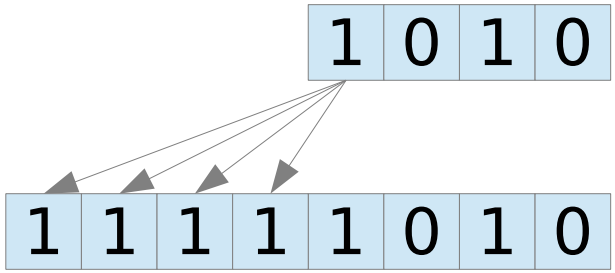
- El número que representa una cadena de bits A en notación complemento a 2 lo notamos $A_{(Ca2}$
- Con n bits los números representables son los enteros en el intervalo $[-2^{n-1}, 2^{n-1} - 1]$ (un total de 2^n)
- Para representar un número X se toma una cadena A tal que $A_{(2}=X \bmod 2^n$. Ejemplo: “10100110”$_{(Ca2}$=-90
- Propiedad: $A_{(Ca2}=2^0A_0+2^1A_1+2^2A_2+\ldots+2^{n-2}A_{n-2} - 2^{n-1}A_{n-1}$
- Propiedad: Si $A_{(Ca2} \neq -2^{n-1}$ entonces $Ca2(A)_{(Ca2}=-A_{(Ca2}$
- Propiedad: $-A_{(Ca2}=\overline{A}_{(Ca2}+1$
- Sólo la cadena con todos los bits iguales a 0 representa el número cero (“0000” para n=4)
- Al bit más significativo de A se le llama bit de signo ya que si vale 0 entonces $A_{(Ca2} \geq 0$ y si vale 1 entonces $A_{(Ca2}<0$.

notación complemento a 2

- Si S es el bit más significativo de A, $\{S,A\}_{(Ca2}=A_{(Ca2}$, donde {S,A} denota la concatenación de S y A.

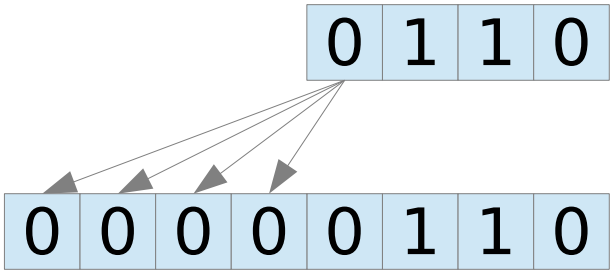
- Esto permite encontrar una cadena más larga que A que en notación complemento a 2 represente el mismo número que A usando un procedimiento denominado *extensión de signo*.

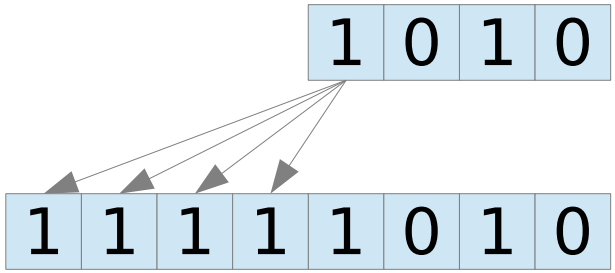




notación complemento a 2

- ¿Como se realiza el proceso inverso de extensión de signo?
  - Se tiene una cadena A de n bits.
  - Se busca una cadena A’ de longitud m<n tal que $A'_{(Ca2}=A_{(Ca2}$
- La cadena A’ que buscamos existe si y sólo si A es la extensión de signo de A’, es decir, si se cumple:
  - Los n-m+1 bits más significativos de A son iguales.
  - A’ son los m bits menos significativos de A.





notación exceso

- Sea K un número entero, el número que representa una cadena de bits A en notación exceso K lo notamos $A_{(\text{exceso } K}$
- Con n bits los números representables son los enteros en el intervalo $[-K, 2^n-1-K]$ (un total de 2^n)
- Para representar un número X se toma una cadena A tal que $A_{(2}=X+K$. Ejemplo: “1100”$_{(\text{exceso } 8}=4$
- Para un valor fijo de n y K, sólo una cadena representa el número cero (“1000” para n=4 y K=8)
- Suele tomarse $K=2^{n-1}$. En tal caso al bit más significativo de A se le llama bit de signo ya que si dicho bit vale 1 entonces $A_{(\text{exceso } K}\geq 0$ y si vale 0 entonces $A_{(\text{exceso } K}<0$.

notaciones con signo

- Representación de números con signo, ejemplo 4 bits

	S-M	Ca1	Ca2	Exceso $2^3=8$
7	0111	0111	0111	1111
6	0110	0110	0110	1110
5	0101	0101	0101	1101
4	0100	0100	0100	1100
3	0011	0011	0011	1011
2	0010	0010	0010	1010
1	0001	0001	0001	1001
0	0000/1000	0000/1111	0000	1000
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8	-	-	1000	0000

Sumador de magnitud

- Un sumador de magnitud de *n* bits es un dispositivo digital con dos entradas *a* y *b* (sumandos) de *n* bits, una entrada *Cin* (entrada de acarreo) de 1 bit, una salida *S* (suma) de *n* bits y una salida *Cout* (salida de acarreo) de 1 bit.

- Funcionalidad: $\{Cout,S\}_{(2}=a_{(2}+b_{(2}+Cin_{(2}$, donde $\{Cout,S\}$ denota la concatenación de *Cout* y *S*.

- Propiedades:

  - $Cout=1$ si y sólo si $a_{(2}+b_{(2}+Cin_{(2}>2^n-1$

  - Si $Cout=0$ entonces $S_{(2}=a_{(2}+b_{(2}+Cin_{(2}$

  - $S_{(2}=(a_{(2}+b_{(2}+Cin_{(2}) \bmod 2^n$

Restador de magnitud

- Un restador de magnitud de *n* bits es un dispositivo digital con dos entradas *a* y *b* (minuendo y sustraendo) de *n* bits, una entrada *Bin* (entrada de préstamo) de 1 bit, una salida *R* (resta) de *n* bits y una salida *Bout* (salida de préstamo) de 1 bit.

- Funcionalidad: $\{Bout,R\}_{(2}=(a_{(2}-b_{(2}-Bin_{(2})\bmod 2^{n+1}$, es decir, $\{Bout,R\}_{(Ca2}=a_{(2}-b_{(2}-Bin_{(2}$

- Propiedades:

  - $Bout=1$ si y sólo si $a_{(2}-b_{(2}-Bin_{(2}<0$

  - Si $Bout=0$ entonces $R_{(2}=a_{(2}-b_{(2}-Bin_{(2}$

  - $R_{(2}=(a_{(2}-b_{(2}-Bin_{(2})\bmod 2^{n}$

Sumadores con signo: Sumador en Ca2

- Sea un sumador de magnitud de *n* bits con entradas *a, b*, *Cin* y salidas *S*, *Cout*, si $a_{(Ca2}+b_{(Ca2}+Cin_{(2}$ es representable en complemento a 2 con *n* bits entonces $S_{(Ca2}=a_{(Ca2}+b_{(Ca2}+Cin_{(2}$.

- Puede ver la demostración en los apéndices del tema.

- Conclusión: un sumador en Ca2 es funcionalmente idéntico a un sumador de magnitud.

Sumadores con signo: Sumador en Ca2

Tabla de verdad de sumador magnitud/CA2 de 4 bits

		$B_{(CA2}$	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
		$B_{(2}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$A_{(CA2}$	$A_{(2}$	A\B	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0	0000	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1	1	0001	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000
2	2	0010	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001
3	3	0011	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010
4	4	0100	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011
5	5	0101	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100
6	6	0110	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101
7	7	0111	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110
-8	8	1000	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
-7	9	1001	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000
-6	10	1010	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
-5	11	1011	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
-4	12	1100	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
-3	13	1101	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
-2	14	1110	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101
-1	15	1111	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110

S=A+B

Restador en Ca2

- Sea un restador de magnitud de *n* bits con entradas *a* (minuendo), *b* (sustranedo), *Bin* y salidas *R*, *Bout*, si $a_{(Ca2}$-$b_{(Ca2}$-$Bin_{(2}$ es representable en complemento a 2 con *n* bits entonces $R_{(Ca2}=a_{(Ca2}$-$b_{(Ca2}$-$Bin_{(2}$.

- Puede ver la demostración en los apéndices del tema.

- Conclusión: un restador en Ca2 es funcionalmente idéntico a un restador de magnitud.

Aritmética binaria

- La notación entera sin signo más utilizada para realizar cálculos aritméticos es la base 2.
  - Los sumadores/restadores de magnitud pueden implementarse de forma sencilla y eficiente.
- La notación entera con signo más utilizada para realizar cálculos aritméticos es Ca2.
  - Los sumadores/restadores de magnitud también son sumadores/restadores Ca2. Por eso los procesadores actuales usan las mismas instrucciones de suma/resta entera para ambas notaciones.
  - Sumar/restar en otras notaciones con signo es más complicado.

Ejemplo: Sumar en S-M

- Para obtener la suma de dos números en S-M:
  - Si ambos tienen el mismo signo
    - La magnitud del resultado coincide con la suma de las magnitudes.
    - El bit de signo del resultado es el mismo que el de cualquiera de los sumandos.
  - Si los números tienen distinto signo
    - La magnitud del resultado se obtiene restando la magnitud menor de la mayor.
    - El signo del resultado se corresponde con el signo que tenga la magnitud mayor.
- El circuito que realizase ese cálculo requeriría sumadores de magnitud, restadores de magnitud, comparadores, multiplexores, etc.

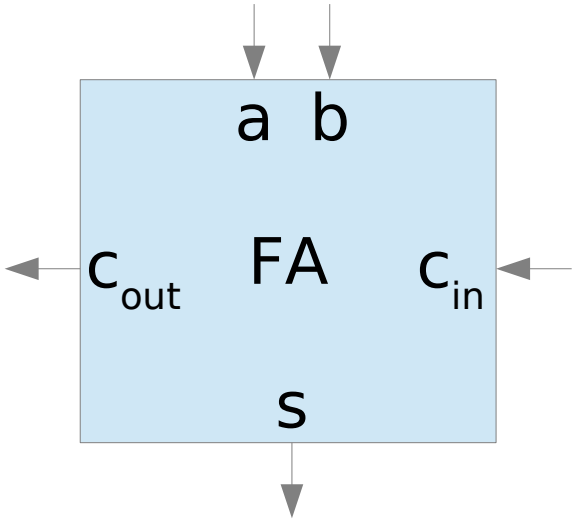
Ejemplo: Sumar en Ca1

- Puede hacerse en dos pasos:
  - Paso 1: Se alimenta un sumador de magnitud con los sumandos Ca1 y se anota el valor de las salidas.
  - Paso 2: Sumar el valor obtenido de C_{OUT} al resultado

1 1 0 0 0	acarreos		0 0 1 1 0	acarreos	
1 1 0 1 0	-5		1 0 0 1 1		
1 1 0 0 1	-6		+ 1		
1 0 0 1 1	-12	(MAL)	1 0 1 0 0	-11	(OK)

Circuitos sumadores básicos

- Sumador completo Full Adder (F.A.): Tiene la funcionalidad de un sumador de magnitud de un bit.



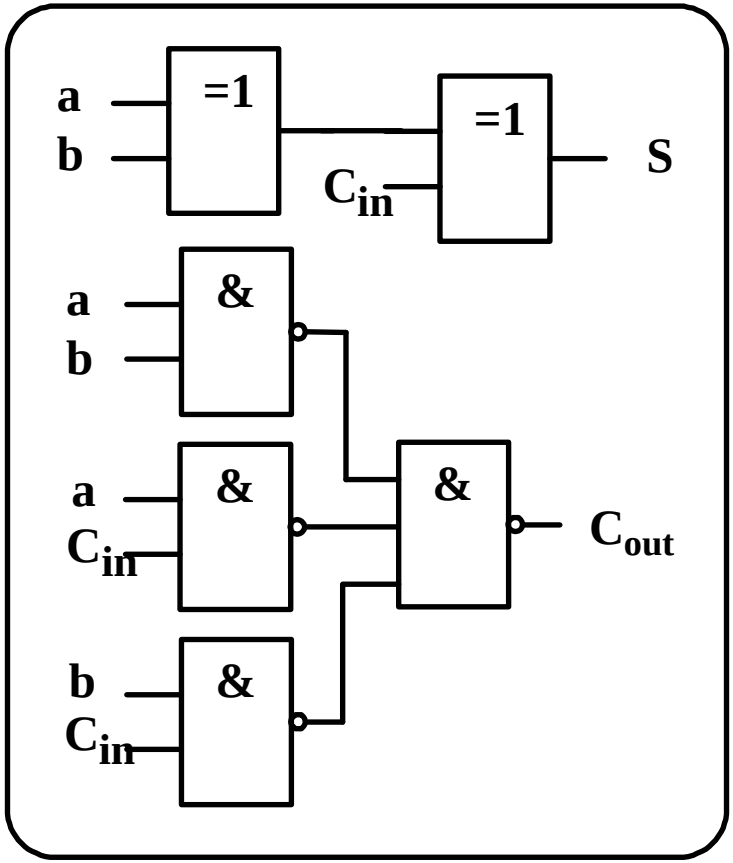
a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = a\,b + a\,c_{in} + b\,c_{in}$$

Circuitos sumadores básicos

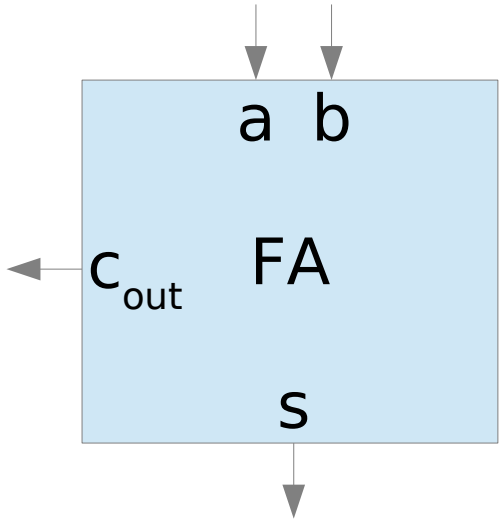
- Sumador completo Full Adder (FA)
  - Una implementación mediante puertas lógicas

$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = a\,b + a\,c_{in} + b\,c_{in}$$



Circuitos sumadores básicos

- Semisumador o Half Adder (H.A.): Carece de entrada de carry.



a	b	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

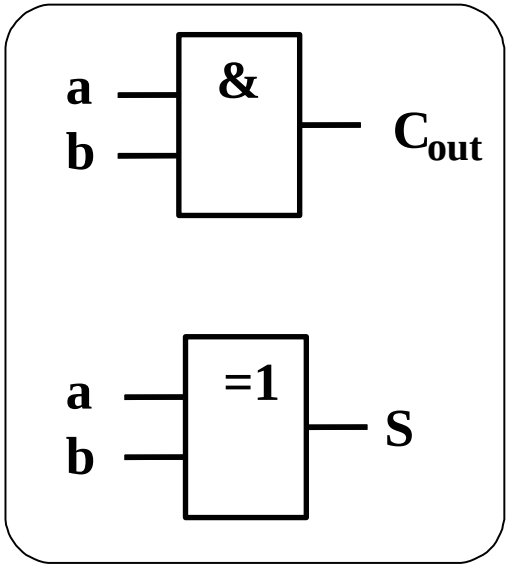
$s = a \oplus b$

$c_{out} = a\ b$

Circuitos sumadores básicos

- Semisumador o Half Adder (HA)
  - Una posible implementación mediante puertas lógicas

$s = a \oplus b$
$c_{out} = a\ b$



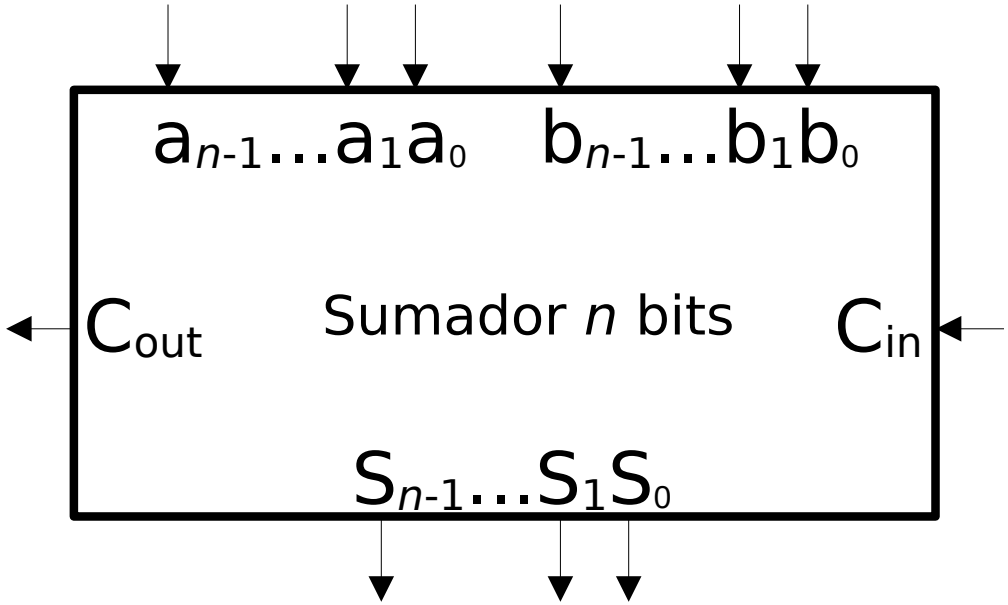
Circuitos sumadores básicos

- Descripción LDH de un sumador completo (FA)

```
//////////////////////////// Sumador FA ////////////////////////////////////////

module fa(
        input x, // primer operando
        input y, // segundo operando
        input cin, // acarreo de entrada
        output z, // salida de suma
        output cout  // acarreo de salida
        );

        assign z = x ^ y ^ cin;
        assign cout = x & y | x & cin | y & cin;

endmodule   // fa
```

Implementación de un sumador de *n* bits

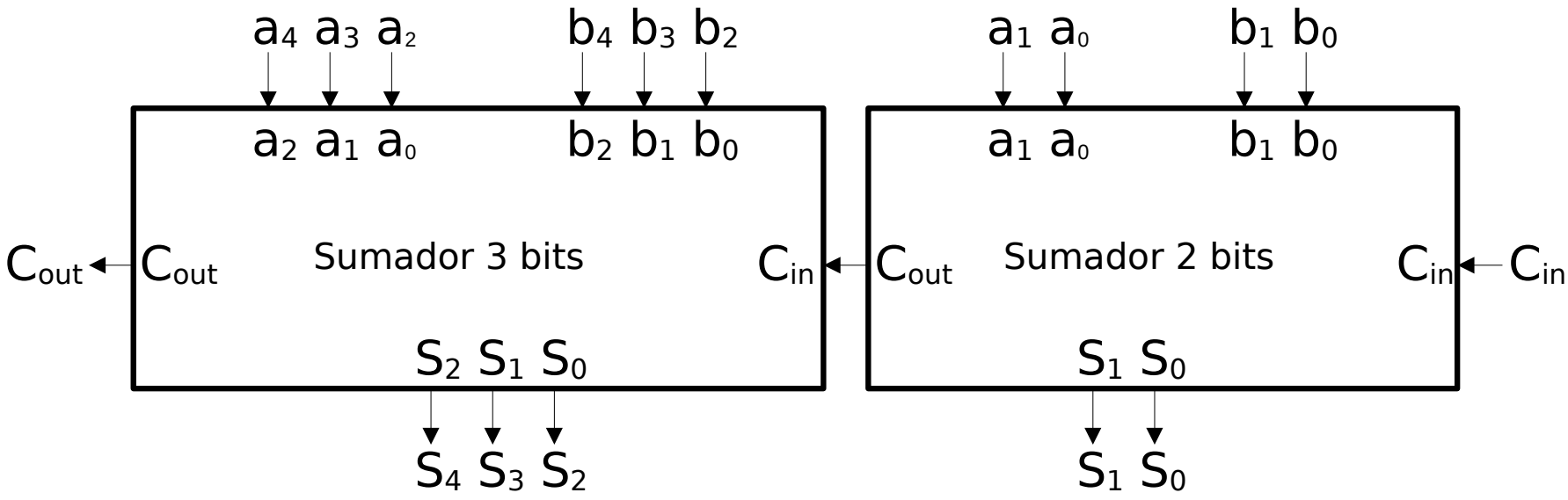
- Un sumador de magnitud/Ca2 de *n* bits con entrada de acarreo tiene esta estructura.



$$
\begin{array}{rcl}
C_{out}\ c_{n-1} \ldots c_2\ c_1\ c_0 & = & C_{in} \\
+\quad a_{n-1} \ldots a_2\ a_1\ a_0 & & \\
\quad b_{n-1} \ldots b_2\ b_1\ b_0 & & \\
\hline
S_{n-1} \ldots S_2\ S_1\ S_0 & &
\end{array}
$$

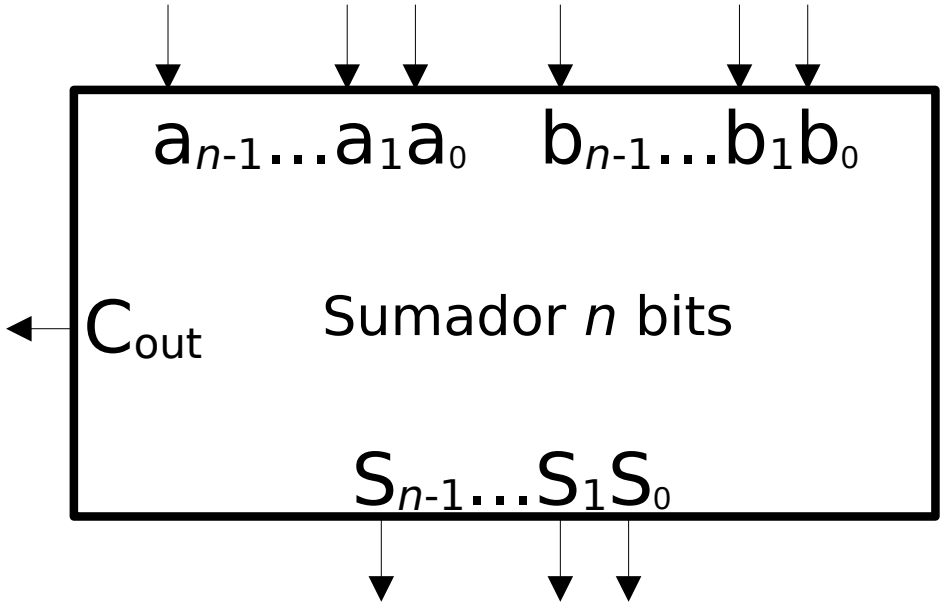
Implementación de un sumador de *n* bits

- Un sumador/restador puede construirse a partir de sumadores/restadores más pequeños encadenando sus entradas de carry/borrow. Por ejemplo, un sumador de 5 bits puede construirse encadenando un sumador de 2 bits son otro de tres bits.



Implementación de un sumador de *n* bits

- Un sumador de magnitud/Ca2 de *n* bits sin entrada de acarreo funciona como un sumador que tuviese la entrada de acarreo fija a 0. Tiene esta estructura.

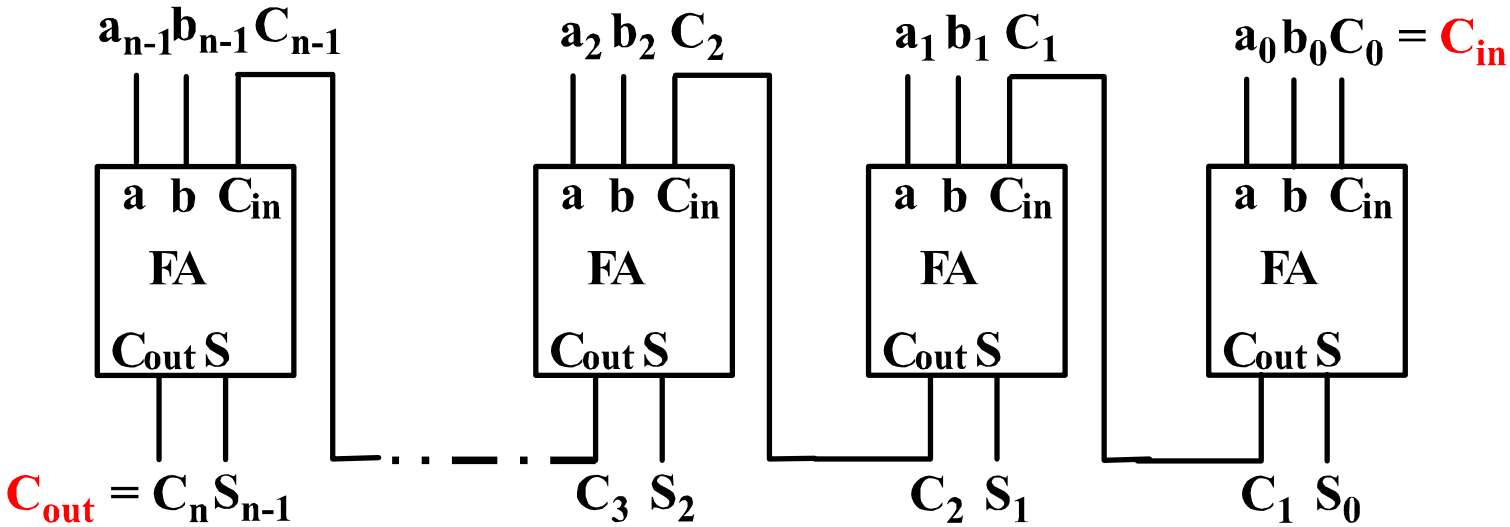


$$
\begin{array}{lcccccc}
C_{out} & c_{n-1} & \ldots & c_2 & c_1 & & \\
 & a_{n-1} & \ldots & a_2 & a_1 & a_0 & \\
+ & b_{n-1} & \ldots & b_2 & b_1 & b_0 & \\
\hline
 & S_{n-1} & \ldots & S_2 & S_1 & S_0 &
\end{array}
$$

Implementación de un sumador de *n* bits

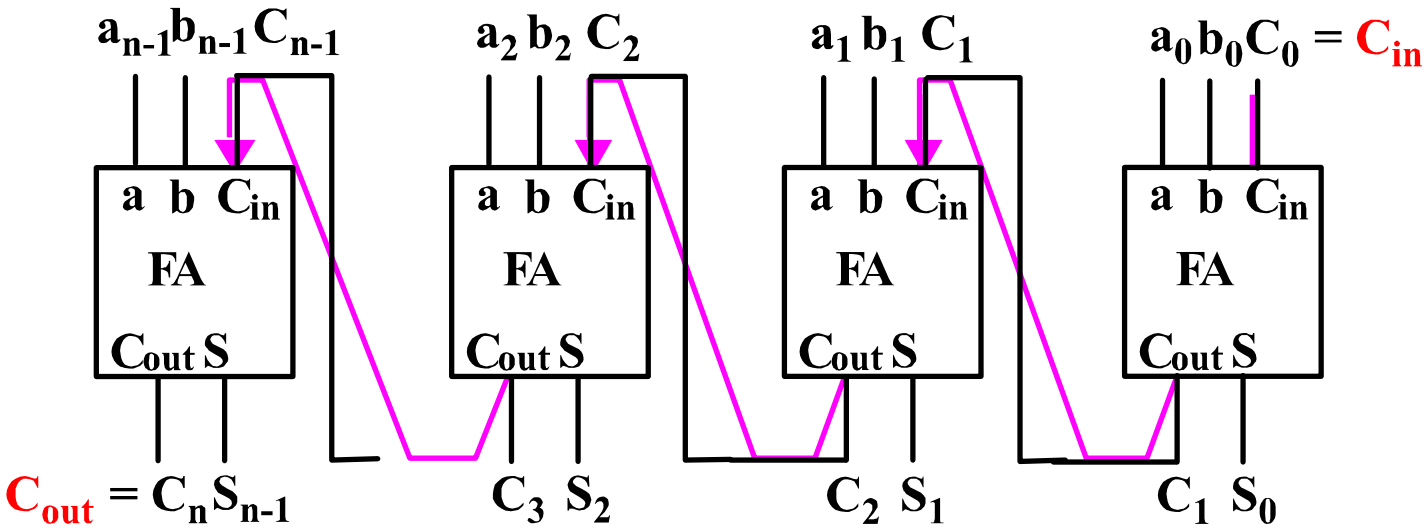
- La implementación conocida como sumador con acarreo serie (también llamada sumador de rizado o ripple adder) es la más intuitiva y tiene un coste razonablemente bajo.

- Se trata de un circuito modular construido con sumadores completos de un bit. Si no hay entrada de acarreo se usa un semisumador para generar el bit menos significativo.



Implementación de un sumador de *n* bits

- Es lento debido a la propagación serie del acarreo
- El tiempo que tarda en realizarse una suma crece linealmente con el número de bits

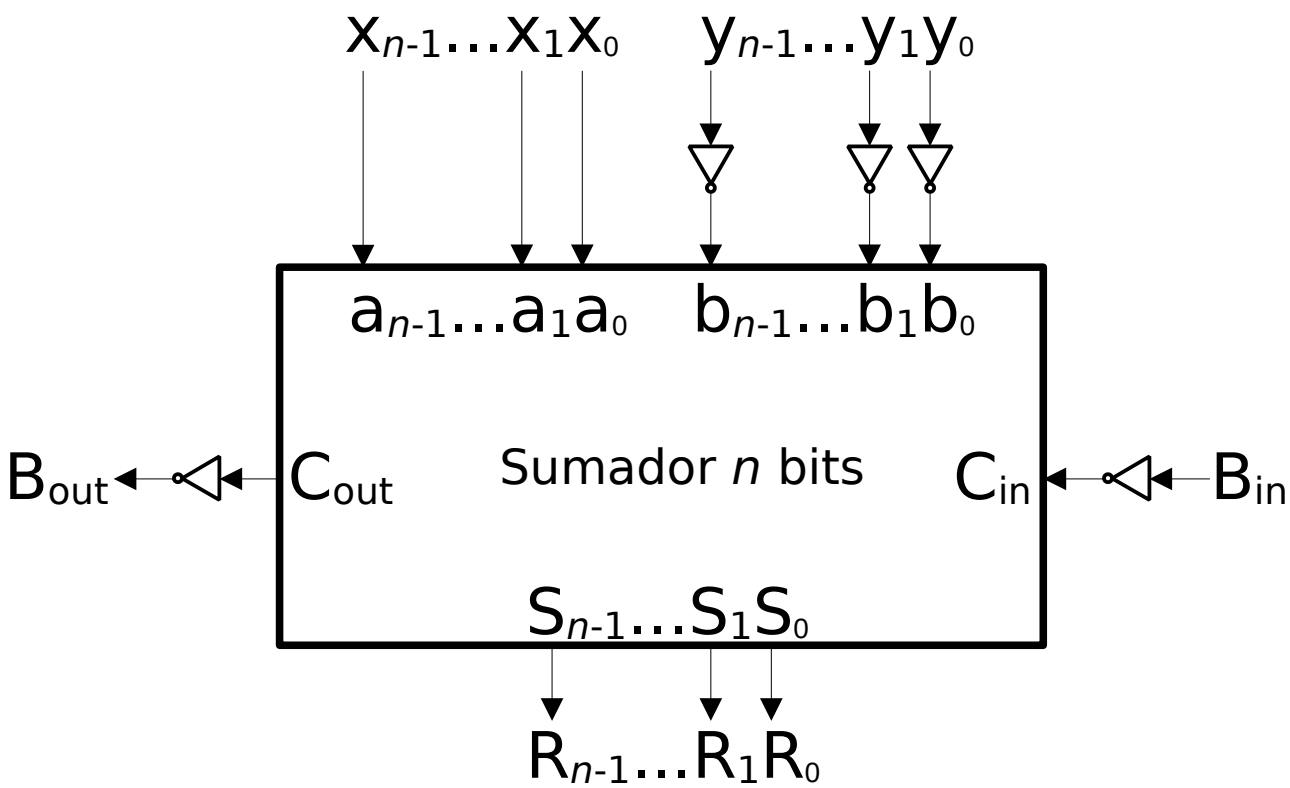


- Descripción LDH de un sumador de 8 bits

```
//////////////////////// // Sumador 8 bits con FA // ////////////////////////////////////
module adder8_e(
        input [7:0] a, // primer operando
        input [7:0] b, // segundo operando
        input cin, // acarreo de entrada
        output [7:0] s, // salida de suma
        output cout  // acarreo de salida
        );
/* Este sumador se construye mediante la conexión en cascada de 8
* sumadores completos (FA). Cada FA genera un bit del resultado.

* 'c' es una señal auxiliar para la conexión del acarreo de salida de
una
* etapa con el acarreo de salida de la etapa siguiente */
wire [7:1] c;

/* El acarreo de entrada del primer FA es el acarreo de entrada del
* módulo sumador */
        fa fa0 (a[0], b[0], cin, s[0], c[1]);
        fa fa1 (a[1], b[1], c[1], s[1], c[2]);
        fa fa2 (a[2], b[2], c[2], s[2], c[3]);
        fa fa3 (a[3], b[3], c[3], s[3], c[4]);
        fa fa4 (a[4], b[4], c[4], s[4], c[5]);
        fa fa5 (a[5], b[5], c[5], s[5], c[6]);
        fa fa6 (a[6], b[6], c[6], s[6], c[7]);
/* El acarreo de salida del último FA es el acarreo de salida del
 * módulo sumador */
        fa fa7 (a[7], b[7], c[7], s[7], cout);
endmodule   // adder8_e
```

Implementación de un restador de *n* bits

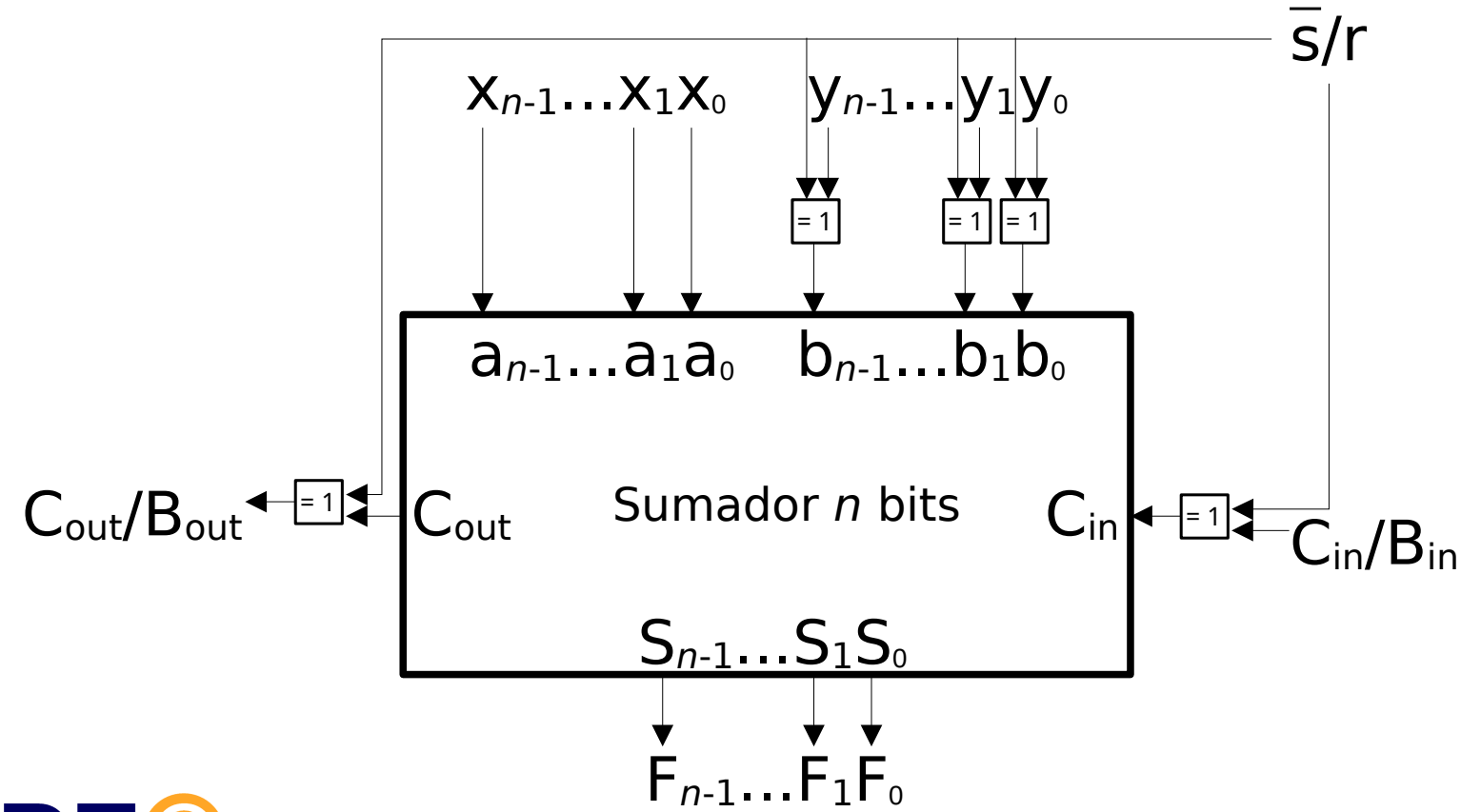
- Un restador de magnitud/Ca2 de *n* bits con señales de entrada *x*, *y*, *Bin* (minuendo, sustraendo, entrada de prestamo) y señales de salida *R*, *Bout* (resta, salida de prestamo) puede implementarse así:



- Puede ver la demostración en los apéndices del tema.

Implementación de un sumador-restador

- Conclusión: Este circuito funciona como un sumador de magnitud/Ca2 o como un restador de magnitud/Ca2 dependiendo del valor de la señal de control $\overline{s}/r$.



Desbordamiento en operaciones enteras

- A veces, a un circuito diseñado para realizar cálculos en alguna notación entera se le pide computar una operación cuyo resultado no es representable.
- Cuando eso ocurre se dice que se produce un desbordamiento.
- Ejemplo: Supongamos que un sumador/restador de 4 bits como el mostrado anteriormente se pone x=1111, y=0001, $\overline{s}/r$=0 y Cin/Bin=0. ¿Se produciría desbordamiento?

Desbordamiento en operaciones enteras

- A veces, a un circuito diseñado para realizar cálculos en alguna notación entera se le pide computar una operación cuyo resultado no es representable.
- Cuando eso ocurre se dice que se produce un desbordamiento.
- Ejemplo: Supongamos que un sumador/restador de 4 bits como el mostrado anteriormente se pone x=1111, y=0001, $\overline{s}/r$=0 y Cin/Bin=0. ¿Se produciría desbordamiento?
- La pregunta está incompleta porque ese dispositivo sirve para hacer operaciones en dos notaciones, y no se menciona cual de ellas se está usando:
  - Si es sin signo (base 2), sí hay desbordamiento
  - Si es con signo (Ca2), no hay desbordamiento

Salidas de estado

- En un circuito aritmético, las salidas empleadas para representar los valores que se pretenden calcular se denominan *salidas de datos*.

- El resto de salidas proporciona algún tipo de información sobre esos valores y se denominan *salidas de estado* o, si son de un sólo bit, *salidas de condición* o *flags*. Las típicas de un sumador/restador entero se denominan *C*, *V*, *S*, *N* y *Z*.

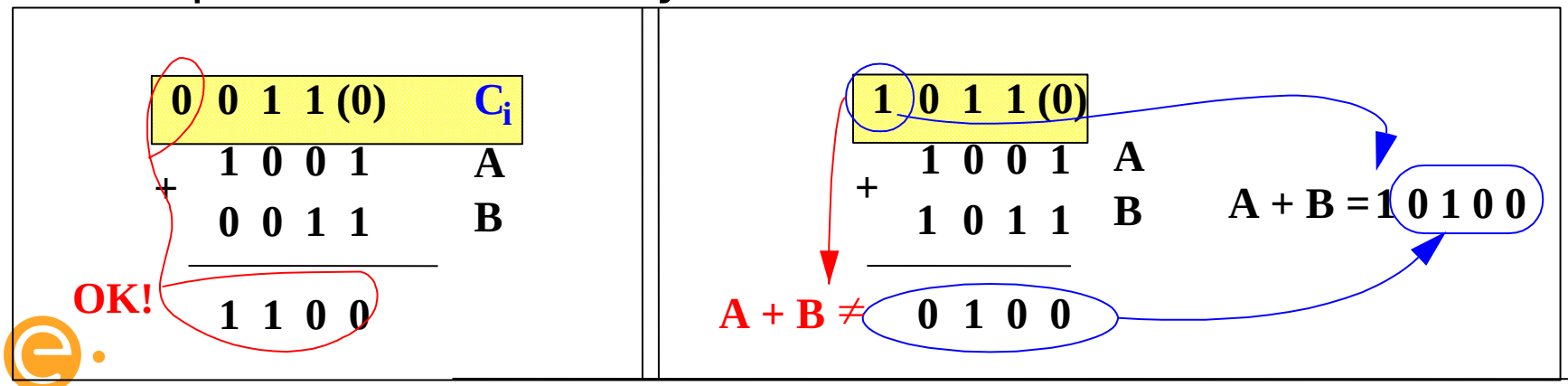
Salida de estado *C*

- Vale 1 si y sólo si hay desbordamiento usando notación sin signo.

- En nuestro sumador/restador de *n* bits es la salida *Cout/Bout* ya que:

  – En una suma es imposible que el resultado sea menor que 0, por tanto habrá desbordamiento si y sólo si es mayor que 2^n-1, es decir, si y sólo si *Cout*=1.

  – En una resta es imposible que el resultado sea mayor que 2^n-1, por tanto habrá desbordamiento si y sólo si es menor que 0, es decir, si y sólo si *Bout*=1.



Salida de estado *V*

- Del inglés *oVerflow*
- Vale 1 si y sólo si hay desbordamiento usando notación con signo.
- Por ejemplo, supongamos que añadimos esta salida a un sumador/restador de 4 bits como el del ejemplo y lo usamos para realizar las siguientes sumas:

```
   1001 = -7        11 00 = -4        0101 = +5
+  0101 = +5     +    0100 = +4     +  0100 = +4
   111 0 = -2        1 0000 = 0       0 1001 = -7

   11 00 = -4        0011  = +3        1001 = -7
+  1111  = -1     +  0100 = +4     +  1010 = -6
  1 1011  = -5       0111  = +7      1 0011  = +3
```

V=0 (no hay desbordamiento)

V=1 (hay desbordamiento)

Salida de estado *S*

- Del inglés *Sign*. No debe confundirse con la salida de datos homónima del sumador
- Vale 1 si y sólo si el resultado es menor que cero usando notación complemento a 2.
- Proporciona información válida incluso cuando hay desbordamiento.
- Si añadimos esta señal a nuestro sumador/restador, al usar notación complemento a 2 el resultado correcto será siempre $\{S,F\}_{(Ca2}$

1001 = -7	1100 = -4	0101 = +5
+ 0101 = +5	+ 0100 = +4	+ 0100 = +4
1110 = -2	10000 = 0	01001 = -7
1100 = -4	0011 = +3	1001 = -7
+ 1111 = -1	+ 0100 = +4	+ 1010 = -6
11011 = -5	0111 = +7	10011 = +3
V=0 (no hay desbordamiento)		*V*=1 (hay desbordamiento)

Salida de estado *N*

- Del inglés *Negative*
- Es igual al bit más significativo de la salida de datos.
- *N* y *S* pueden ser distintos, porque puede haber desbordamiento.

```
  1001 = -7        11 00 = -4        0101 = +5
+ 0101 = +5      +  0100 = +4      + 0100 = +4
  111 0 = -2       1 0000 = 0       0 1001 = -7
  N=S=1            N=S=0            N=1, S=0

  11 00 = -4       0011  = +3       1001 = -7
  1111  = -1     + 0100 = +4      + 1010 = -6
+1 1011  = -5      0111  = +7       1 0011  = +3
  N=S=1            N=S=0            N=0, S=1
```

V=0 (no hay desbordamiento)

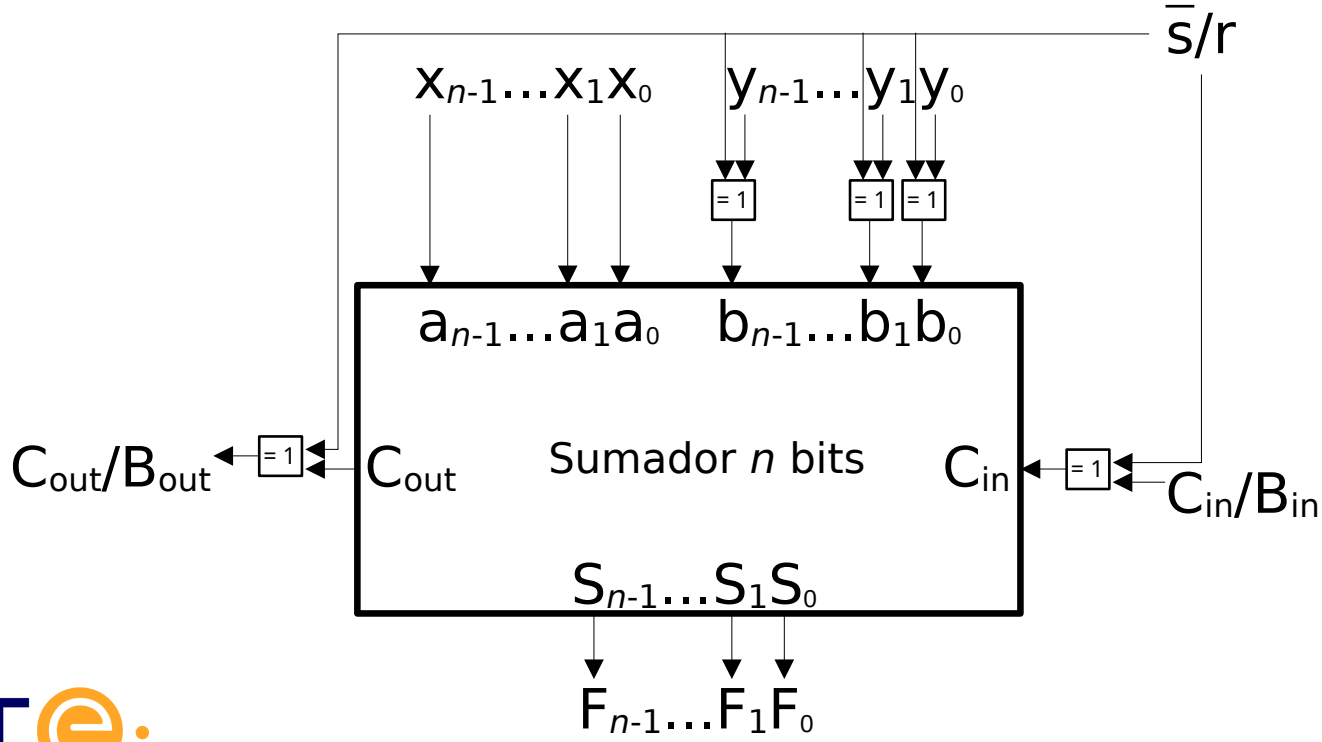
V=1 (hay desbordamiento)

Salida de estado *Z*

- Del inglés *Zero*
- Vale 1 si y sólo si todos los bits de la salida de datos valen 0.
- Si añadimos esta salida a un sumador/restador de *n* bits como el del ejemplo, independientemente de si se usa notación con signo nos dará la siguiente información:
  - Si Z=1 el valor del resultado mod 2^n es igual a cero.
  - Si Z=0 el valor del resultado mod 2^n es distinto de cero.

Salidas de estado

- ¿Como añadimos las salidas *C*, *V*, *S*, *N* y *Z* en el sumador/restador de *n* bits como el del ejemplo?
- *C* es Cout/Bout.
- *N* es F_{n-1}.
- *Z* es el NOR de los bits de F.

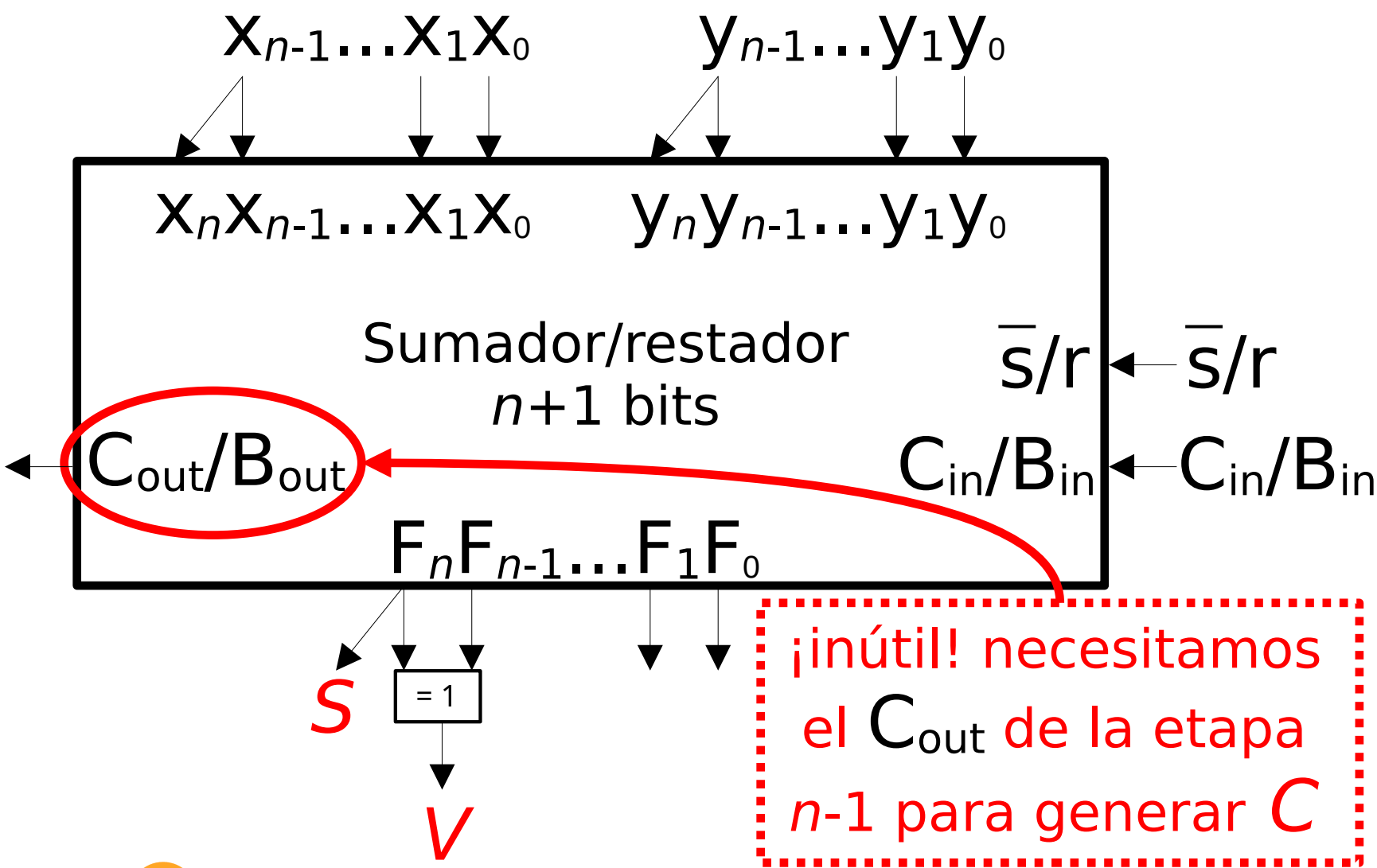


Salidas de estado

- Para implementar *V* y *S* consideremos lo siguiente:
  - Con n bits, los números representables en Ca2 son los enteros en el intervalo $[-2^{n-1},2^{n-1} - 1]$
  - El resultado más bajo sería $(-2^{n-1})+(-2^{n-1})=-2^{n}$
  - El resultado más alto sería $(2^{n-1} - 1)-(-2^{n-1})=2^{n} - 1$
  - Con n+1 bits, los números representables en Ca2 son los enteros en el intervalo $[-2^{n},2^{n} - 1]$
  - Por tanto, el resultado del sumador/restador de n bits puede siempre representarse en Ca2 con n+1 bits.
  - La representación correcta del resultado en Ca2 con n+1 bits podemos obtenerla alimentando un sumador/restador de n+1 bits con la extensión de signo de los operandos.

Salidas de estado

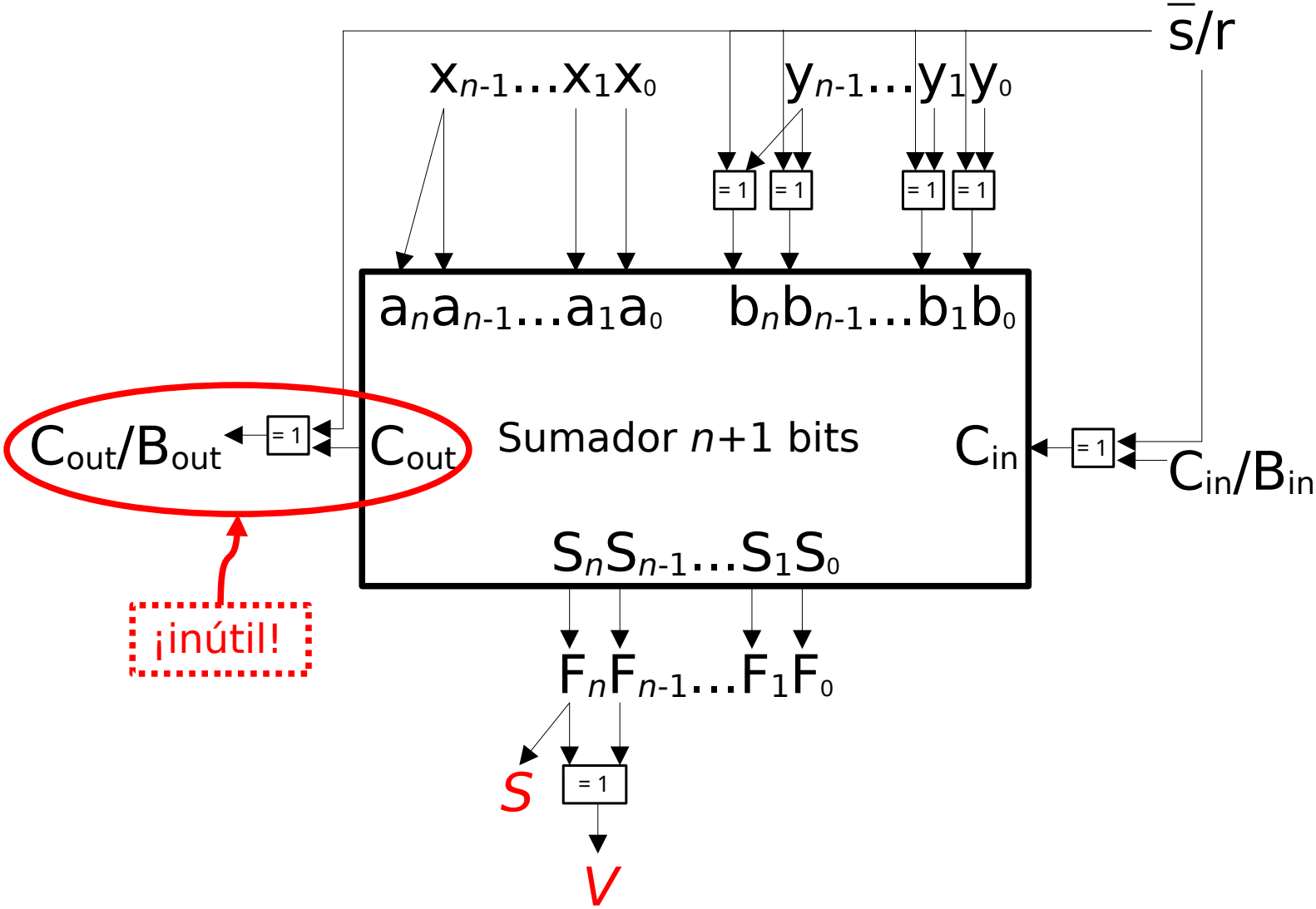
Por tanto *V* y *S* podrían generarse con este circuito:



¡inútil! necesitamos el C_{out} de la etapa n-1 para generar C

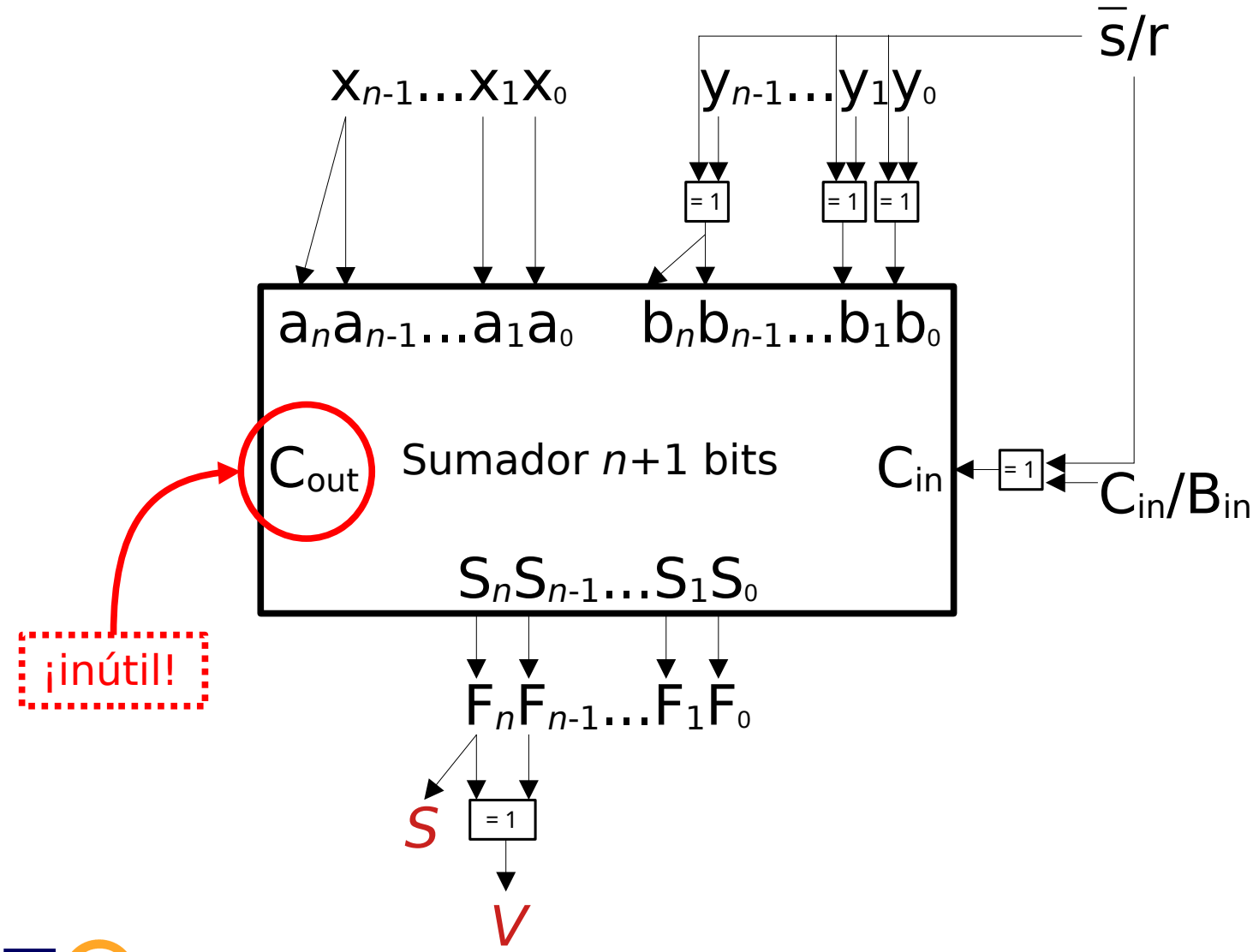
Salidas de estado

El sumador/restador de n+1 bits podría hacerse así:



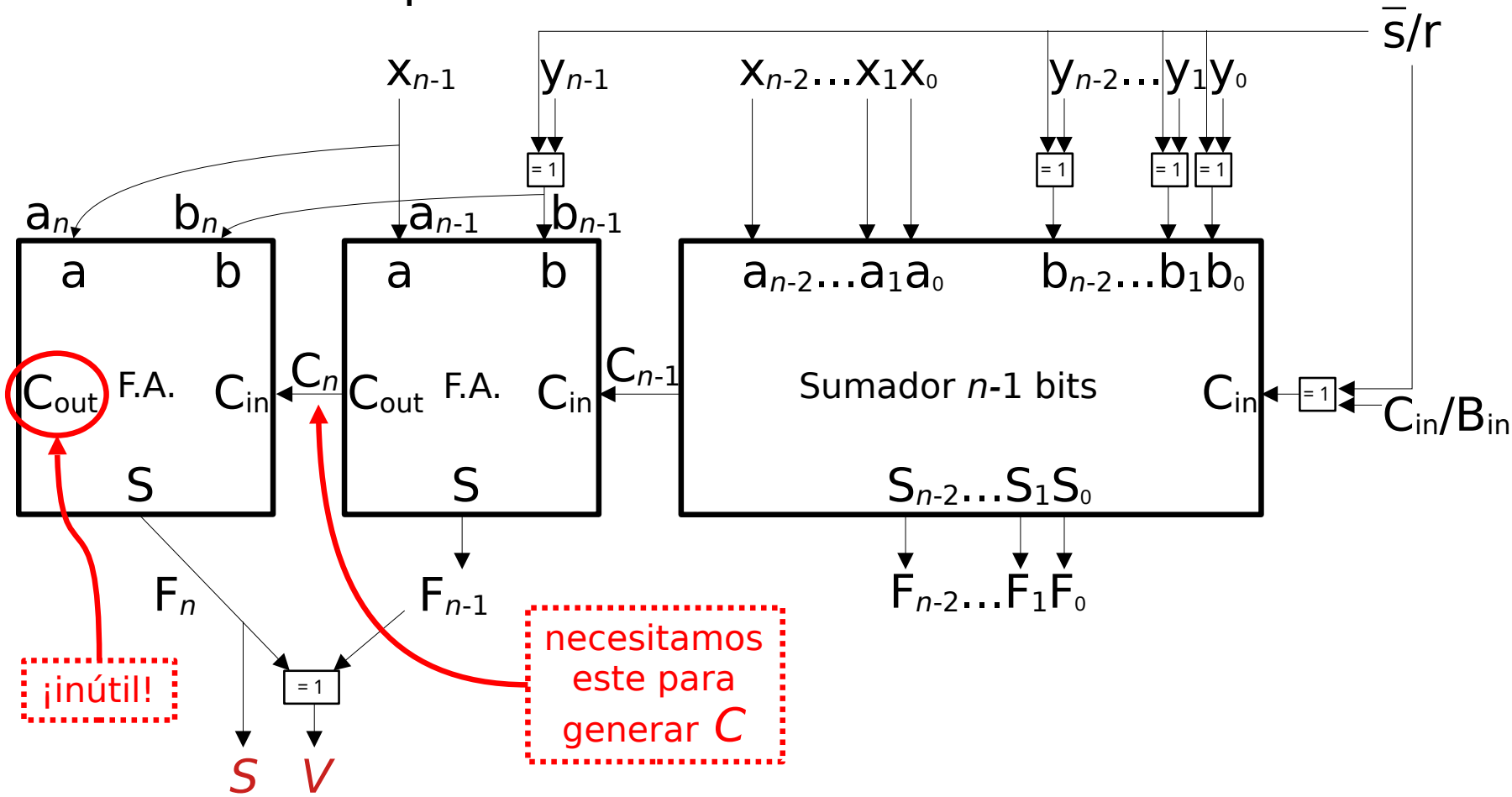
Salidas de estado

Optimizando:



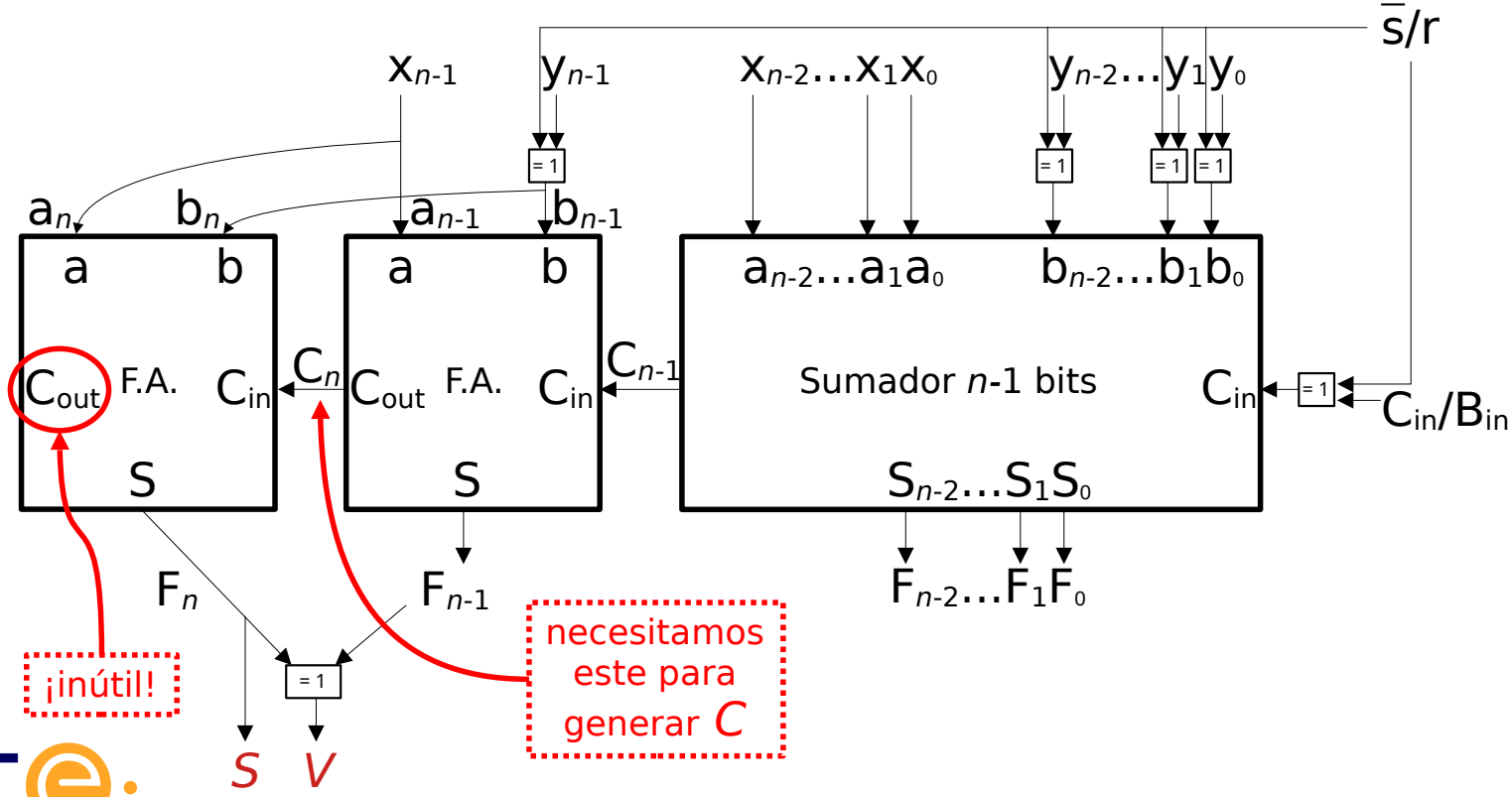
Salidas de estado

Continuemos la optimización. El sumador de n+1 bits podría hacerse con un sumador de n-1 bits en serie con dos sumadores completos de un bit:



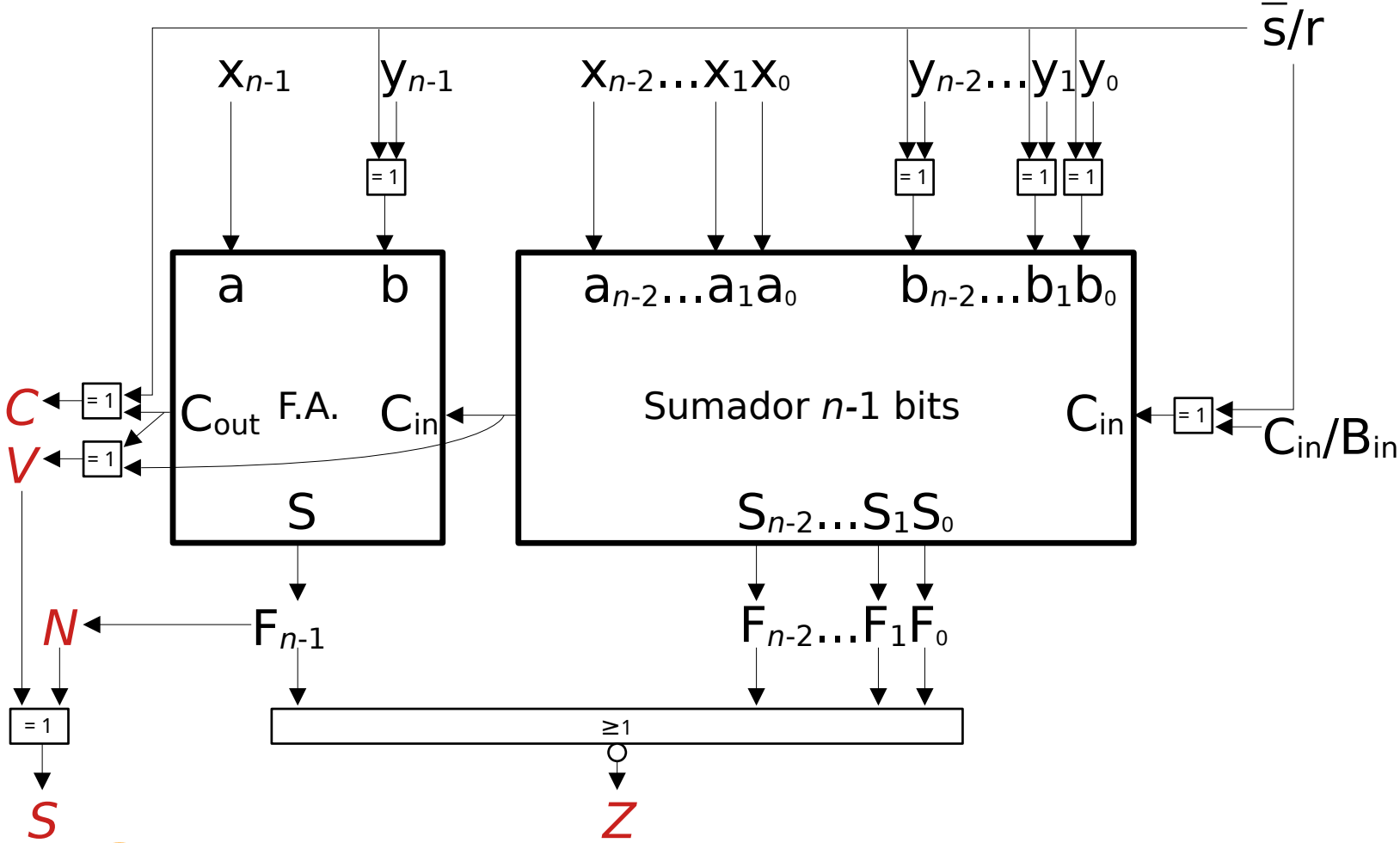
Salidas de estado

- $F_{n-1}=a_{n-1}\oplus b_{n-1}\oplus C_{n-1}$
- $F_n=a_n\oplus b_n\oplus C_n=a_{n-1}\oplus b_{n-1}\oplus C_n$
- $V=F_n\oplus F_{n-1}=C_n\oplus C_{n-1}$
- $S=F_n=F_n\oplus 0=F_n\oplus F_{n-1}\oplus F_{n-1}=V\oplus F_{n-1}=V\oplus N$



Salidas de estado

Por tanto podemos prescindir de uno de los sumadores completos. La implementación del sumador/restador de n bits con las salidas de estado mencionadas podría ser así:



• Descripción LDH de un sumador/restador de n bits

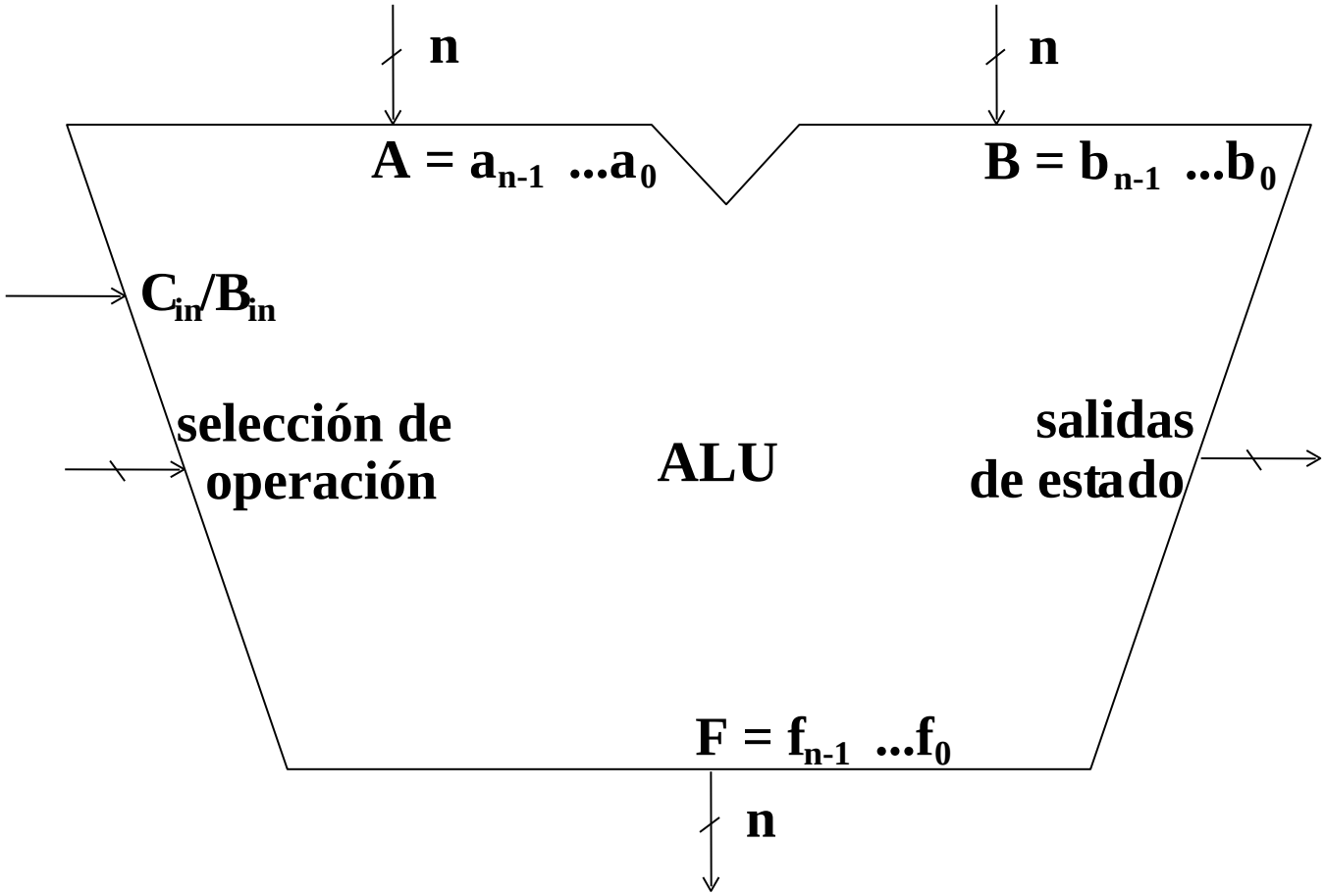
```
//////////////////////// // Sumador/Restador // ////////////////////////////////////
module sumsub1 (
        input signed [WIDTH-1:0] a, // primer operando
        input signed [WIDTH-1:0] b, // segundo operando
        input op, // operación (0-suma, 1-resta)
        output signed [WIDTH-1:0] f, // salida
        output ov // desbordamiento
        );
        parameter WIDTH = 8;
        reg f, ov;
/* f y ov se declaran como variables (tipo 'reg') porque van a usarse en un procedimiento 'always' */
        always @*
        begin :sub
/* Definimos una variable local al bloque para realizar la suma con un bit adicional
* La definición de varialbles locales es posible sólo si se nombra el bloque ('sub') en * este caso */
                reg signed [WIDTH:0] s;
/* Aquí, la construcción 'if' hubiera sido igual de efectiva que el 'case' pero, en general, cuando la decisión
* depende de una sola variable (en este caso 'op') 'case' resulta más claro, especialmente cuando el
* número de posibles valores de la variable es elevado */
                case (op)
                        0:
                                s = a + b;
                        default:
                                s = a - b;
                endcase
// Salida de desbordamiento
/* 's' contiene el valor correcto de la operación. La extensión del signo se realiza automáticamente ya que
* los tipos son ‘signed’. El desbordamiento se obtiene:  */
                if (s[WIDTH] != s[WIDTH-1])
                        ov = 1;
                else
                        ov = 0;
// Salida
                f = s[WIDTH-1:0];
        end
endmodule // sumsub1
```

Unidad aritmético-lógica (ALU)

- Es un circuito que realiza operaciones aritméticas y lógicas. Normalmente usa dos operandos de entrada.
- Tiene señales de control de entrada que permiten seleccionar la operación a realizar.

Unidad aritmético-lógica (ALU)

- Representación esquemática de una ALU



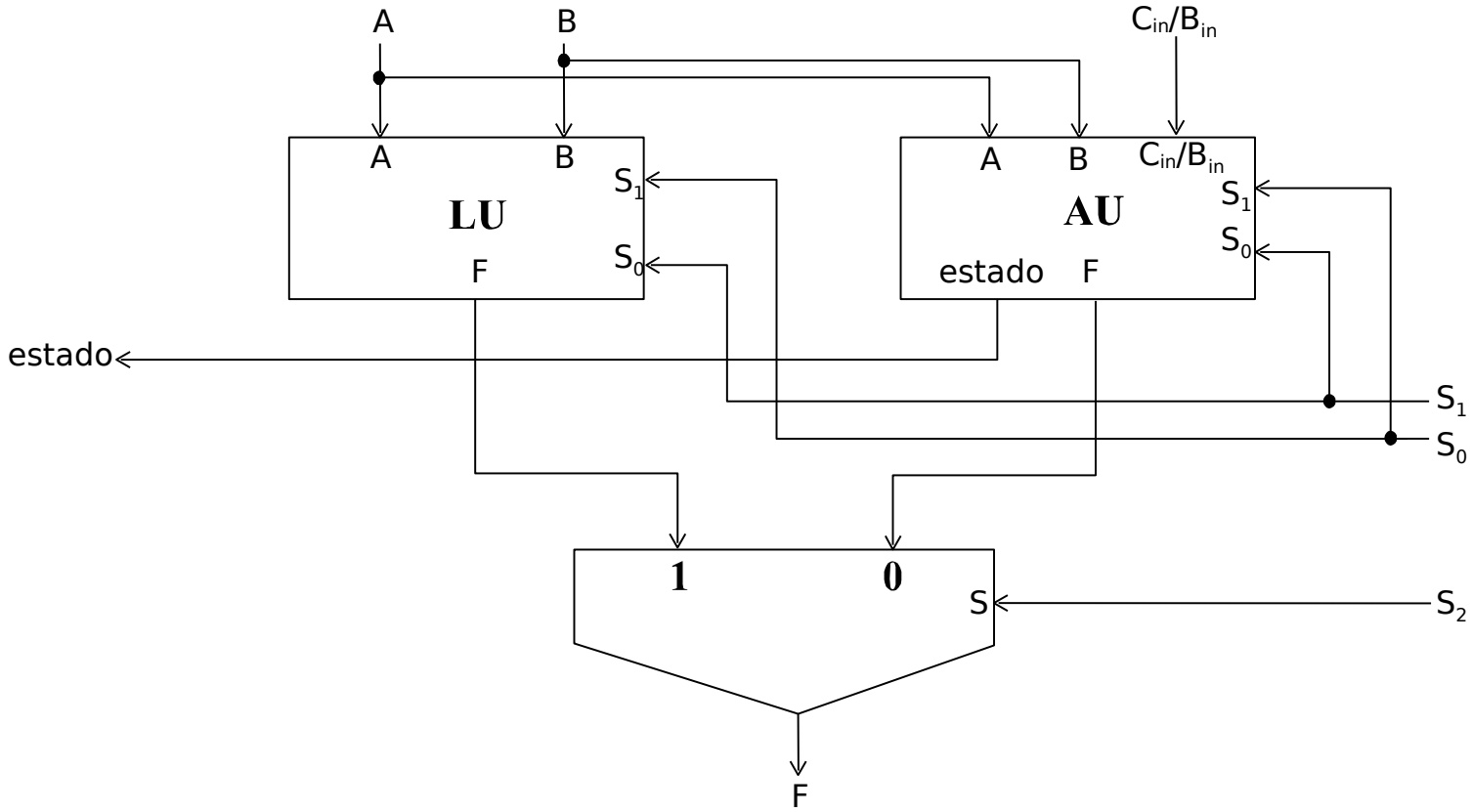
Unidad aritmético-lógica (ALU)

- Ejemplo de funciones de una posible ALU con tres señales de selección de operación:

S_2 S_1 S_0	Función
0 0 0	$F_{(2} = A_{(2} + B_{(2} \bmod 2^n$
0 0 1	$F_{(2} = A_{(2} - B_{(2} \bmod 2^n$
0 1 0	$F_{(2} = A_{(2} + B_{(2} + Cin/Bin_{(2} \bmod 2^n$
0 1 1	$F_{(2} = A_{(2} - B_{(2} - Cin/Bin_{(2} \bmod 2^n$
1 0 0	F = A AND B
1 0 1	F = A OR B
1 1 0	F = NOT A
1 1 1	F = A XOR B

Unidad aritmético-lógica (ALU)

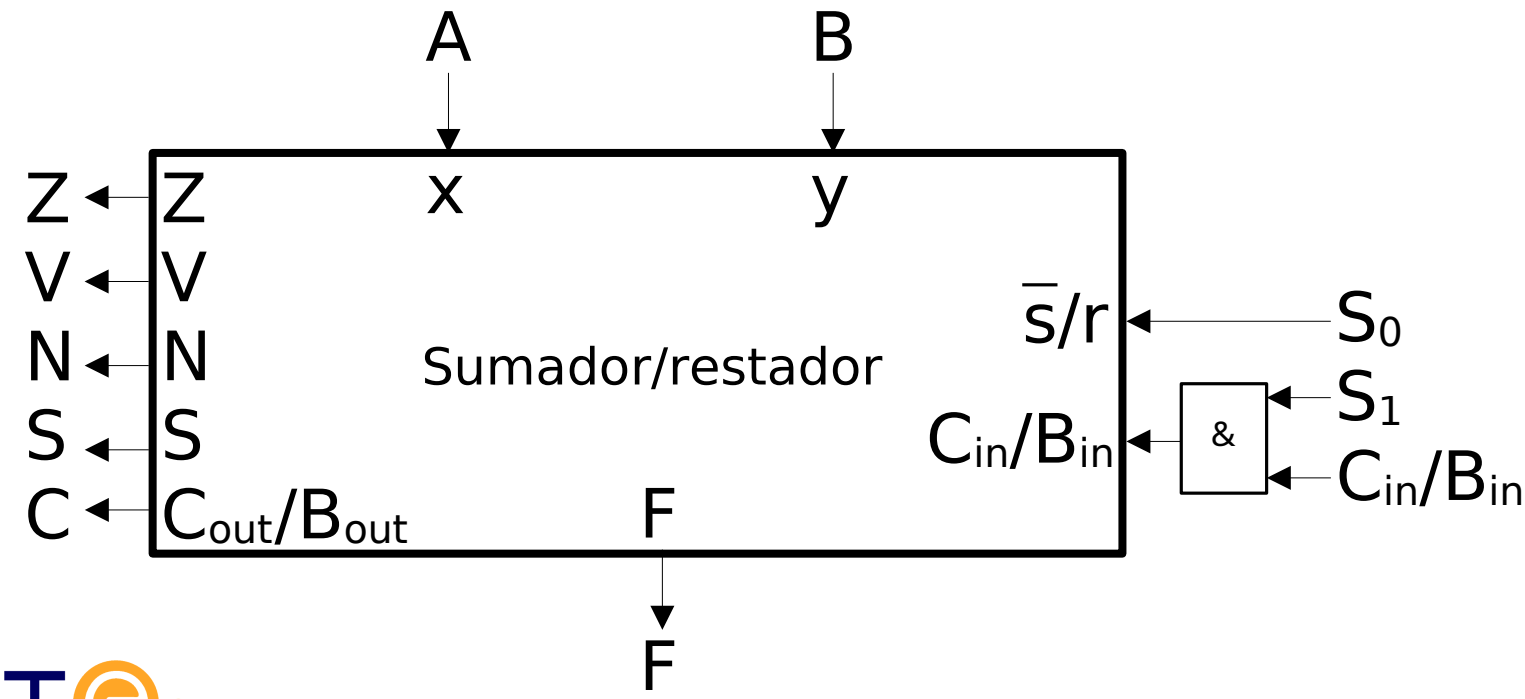
- Ejemplo de implementación de la ALU del ejemplo:
  Se divide el diseño en una unidad aritmética (AU) y una unidad lógica (LU).



Unidad aritmético-lógica (ALU)

- Ejemplo de implementación de la unidad aritmética:

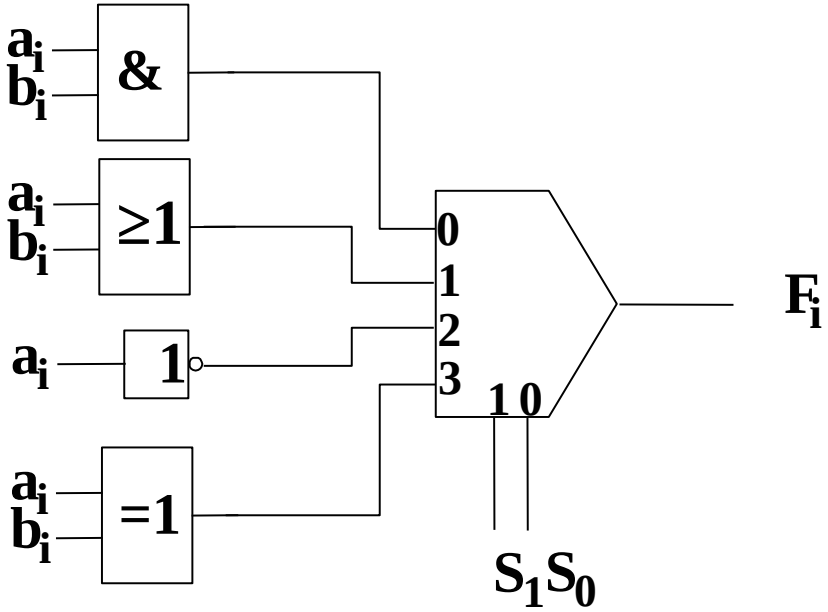
S_2 S_1 S_0	Función ALU
0 0 0	$F_{(2} = A_{(2} + B_{(2} \bmod 2^n$
0 0 1	$F_{(2} = A_{(2} - B_{(2} \bmod 2^n$
0 1 0	$F_{(2} = A_{(2} + B_{(2} + Cin/Bin_{(2} \bmod 2^n$
0 1 1	$F_{(2} = A_{(2} - B_{(2} - Cin/Bin_{(2} \bmod 2^n$



Unidad aritmético-lógica (ALU)

- Ejemplo de implementación de la unidad lógica:

S_2 S_1 S_0	Función ALU
1 0 0	F = A AND B
1 0 1	F = A OR B
1 1 0	F = NOT A
1 1 1	F = A XOR B



Apéndices

Sumadores con signo: Sumador en Ca2

- Sea un sumador de magnitud de n bits con entradas a, b, Cin y salidas S, $Cout$, si $a_{(Ca2}+b_{(Ca2}+Cin_{(2}$ es representable en complemento a 2 con n bits entonces $S_{(Ca2}=a_{(Ca2}+b_{(Ca2}+Cin_{(2}$.

- Demostración: La hipótesis dice que hay una cadena S' de n bits tal que $S'_{(Ca2}=a_{(Ca2}+b_{(Ca2}+Cin_{(2}$. Hay que demostrar que $S'=S$. Por definición tenemos

$$S'_{(2}=S'_{(Ca2} \bmod 2^n=(a_{(Ca2}+b_{(Ca2}+Cin_{(2}) \bmod 2^n=$$

$$=[(a_{(Ca2} \bmod 2^n)+(b_{(Ca2} \bmod 2^n)+Cin_{(2}] \bmod 2^n=(a_{(2}+b_{(2}+Cin_{(2}) \bmod 2^n=S_{(2}$$

$S'_{(2}=S_{(2}$ implica que $S'=S$, pues tienen la misma longitud.

- Conclusión: un sumador en Ca2 es funcionalmente idéntico a un sumador de magnitud.

Restador en Ca2

- Sea un restador de magnitud de n bits con entradas a (minuendo), b (sustranedo), Bin y salidas R, $Bout$, si $a_{(Ca2}$-$b_{(Ca2}$-$Bin_{(2}$ es representable en complemento a 2 con n bits entonces $R_{(Ca2}=a_{(Ca2}$-$b_{(Ca2}$-$Bin_{(2}$.

- Demostración: La hipótesis dice que hay una cadena R' de n bits tal que $R'_{(Ca2}=a_{(Ca2}$-$b_{(Ca2}$-$Bin_{(2}$. Hay que demostrar que $R'=R$. Por definición tenemos

$$R'_{(2}=R'_{(Ca2} \bmod 2^n=(a_{(Ca2}-b_{(Ca2}-Bin_{(2}) \bmod 2^n=$$

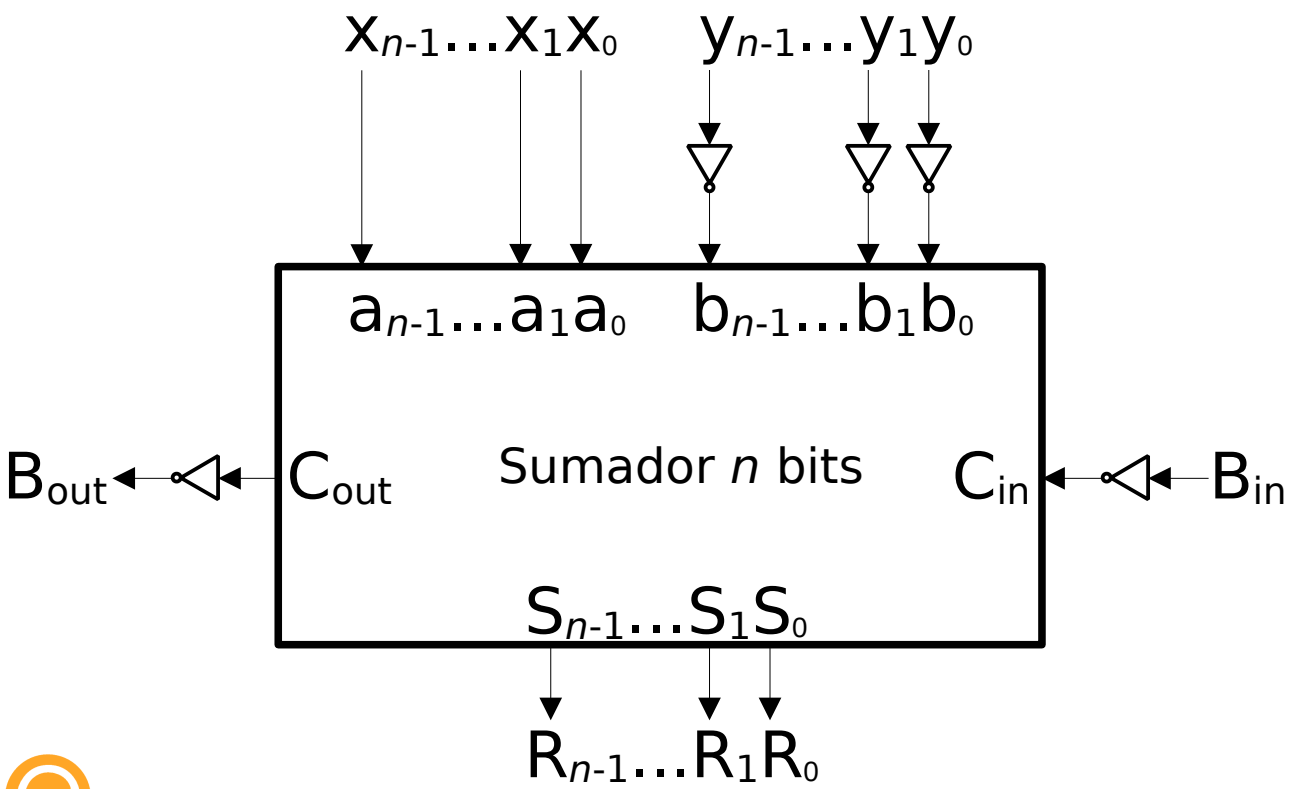
$$=[(a_{(Ca2} \bmod 2^n)-(b_{(Ca2} \bmod 2^n)-Bin_{(2}] \bmod 2^n=(a_{(2}-b_{(2}-Bin_{(2}) \bmod 2^n=R_{(2}$$

$R'_{(2}=R_{(2}$ implica que $R'=R$, pues tienen la misma longitud.

- Conclusión: un restador en Ca2 es funcionalmente idéntico a un restador de magnitud.

Implementación de un restador de *n* bits

- Un restador de magnitud/Ca2 de *n* bits con señales de entrada *x*, *y*, *Bin* (minuendo, sustraendo, entrada de prestamo) y señales de salida *R*, *Bout* (resta, salida de prestamo) puede implementarse así:



Implementación de un restador de *n* bits

- Antes de demostrar que funciona, véase el pequeño lema $L \bmod 2^{n+1}=0$, siendo $L=(1+\overline{Cout}_{(2}-Cout_{(2})\ 2^n$
- Sabemos que
  - $\{Cout,S\}_{(2}=a_{(2}+b_{(2}+Cin_{(2}=x_{(2}+\overline{y}_{(2}+\overline{Bin}_{(2}$
  - $Bout=\overline{Cout}$
  - $R=S$
- Hay que demostrar
  - $\{Bout,R\}_{(2}=(x_{(2}-y_{(2}-Bin_{(2})\bmod 2^{n+1}$

Implementación de un restador de *n* bits

- Demostración:

$\{Bout,R\}_{(2}=\{Bout,R\}_{(2}\text{mod } 2^{n+1}=\{\overline{Cout},S\}_{(2}\text{mod } 2^{n+1}=$
$(\overline{Cout}_{(2}2^n+S_{(2})\text{mod } 2^{n+1}=$
$(2^n+\overline{Cout}_{(2}2^n-Cout_{(2}2^n+Cout_{(2}2^n+S_{(2}-2^n)\text{mod } 2^{n+1}=$
$[(1+\overline{Cout}_{(2}-Cout_{(2})2^n+\{Cout,S\}_{(2}-2^n]\text{mod } 2^{n+1}=$
$(L+\{Cout,S\}_{(2}-2^n)\text{mod } 2^{n+1}=$
$[(L \text{ mod } 2^{n+1}+(\{Cout,S\}_{(2}-2^n)\text{mod } 2^{n+1}]\text{mod } 2^{n+1}=$
$[(\{Cout,S\}_{(2}-2^n)\text{mod } 2^{n+1}]\text{mod } 2^{n+1}=$
$(\{Cout,S\}_{(2}-2^n)\text{mod } 2^{n+1}=(x_{(2}+\overline{y}_{(2}+\overline{Bin}_{(2}-2^n)\text{mod } 2^{n+1}=$
$(x_{(2}+2^n-1-y_{(2}+2^1-1-Bin_{(2}-2^n)\text{mod } 2^{n+1}=$
$(x_{(2}-y_{(2}-Bin_{(2})\text{mod } 2^{n+1}$