
Circuitos electrónicos digitales

Universidad de Sevilla

Tema 6

Unidades aritméticas y lógicas

Índice

- **Introducción**
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

Introducción

- Los sistemas digitales poseen una **gran potencia de cálculo** ya que permiten ejecutar con gran velocidad **operaciones aritméticas y lógicas**
- Una operación aritmética en un computador puede ser realizada de **dos formas**:
 - *hardware*: existe un circuito en el procesador que realiza esa operación (gran velocidad y alto coste)
 - *software*: existe un algoritmo que descompone esa operación en otras más elementales que son realizadas mediante hardware
- Aritmética binaria no entera
 - Coma fija
 - Coma flotante

Introducción

- **Hardware aritmético** en los procesadores:
 - Todos los procesadores poseen al menos un sumador-restador
 - Algunos poseen circuitos para otras operaciones enteras como multiplicación o división.
- **Instrucciones aritméticas** en los procesadores:
 - Los procesadores más simples poseen instrucciones para aritmética entera.
 - Otros poseen instrucciones para aritmética no entera, pudiendo incluso incluir operaciones tan complejas como exponenciales, logaritmos u operadores trigonométricos.

Introducción

- Las principales diferencias entre la forma de operar manual y la de un computador digital son:
 - La base del sistema de numeración es $B = 2$ (binaria).
 - La forma de representar números con signo normalmente no es con signo-magnitud, sino a través de los complementos (a 2 o a 1).
 - El número de bits de los datos está acotado, lo que introduce errores de desbordamiento, de precisión y de cumplimiento de propiedades algebraicas (algunas operaciones son no cerradas y pueden incumplirse las propiedades asociativas y distributiva).

Índice

- Introducción
- **Aritmética binaria**
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - Suma aritmética

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

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - Suma aritmética

0 1 1 0 0	Acarreo
1 0 0 1 1 0	Sumando 1
+ 1 1 0 1	Sumando 2
<hr/>	
1 1 0 0 1 1	Suma

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - Resta

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

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - Resta

1 0 0 1 1 0	Minuendo
- 1 1 0 1	Sustraendo
1 1 0 0 1	Acarreo
<hr/>	
0 1 1 0 0 1	Resta

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - Multiplicación

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

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - **División**

1 0 0 1 1 0

1 1 0 1

Aritmética binaria

- ¿Cómo se realizan las cuatro operaciones básicas?
 - **División**

Dividendo

$$\begin{array}{r} 100110 \\ - 1101 \\ \hline 01100 \\ - 0000 \\ \hline 1100 \end{array}$$

$$\begin{array}{r} 1101 \text{ Divisor} \\ \hline 10 \text{ Cociente} \end{array}$$

Resto

Aritmética binaria

- Representación de números con signo
 - Signo-Magnitud
 - Complemento a 1
 - Complemento a 2
 - Exceso

Aritmética binaria

- Representación de números con signo
 - Signo-Magnitud

		[] [] [] [] [] [] [] []						
	Signo	Magnitud						
+ 90 ₍₁₀₎	→ 0	1	0	1	1	0	1	0
- 90 ₍₁₀₎	→ 1	1	0	1	1	0	1	0

- Signo: Positivo (0) ; Negativo (1)
- Representable con n bits: $[-(2^{n-1} - 1), 2^{n-1} - 1]$
- Doble representación del cero (4 bits): 0000 y 1000

Aritmética binaria

- Representación de números con signo
 - **Complemento a 1 (Ca1)**
 - Números positivos
 - Igual que en signo-magnitud
 - Ejemplo: $+90_{(10)} \rightarrow 01011010$
 - Números negativos
 - Se obtiene el Ca1 de su representación como número positivo.
 - Ejemplo: $-90_{(10)} \rightarrow 10100101$
 - Obtención del Ca1: se complementan todos los bits
 - Signo: Positivo (0) ; Negativo (1)
 - Representable con n bits: $[-(2^{n-1} - 1), 2^{n-1} - 1]$
 - Doble representación del cero (4 bits): 0000 y 1111

Aritmética binaria

- Representación de números con signo
 - **Complemento a 2 (Ca2)**
 - Números positivos
 - Igual que en signo-magnitud
 - Ejemplo: $+90_{(10)} \rightarrow 01011010$
 - Números negativos
 - Se obtiene el Ca2 de su representación como número positivo.
 - Ejemplo: $-90_{(10)} \rightarrow 10100110$
 - Obtención del Ca2: comenzando por el LSB, se conservan los bits a cero y el primer uno, complementando el resto.
 - Signo: Positivo (0) ; Negativo (1)
 - Representable con n bits: $[-2^{n-1}, 2^{n-1} - 1]$
 - Representación única del cero (4 bits): 0000

Aritmética binaria

- Representación de números con signo
 - Exceso K (donde K es un número natural)
 - Con n bits permite representar cualquier entero en el rango $[-K, 2^n-1-K]$.
 - En esta notación, a cualquier entero X se le asigna el mismo código binario que le corresponde al entero $X+K$ en notación sin signo.
- Representable con n bits: (2^n) números
- Representación única del cero (4 bits con $K=8$): 1000

Aritmética binaria

- Representación de números con signo, ejemplo 4 bits

	S-M	Ca1	Ca2	Exceso 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

Aritmética binaria

- Aritmética 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.
 - **En el diseño lógico del circuito se necesitan sumadores, restadores, comparadores, etc.**

Aritmética binaria

- Aritmética en Ca1 y Ca2
 - La notación en Ca1 y Ca2 elimina la necesidad de utilizar circuitos restadores y comparadores ya que:

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

- En Ca1 si $C_{OUT} = 1$ se añade al resultado

1	1 0 0 0	acarreo	0 0 1 1 0	acarreo
	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)

- En Ca2 si $C_{OUT} = 1$ se desprecia

1 1 0 1 0	acarreo
1 1 0 1 1	-5
1 1 0 1 0	-6
1 0 1 0 1	-11 (OK)

- Esto hace que Ca2 sea la más utilizada

Índice

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

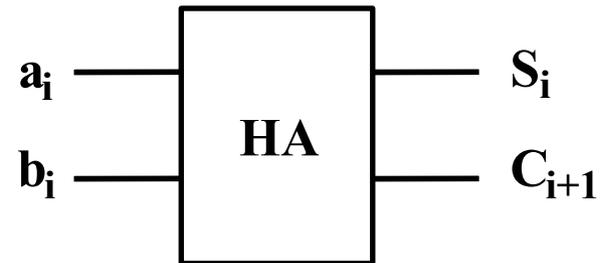
Índice

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

Circuitos sumadores básicos

- Semisumador o Half Adder (HA)
 - Se trata del circuito que suma dos bits.
 - Obtiene como salida el bit de suma y el acarreo.

a_i	b_i	C_{i+1}	S_i
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

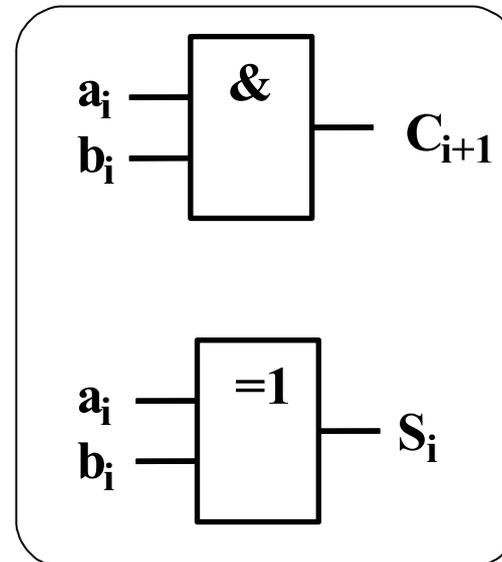


Circuitos sumadores básicos

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

$$C_{i+1} = a_i \cdot b_i$$

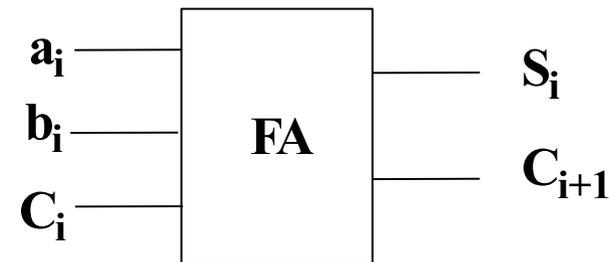
$$S_i = a_i \oplus b_i$$



Circuitos sumadores básicos

- Sumador completo Full Adder (FA)
 - Permite realizar la suma de tres bits simultáneamente.
 - Obtiene como salida el bit de suma y el acarreo.

a_i	b_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

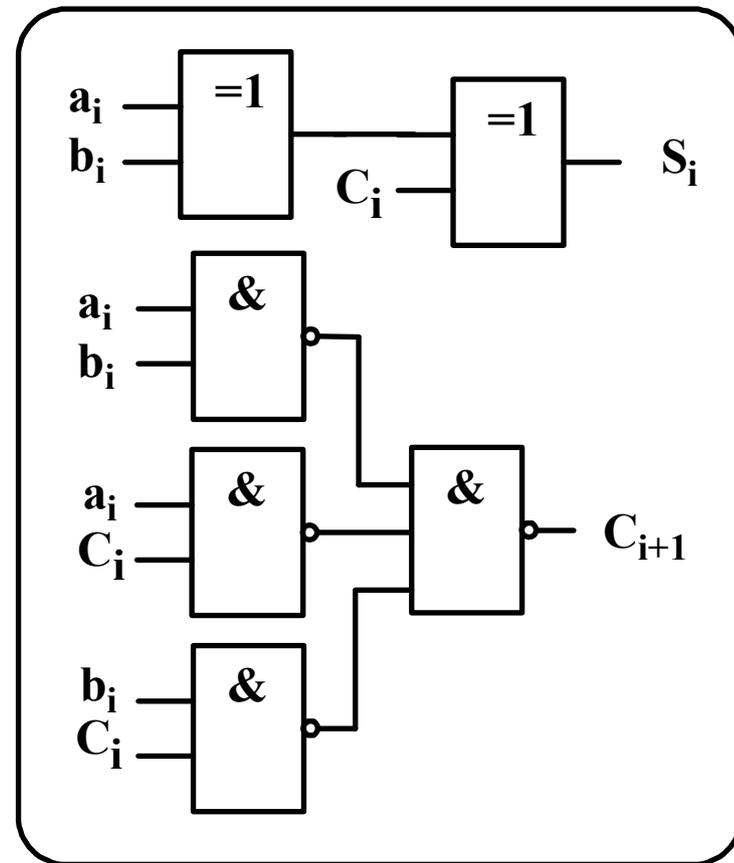


Circuitos sumadores básicos

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

$$C_{i+1} = a_i \cdot b_i + a_i \cdot C_i + b_i \cdot C_i$$

$$S_i = a_i \oplus b_i \oplus C_i$$

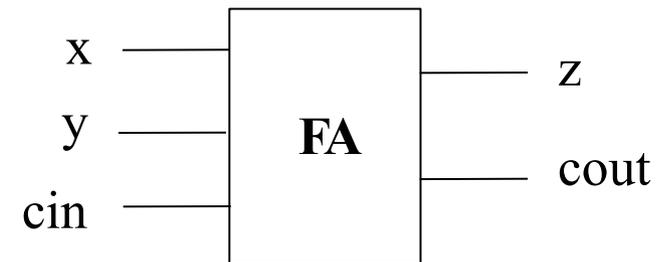


Circuitos sumadores básicos

- Descripción Verilog 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
```

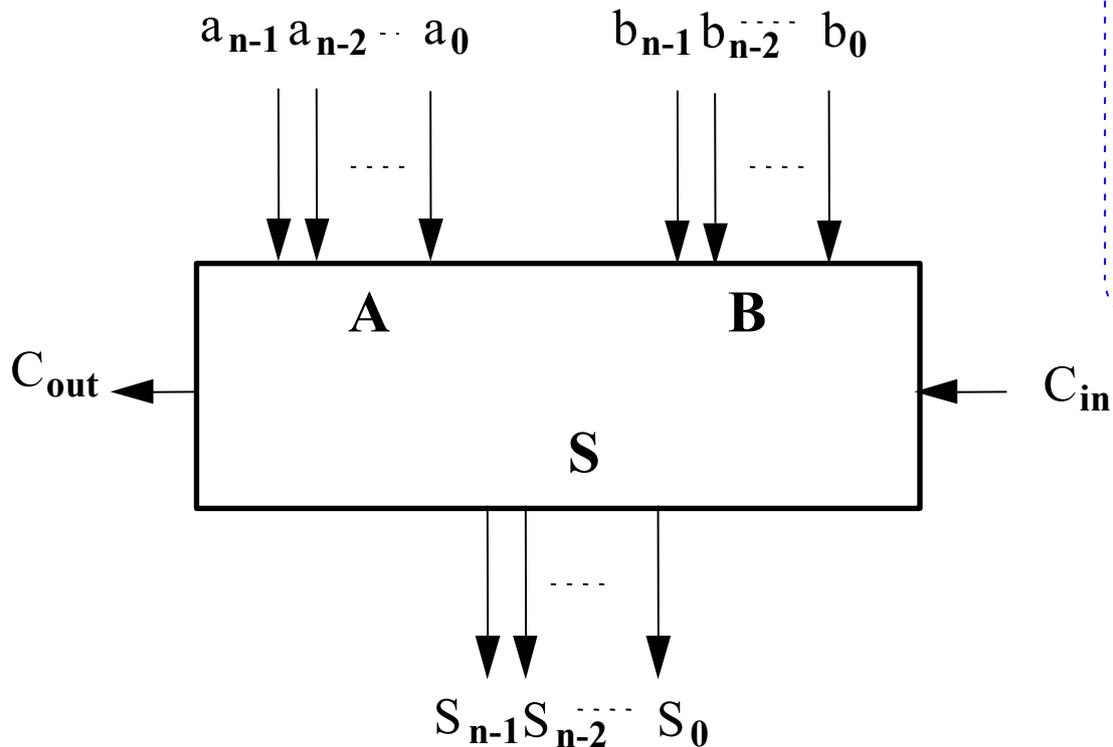


Índice

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

Sumador de n bits

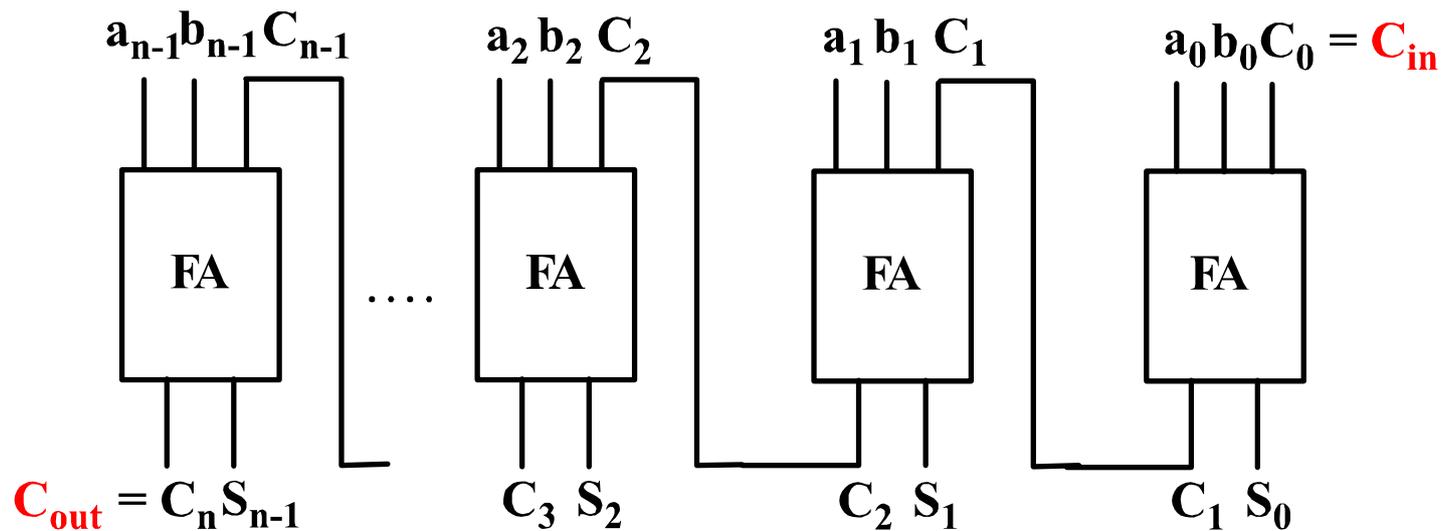
- Un sumador de **n bits**, es un dispositivo lógico combinacional de **2n+1 entradas** y **n+1 salidas** que realiza la suma de dos numerales binarios de **n bits**.



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

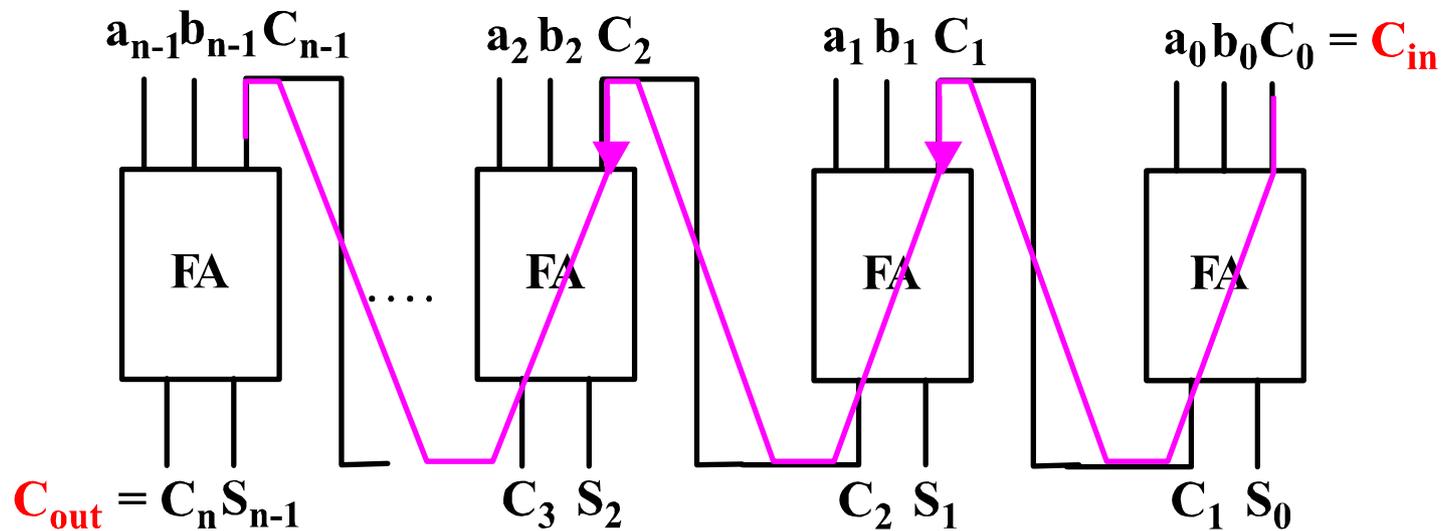
Sumador de n bits

- Sumador con acarreo serie
 - Es el más intuitivo y tiene un coste razonablemente bajo.
 - También es conocido como **sumador de rizado** o **ripple adder**
 - Se trata de un circuito modular



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 Verilog de un sumador de 8 bits

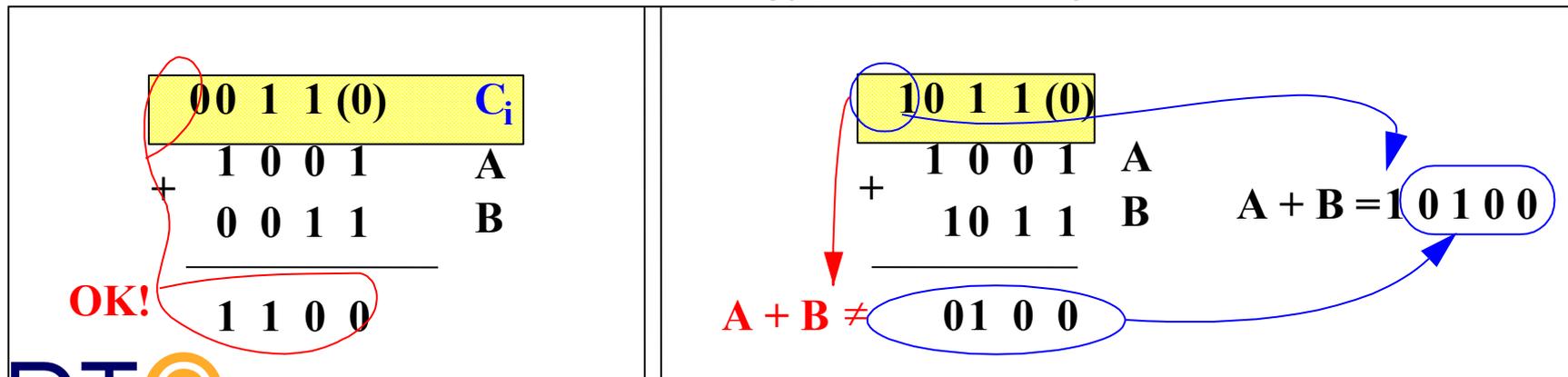
```
//////////////////////////////// // Sumador 8 bits con FA // //////////////////////////////////
module adder8_e(
    input [7:0] a, // primer operando
    input [7:0] b, // segundo operando
    input cin, // acarreo de entrada
    output [7:0] s, // salida de suma
    output cout // acarreo de salida
);
/* 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
```

Sumador de n bits

- El problema del desbordamiento en la suma de magnitudes
 - Con n bits el **rango** representable sin signo es $[0, 2^n - 1]$
 - Si $A + B > 2^n - 1$
 - el resultado no es representable
 - hay desbordamiento (**overflow**)
 - C_{out} señala la existencia de desbordamiento
 - El resultado correcto será el representado por el numeral de **$n+1$ bits** $C_{out} S_{n-1} \dots S_0$



Índice

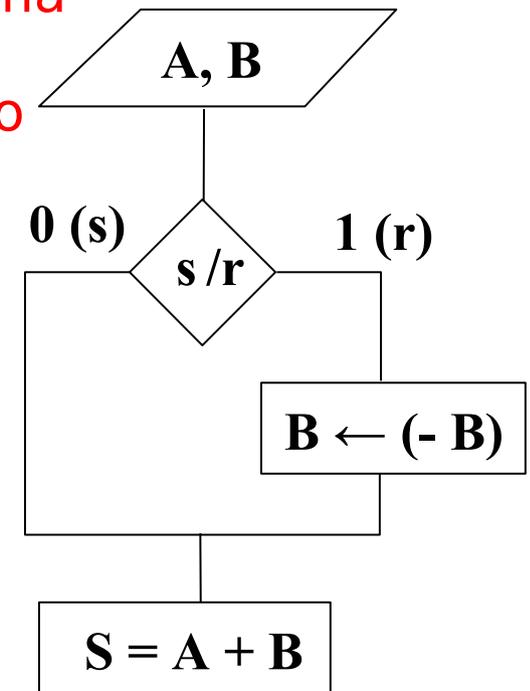
- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- Unidad aritmético-lógica (ALU)

Sumador/Restador

- La suma-resta de números con signo
 - Calcular la diferencia $A - B$ es equivalente a calcular $A + (-B)$
 - la resta aritmética **se reduce a una suma**
 - implica trabajar con **números con signo**

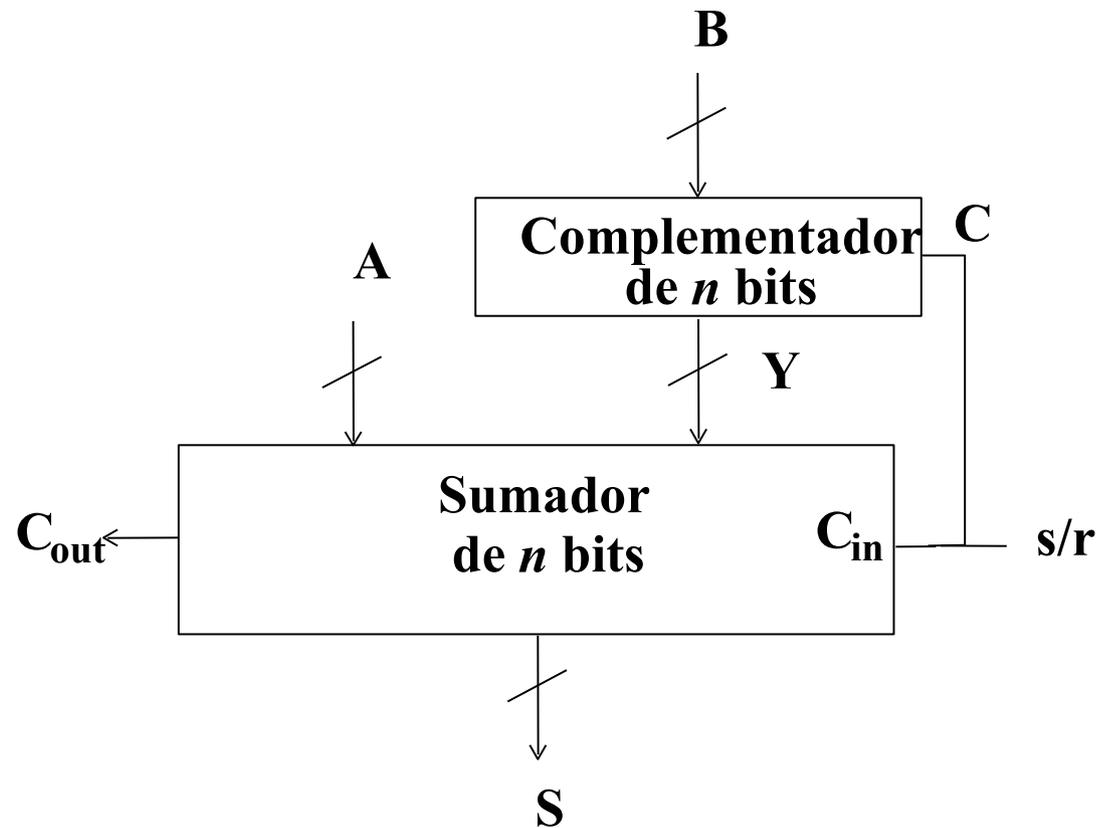
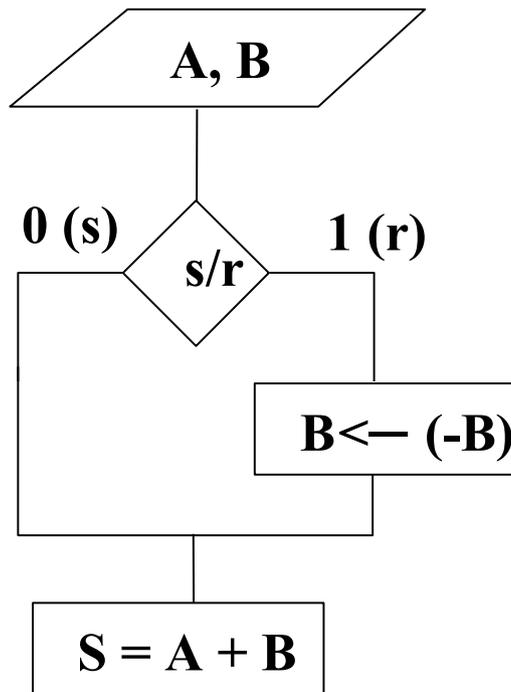
– \bar{B} es $-B_{(CA1)}$

– $\bar{B}+1$ es $-B_{(CA2)}$



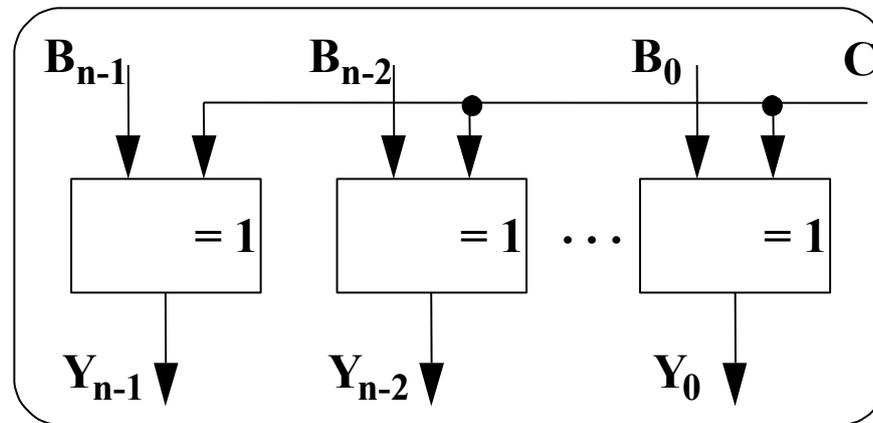
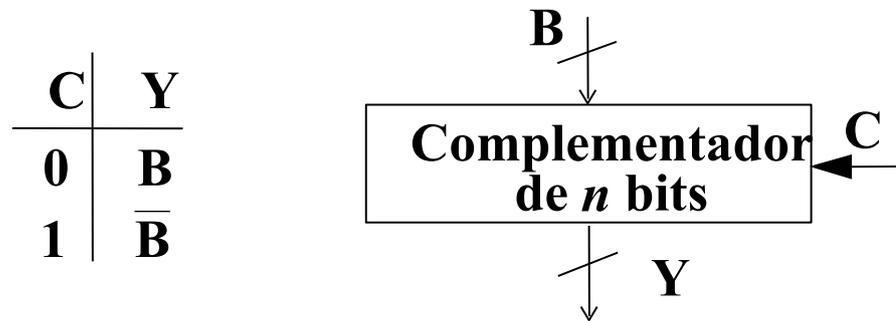
Sumador/Restador

- La suma-resta de números en complemento a 2
 - En general: $A - B = A + (-B)$



Sumador/Restador

- El complementador es simplemente una colección de puertas XOR



Sumador/Restador

- La suma-resta de números en complemento a 2
- Utilizaremos la notación complemento a 2 para representar los números positivos y negativos

$$\begin{array}{r} 1001 = -7 \\ 0101 = +5 \\ \hline 1110 = -2 \end{array}$$

$$\begin{array}{r} 1100 = -4 \\ 1111 = -1 \\ \hline 11011 = -5 \end{array}$$

$$\begin{array}{r} 1100 = -4 \\ 0100 = +4 \\ \hline 10000 = 0 \end{array}$$

$$\begin{array}{r} 0101 = +5 \\ 0100 = +4 \\ \hline 1001 = -7 \end{array}$$

$$\begin{array}{r} 0011 = +3 \\ 0100 = +4 \\ \hline 0111 = +7 \end{array}$$

$$\begin{array}{r} 1001 = -7 \\ 1010 = -6 \\ \hline 10011 = +3 \end{array}$$

desbordamiento

Sumador/Restador

- El problema del desbordamiento en la suma-resta de números con signo
 - Se pone de manifiesto porque **el bit de signo no es correcto**
 - En caso de desbordamiento, el resultado correcto está representado por el numeral de $n+1$ bits $\mathbf{C}_{out}\mathbf{S}_{n-1} \dots \mathbf{S}_0$
 - La detección del desbordamiento se lleva a cabo mediante una señal adicional: **el bit de overflow (V)**

Sumador/Restador

- El problema del desbordamiento en la suma-resta de números con signo
 - En la suma, el desbordamiento se produce cuando:
 - al sumar dos números positivos se obtiene uno negativo
 - al sumar dos números negativos se obtiene uno positivo

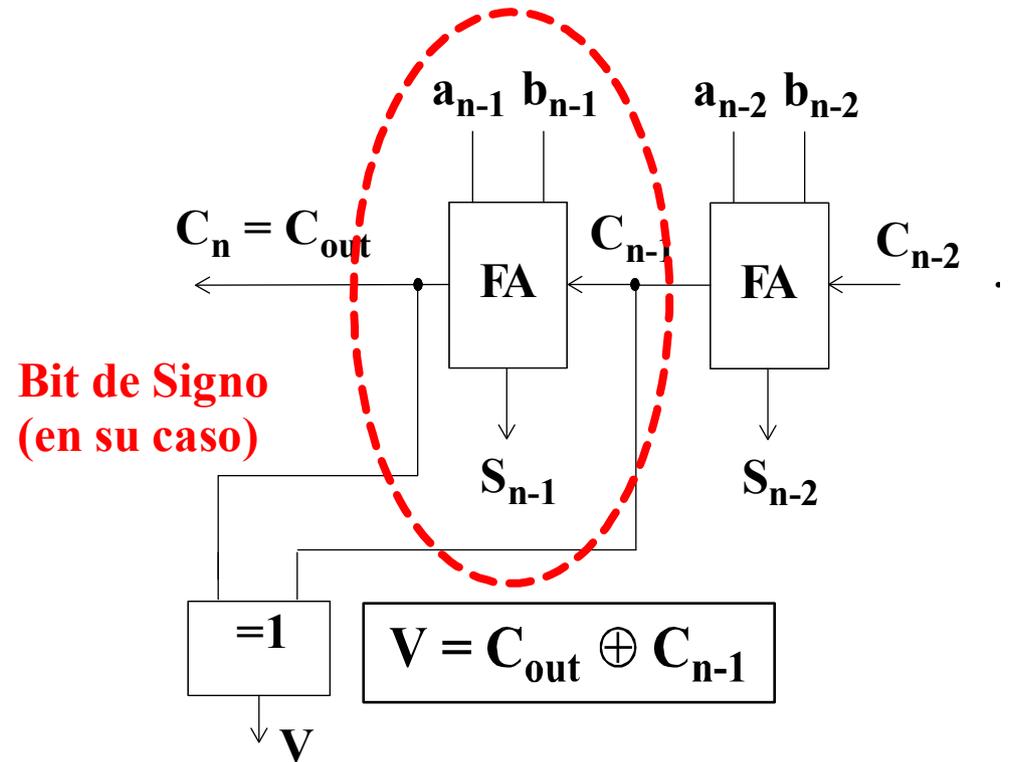
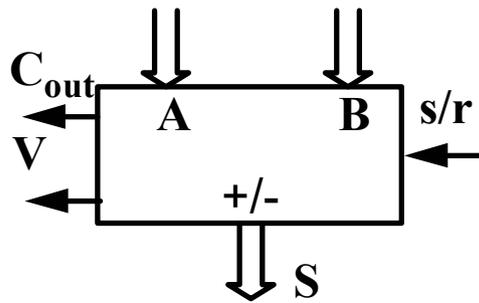
$$\begin{array}{r} C_n \quad C_{n-1} \\ 0 \quad 1 \\ + \quad 0 \quad a_{n-2} \quad \dots \quad a_0 \\ \quad 0 \quad b_{n-2} \quad \dots \quad b_0 \\ \hline 1 \quad S_{n-2} \quad \dots \quad S_0 \end{array}$$

$$\begin{array}{r} C_n \quad C_{n-1} \\ 1 \quad 0 \\ + \quad 1 \quad a_{n-2} \quad \dots \quad a_0 \\ \quad 1 \quad b_{n-2} \quad \dots \quad b_0 \\ \hline 0 \quad S_{n-2} \quad \dots \quad S_0 \end{array}$$

$$V = C_{out} \oplus C_{n-1}$$

Sumador/Restador

- El sumador restador quedaría:



● Descripción Verilog 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 variables 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
```

Índice

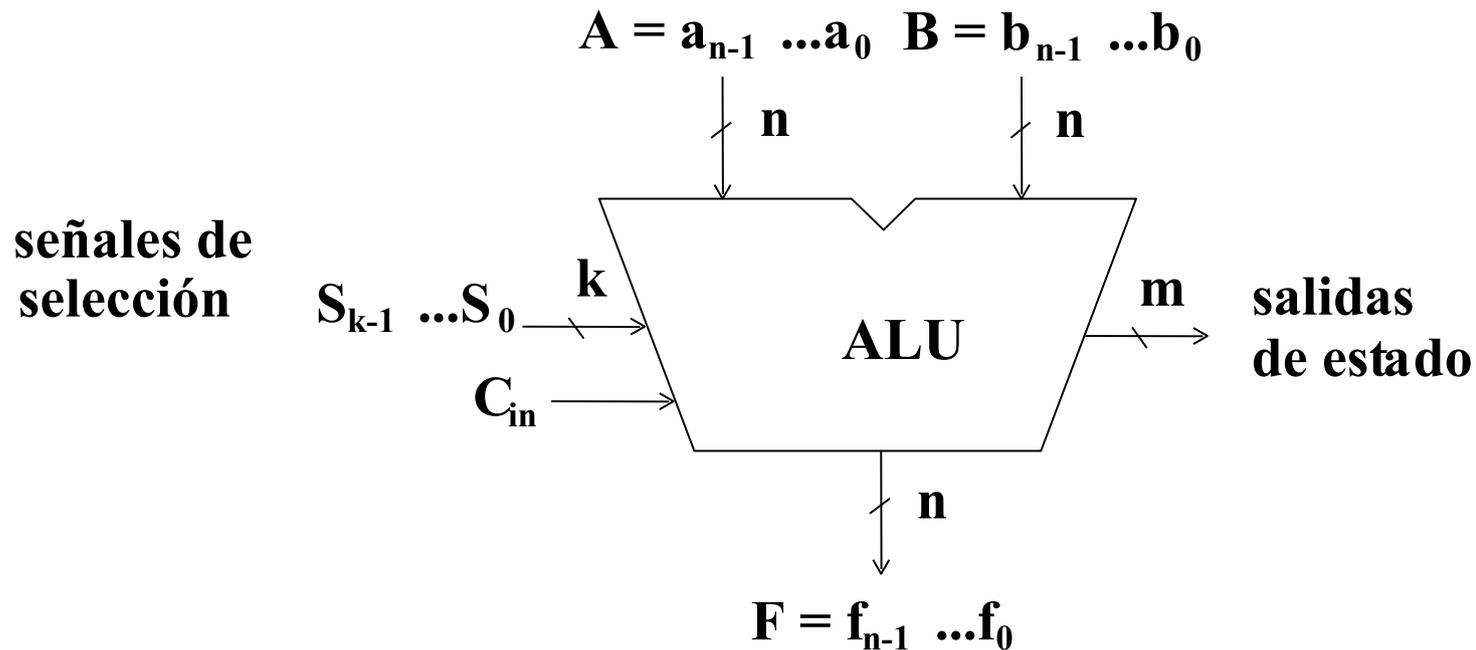
- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de n bits
- Sumador/Restador
- **Unidad aritmético-lógica (ALU)**

Unidad aritmético-lógica (ALU)

- Es el circuito donde se realiza el procesamiento de datos
 - Procesado: operaciones aritméticas y lógicas. Normalmente se opera sobre dos datos
 - Usualmente pueden realizar diversas operaciones. Para elegirlos se incluyen unas señales de selección
 - Además de las salidas que muestran el resultado de la operación, se incluyen otras salidas (*flags*) de estado o de condición.
 - Típicamente son C_{out} , V, Z (Z=1 si el resultado es 0) y S (signo)

Unidad aritmético-lógica (ALU)

- Representación gráfica de una ALU



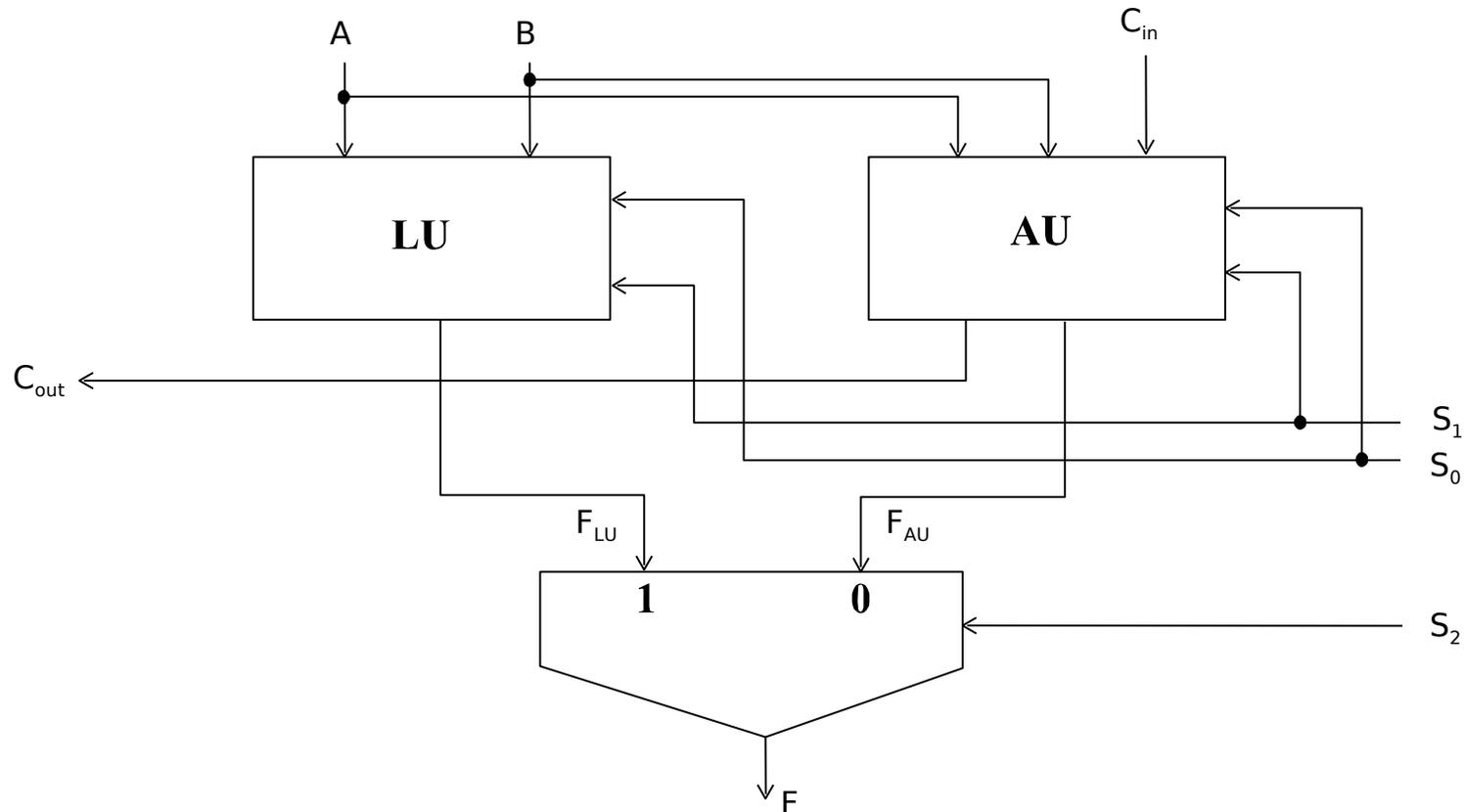
Unidad aritmético-lógica (ALU)

- Ejemplo de una ALU

S_2 S_1 S_0	Función ALU	
	$C_{in} = 0$	$C_{in} = 1$
0 0 0	$F = A$	$F = A + 1$
0 0 1	$F = A + B$	$F = A + B + 1$
0 1 0	$F = A + \bar{B}$	$F = A + \bar{B} + 1$
0 1 1	$F = A - 1$	$F = A$
1 0 0	$F = A \text{ AND } B$	
1 0 1	$F = A \text{ OR } B$	
1 1 0	$F = \text{NOT } A$	
1 1 1	$F = A \text{ XOR } B$	

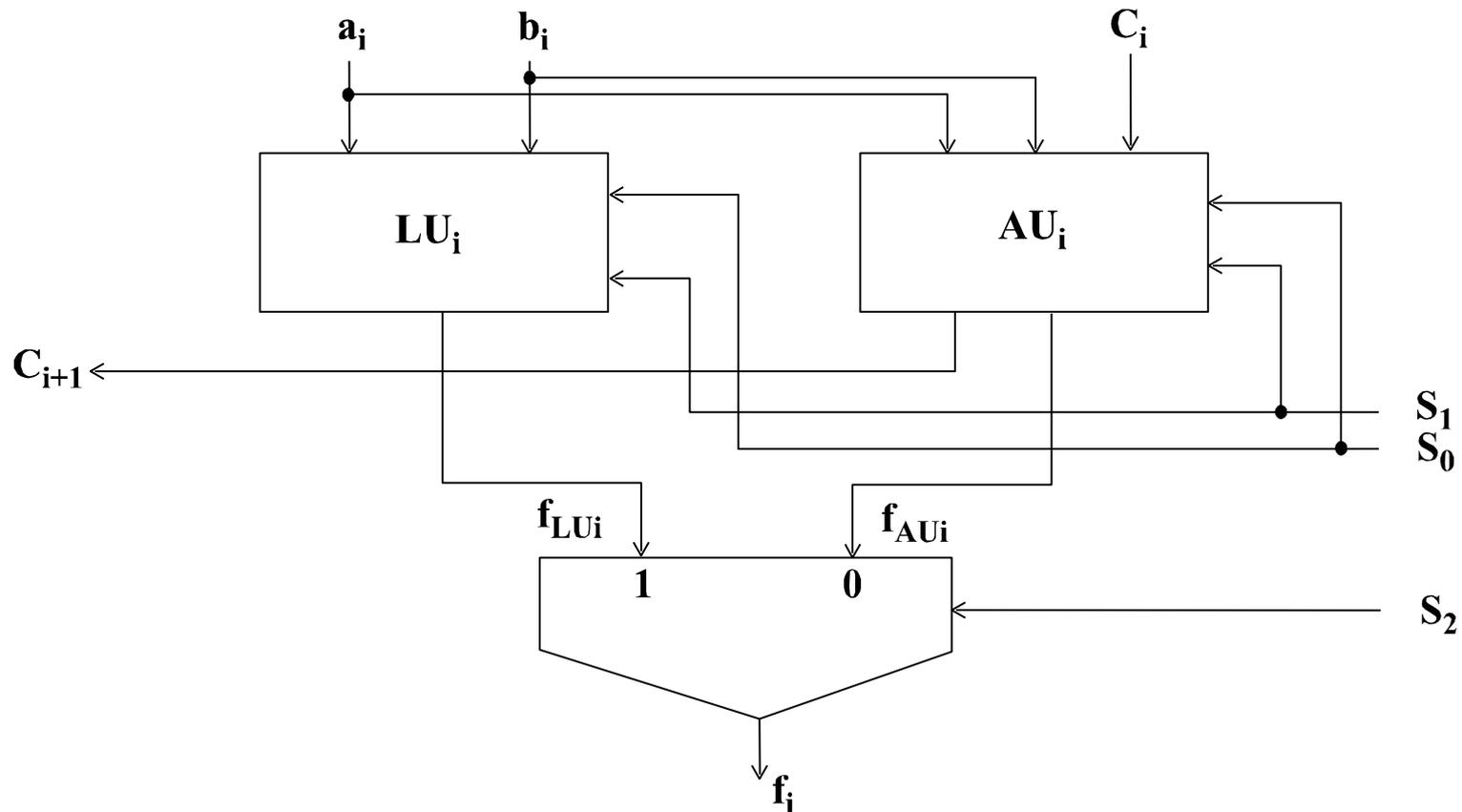
Unidad aritmético-lógica (ALU)

- Realización de una ALU
 - Se separan las partes aritmética (AU) y lógica (LU).



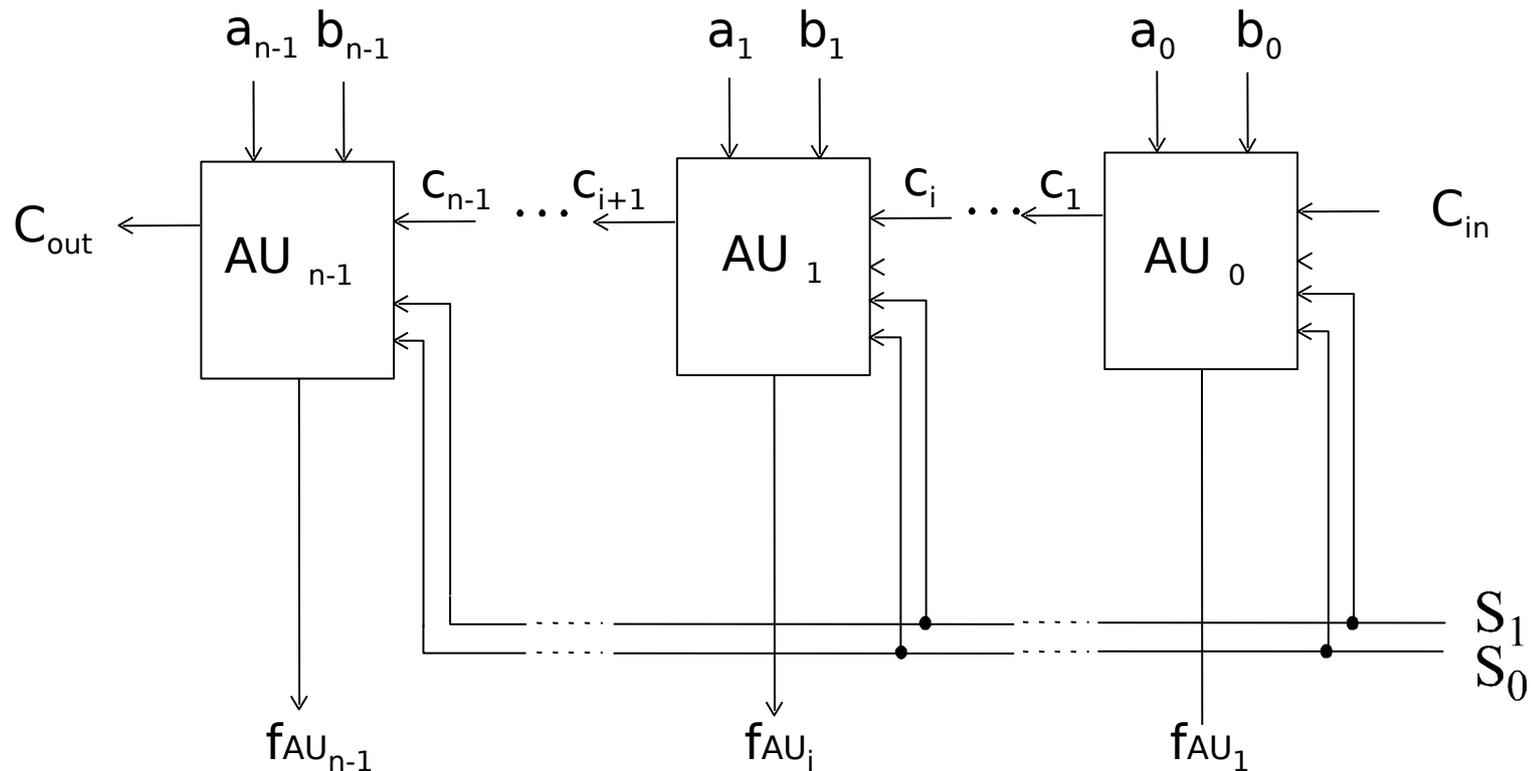
Unidad aritmético-lógica (ALU)

- Realización de una ALU
 - Implica la realización de la ALU para cada pareja de bits entrantes (**etapa típica**)



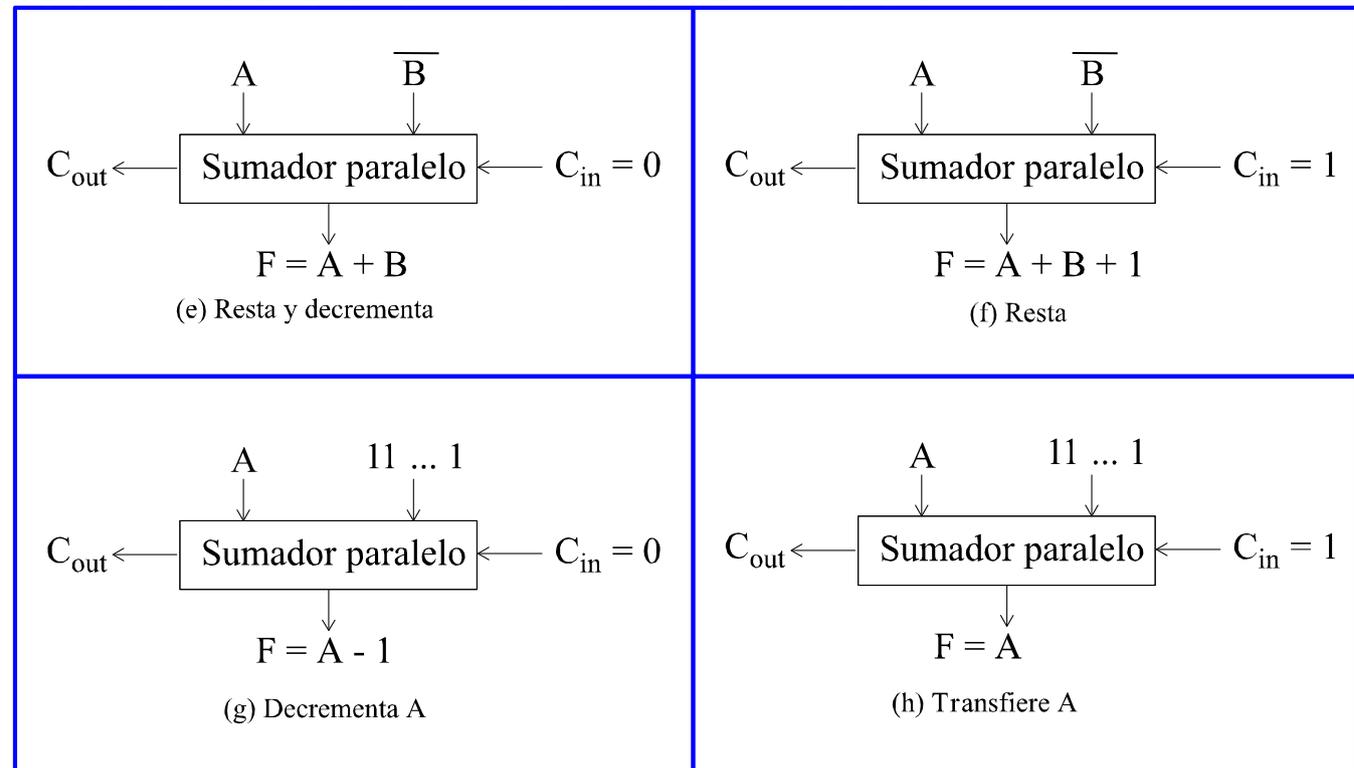
Unidad aritmético-lógica (ALU)

- Realización de la Unidad Aritmética (AU)
 - La AU de n bits se implementa como cascada de n módulos de 1 bit (n etapas típicas):



Unidad aritmético-lógica (ALU)

- Diseño de la **Unidad Aritmética**
 - El bloque aritmético consta básicamente de un sumador.
 - Para obtener las diferentes operaciones se ha de modificar los datos de entrada al sumador



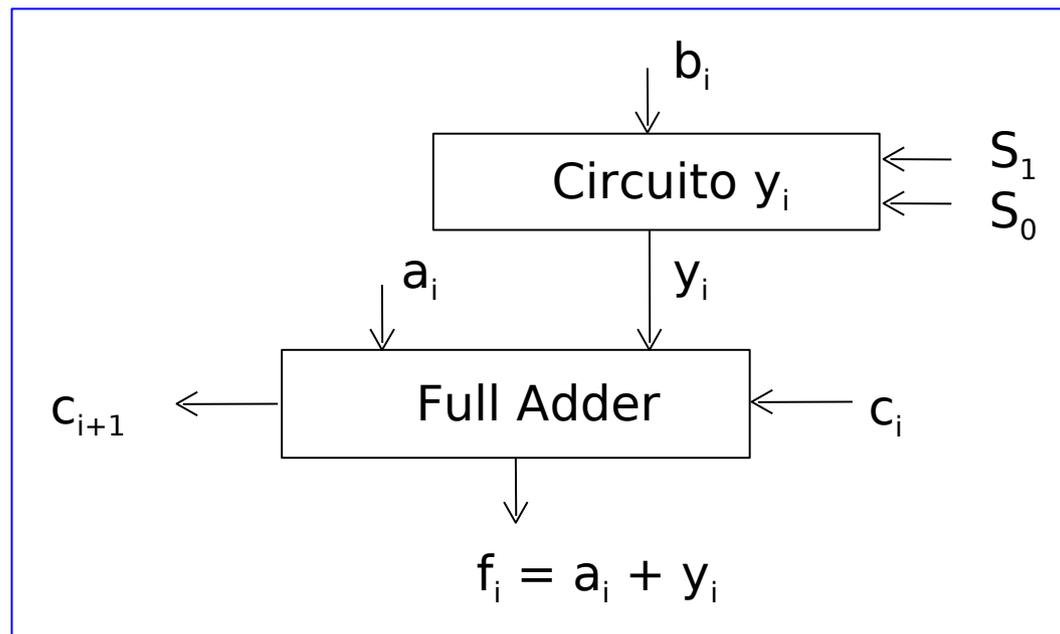
Unidad aritmético-lógica (ALU)

- Diseño de la Unidad Aritmética
 - Datos de entrada al sumador

			$C_{in} = 0$			$C_{in} = 1$		
S_2	S_1	S_0	F	operando izquierdo	operando derecho	F	operando izquierdo	operando derecho
0	0	0	A	A	0	$A + 1$	A	0
0	0	1	$A + B$	A	B	$A + B + 1$	A	B
0	1	0	$A - B - \frac{1}{2} = A + \bar{B}$	A	\bar{B}	$A - B = A + \bar{B} + 1$	A	\bar{B}
0	1	1	$A - 1$	A	11...1	A	A	11...1

Unidad aritmético-lógica (ALU)

- Diseño de la Unidad Aritmética
 - Datos de entrada a la celda básica del sumador sumador

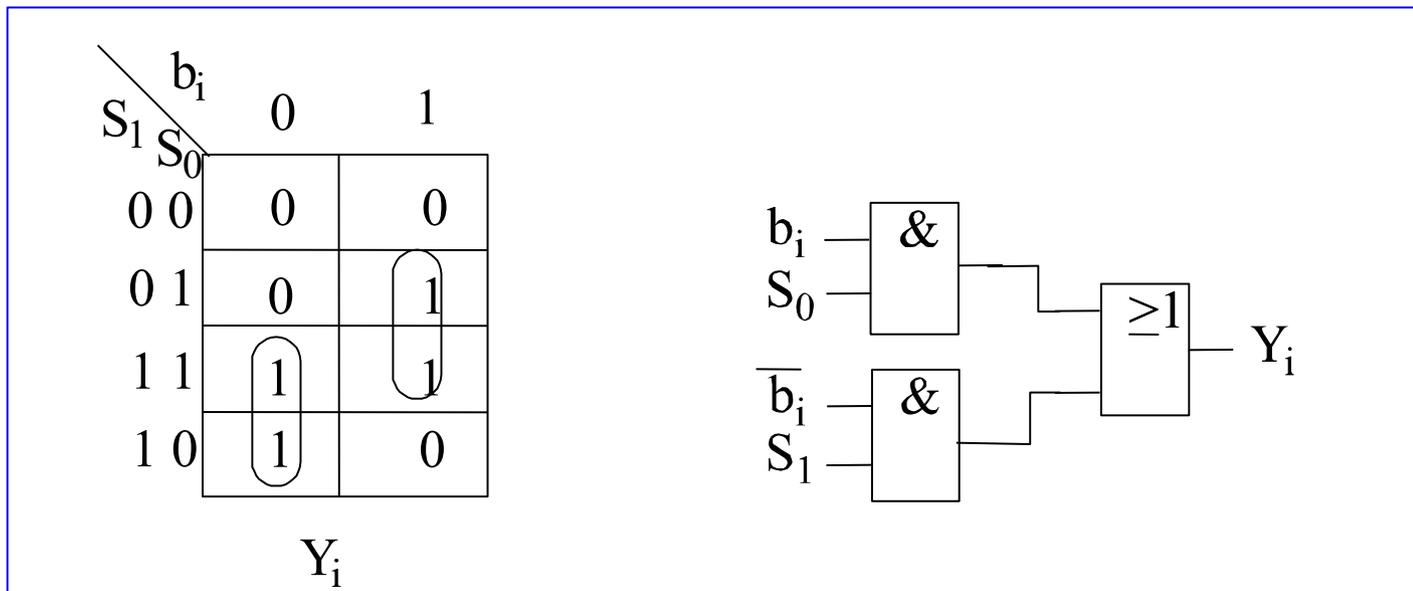


$S_1 S_0$	Y_i
0 0	0
0 1	b_i
1 0	$\overline{b_i}$
1 1	1

Unidad aritmético-lógica (ALU)

- Diseño del “circuito y_i ” con puertas lógicas

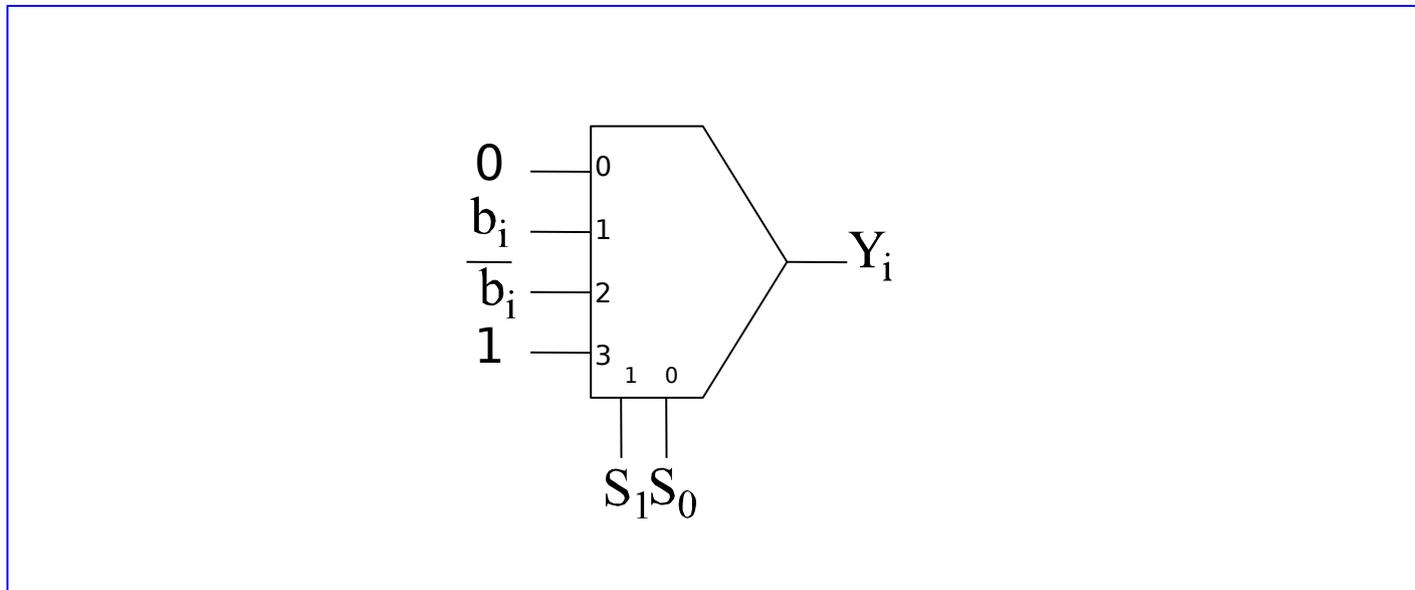
$S_1 S_0$	Y_i
0 0	0
0 1	b_i
1 0	$\overline{b_i}$
1 1	1



Unidad aritmético-lógica (ALU)

- Diseño del “circuito y_i ” con un multiplexor

$S_1 S_0$	Y_i
0 0	0
0 1	b_i
1 0	$\overline{b_i}$
1 1	1



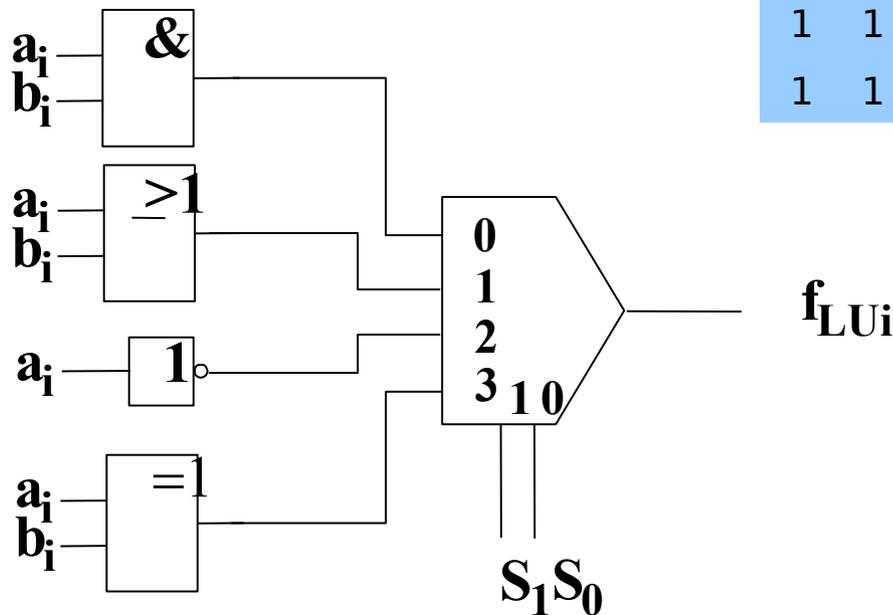
Unidad aritmético-lógica (ALU)

- El acarreo de salida nos puede dar una información muy importante

S_1	S_0	C_{in}		Operación	$C_{out} = 1$ si	Comentario
0	0	0	$F=A$	Transferir A	nunca	$C_{out} = 0$ siempre
0	0	1	$F=A+1$	Incrementar A	$A=2^n-1$	Si $C_{out} = 1 \rightarrow F=0$
0	1	0	$F=A+B$	Sumar A+B	$A+B \geq 2^n$	Overflow si $C_{out} = 1$
0	1	1	$F=A+B+1$	Incrementar A+B	$A+B \geq 2^n - 1$	Overflow si $C_{out} = 1$
1	0	0	$F=A+\bar{B}$	Restar A-B en Ca1	$A_{(BN)} > B_{(BN)}$	Si $C_{out} = 0 \rightarrow A \leq B$ y $F=Ca1(B-A)$ (en notación Ca1 $F=A-B$)
1	0	1	$F=A+\bar{B}+1$	Restar A-B en Ca2	$A_{(BN)} \geq B_{(BN)}$	Si $C_{out} = 0 \rightarrow A < B$ y $F=Ca2(B-A)$ (en notación Ca2 $F=A-B$)
1	1	0	$F=A-1$	Decrementar A	$A \neq 0$	Si $C_{out} = 0 \rightarrow A=0$
1	1	1	$F=A$	Transferir A	siempre	$C_{out} = 1$ siempre

Unidad aritmético-lógica (ALU)

- Diseño de la **Unidad Lógica**
 - Diseño de la etapa típica con un multiplexor y puertas lógicas



S_2	S_1	S_0	Función ALU
1	0	0	$F = A \text{ AND } B$
1	0	1	$F = A \text{ OR } B$
1	1	0	$F = \text{NOT } A$
1	1	1	$F = A \text{ XOR } B$

● Descripción Verilog de una ALU de n bits (1/2)

```
module alu(
    input signed [WIDTH-1:0] a, // primer operando
    input signed [WIDTH-1:0] b, // segundo operando
    input [2:0] op, // entradas de selección de operación
    input cin, // acarreo de entrada en operaciones aritméticas
    output signed [WIDTH-1:0] f, // salida
    output ov, // salida de desbordamiento (overflow) en operaciones aritméticas
    output cout // carry de salida en operaciones aritméticas
);
parameter WIDTH = 8;
reg f, ov;
always @*
begin
/* Nos aseguramos que a cout y ov siempre se les asigne un valor */
    ov = 0;
    cout = 0;
    if (op[2] == 0) // OPERACIONES ARITMÉTICAS
    begin :arith
        reg signed [WIDTH:0] s;
        case (op[1:0])
/* En primer lugar calculamos el resultado en 'f' con el posible bit adicional en 'cout'. El
desbordamiento
* se produce cuando los operandos son del mismo signo y el resultado es de un signo
diferente */
            2'b00: s = a + cin; // a más carry de entrada
            2'b01: s = a + b + cin; // suma a, b y carry de entrada
/* '~' es el operador 'complemento' que invierte todos los bits del operando */
            2'b10: s = a + (~b) + cin; // resta en complemento a 1 y a 2
            2'b11: s = a - 1 + cin; // decremento más carry de entrada
        endcase
    // Cálculo del desbordamiento
        ov = (s[WIDTH] == s[WIDTH-1])? 0: 1;
    // Salidas
        f = s[WIDTH-1:0];
        cout = s[WIDTH];
    end
/* Acaba la descripción del circuito aritmético
```

● Descripción Verilog de una ALU de n bits (2/2)

```
// Comienza la descripción del circuito lógico

    else // OPERACIONES LÓGICAS
    begin
        case (op[1:0])
            2'b00: f = a & b; // AND
            2'b01: f = a | b; // OR
            2'b10: f = ~a; // NOT
            2'b11: f = a ^ b; // XOR
        endcase
    end
// Acaba la descripción del circuito lógico
    end // always
endmodule // alu
// Acaba la descripción de la ALU
```