Unidad 7. Unidades aritméticas y lógicas

Circuitos Electrónicos Digitales
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan <jjchico@dte.us.es> 2010-2018

Contenidos

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de magnitudes
- Números binarios con signo
- Sumador con signo. Desbordamiento
- Sumador/restador
- ALU

Bibliografía

- Lecturas recomendadas
  - LaMeres, 2.3 y 2.4
    - Aritmética binaria y números con signo.
  - LaMeres, 12.1 y 12.2
    - Sumadores y sumadores/restadores.
    - Ejemplos Verilog más detallados que en el tema.
  - curso-verilo.v, unidad 5
    - Ejemplos de diseño de circuitos aritméticos en Verilog.

Bibliografía

- Ampliación
  - Apuntes representación en complemento a 2
    - Demostraciones de las propiedades de la representación en complemento a 2.
  - LaMeres, 12.3 (multiplicación) y 12.4 (división)
    - La multiplicación y división no se trata en el tema. En el libro se introduce de forma sencilla.
  - Floyd, 2.4 a 2.6
    - Aritmética binaria y números con signo.
    - Incluye multiplicación, división y números en coma flotante (no se ve en el tema).
  - Floyd, 6.1 a 6.3
    - Diseño de sumadores.
    - Ejemplos prácticos con dispositivos MSI (no en el tema)

Introducción

- Los circuitos aritméticos hacen operaciones aritméticas sobre datos de n bits: +, -, *, /
- Las operaciones aritméticas son las más importantes en los sistemas digitales (computadores)
- Dos formas de hacer operaciones aritméticas en los computadores:
  - En hardware (mediante circuitos específicos)
    - Mucho más rápido.
    - Operaciones de propósito general.
  - En software (mediante programación)
    - Mucho más lento.
    - Operaciones específicas.

Soporte aritmético hardware en ordenadores personales

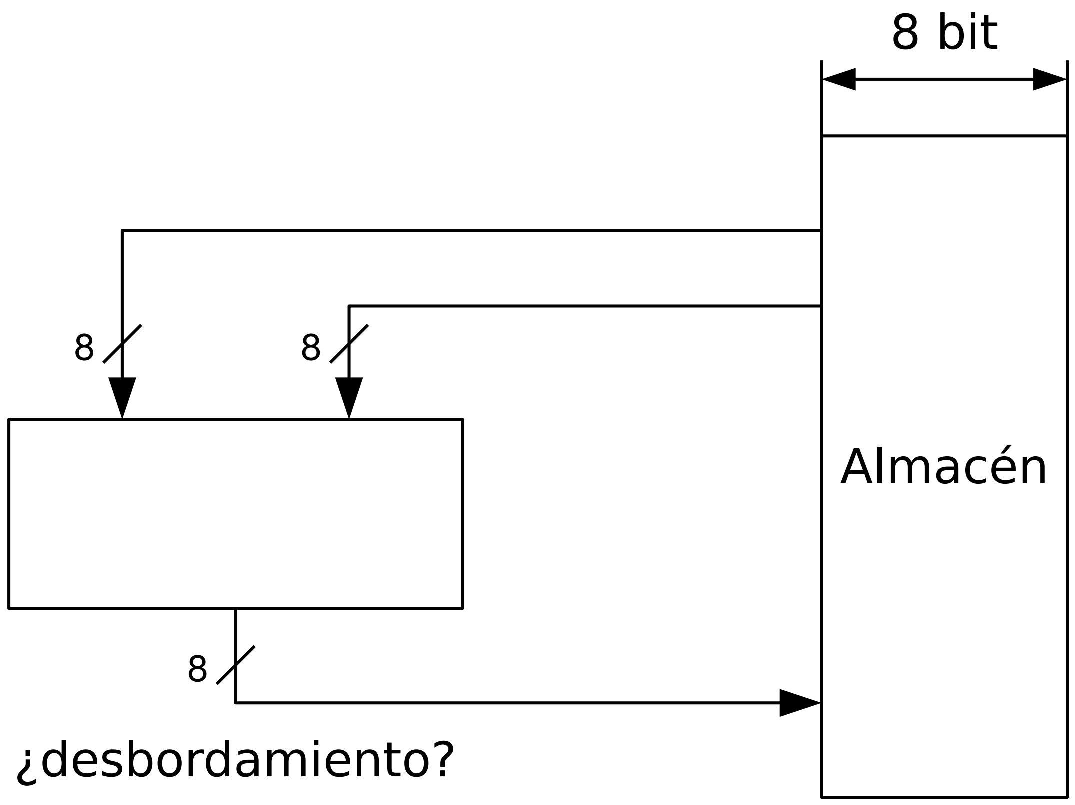
- 1970-1980 (procesadores de 8 bits)
  - Sólo suma y resta de números enteros.
- 1980-1990 (procesadores de 16 bits)
  - Multiplicadores y divisores
  - Co-procesadores matemáticos opcionales: número reales, funciones complejas, etc.
- 1990-2000 (procesadores de 32 bits)
  - Co-procesadores integrados en la CPU
  - Múltiples unidades de enteros: varios cálculos a la vez
  - Operaciones de soporte multimedia
  - Operaciones para gráficos 2-D (en controladores gráficos)
- 2000- (procesadores de 64 bits)
  - Operaciones matemáticas avanzadas
    - Procesamiento digital, simulación física, etc.
  - Operaciones para gráficos 3D (en controladores gráficos)

Contenidos

- Introducción
- **Aritmética binaria**
- Circuitos sumadores básicos
- Sumador de magnitudes
- Números binarios con signo
- Sumador con signo. Desbordamiento
- Sumador/restador
- ALU

Aritmética binaria

- Aritmética usada en sistemas digitales (computadores)
- Basada en el sistema de numeración en base 2
- Número fijo de bits



Aritmética binaria

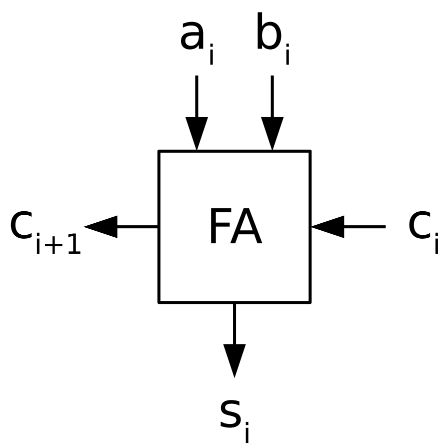
- Ejemplo
  - A = 100110
  - B = 1101
- Operaciones
  - A + B
  - A – B
  - A * B
  - A / B

Contenidos

- Introducción
- Aritmética binaria
- **Circuitos sumadores básicos**
- Sumador de magnitudes
- Números binarios con signo
- Sumador con signo. Desbordamiento
- Sumador/restador
- ALU

Circuitos sumadores básicos. Sumador completo (*full adder*)

$$
\begin{array}{cccccccc}
c_n & c_{n-1} & c_{i+1} & c_i & & c_1 & \\
 & a_{n-1} & \dots & a_i & \dots & a_1 & a_0 \\
+ & b_{n-1} & & b_i & & b_1 & b_0 \\
\hline
 & s_{n-1} & & s_i & & s_1 & s_0
\end{array}
$$



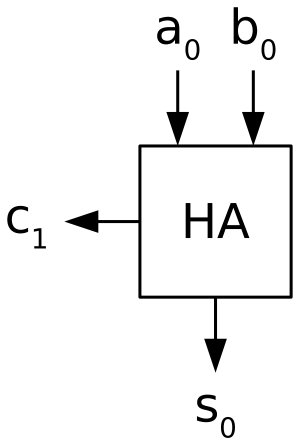
Full Adder

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

$$s_i = a_i \oplus b_i \oplus c_i$$
$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

Circuitos sumadores básicos. Semi sumador (*half adder*)

$$
\begin{array}{ccccccc}
c_n & c_{n-1} & c_{i+1} & c_i & & c_1 & \\
 & a_{n-1} & \dots & a_i & \dots & a_1 & \boxed{a_0} \\
+ & b_{n-1} & & b_i & & b_1 & b_0 \\
\hline
 & s_{n-1} & & s_i & & s_1 & s_0
\end{array}
$$



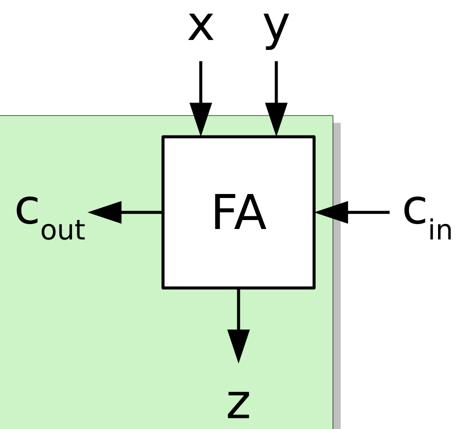
Half Adder

a_0	b_0	c_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

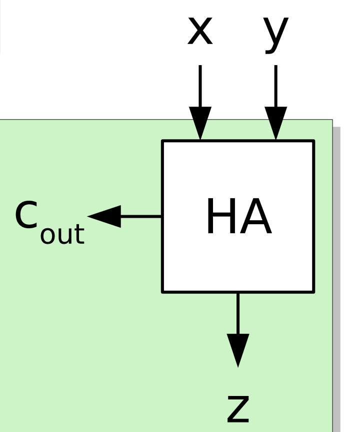
$$
\begin{aligned}
s_0 &= a_0 \oplus b_0 \\
c_1 &= a_0 b_0
\end{aligned}
$$

FA y HA. Descripciones en Verilog

```
module fa(
    input x,
    input y
    input cin,
    output z,
    output cout
    );

    assign z = x ^ y ^ cin;
    assign cout = x&y | x&cin | y&cin;
endmodule // fa
```




```
module ha(
    input x,
    input y
    output z,
    output cout
    );

    assign z = x ^ y;
    assign cout = x&y;
endmodule // ha
```

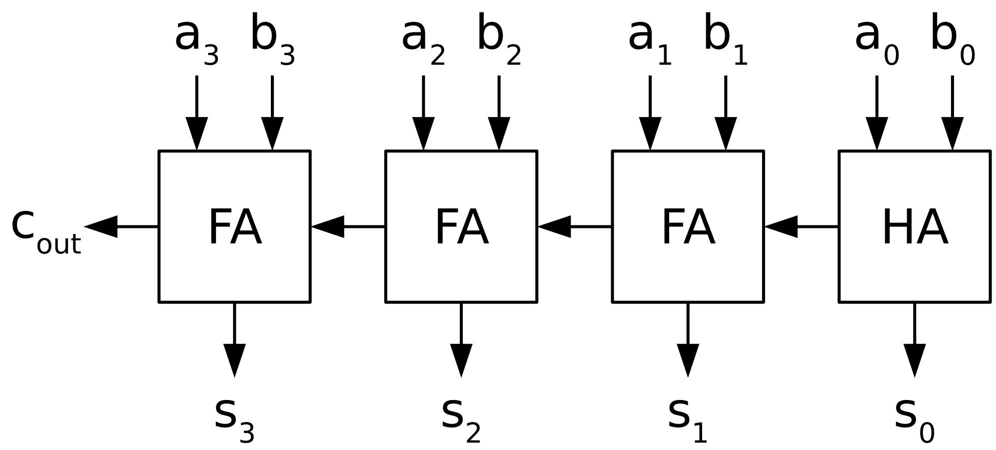


Contenidos

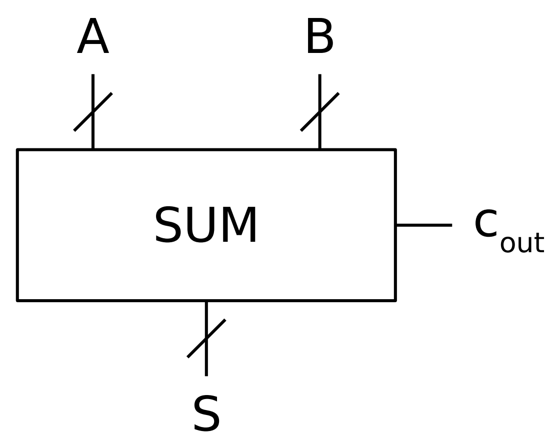
- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- **Sumador de magnitudes**
- Números binarios con signo
- Sumador con signo. Desbordamiento
- Sumador/restador
- ALU

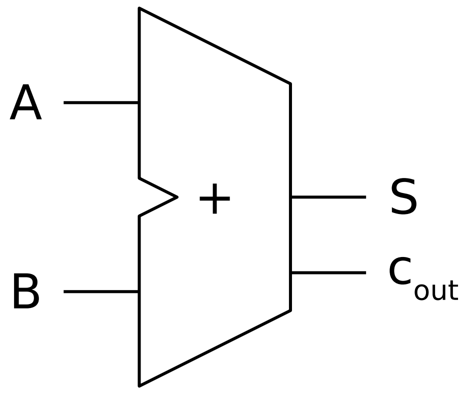
Sumador de magnitudes de n bits

$$
\begin{array}{cccccc}
 & c_{out} & c_3 & c_2 & c_1 & \\
 & & a_3 & a_2 & a_1 & a_0 \\
+ & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & s_3 & s_2 & s_1 & s_0
\end{array}
$$



$$\boxed{S = (A + B) \bmod 2^n}$$

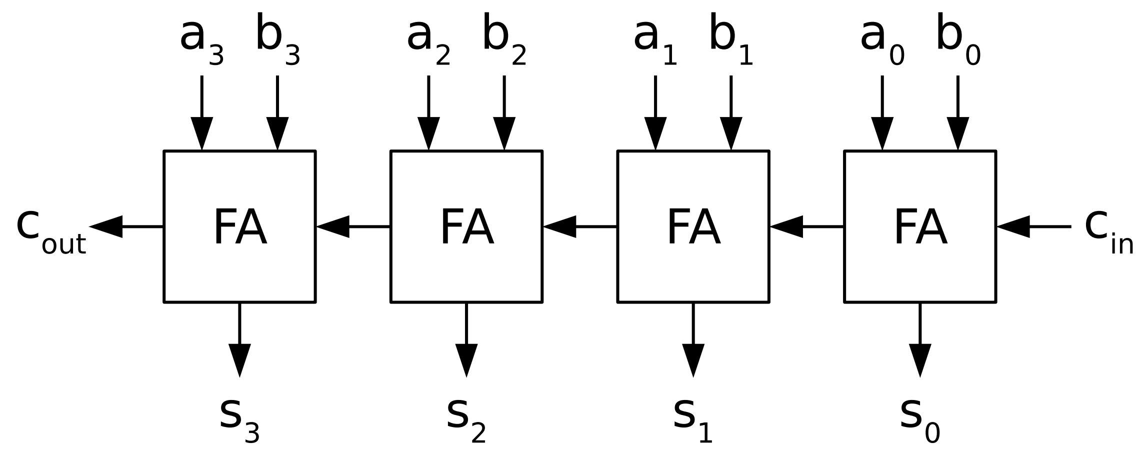




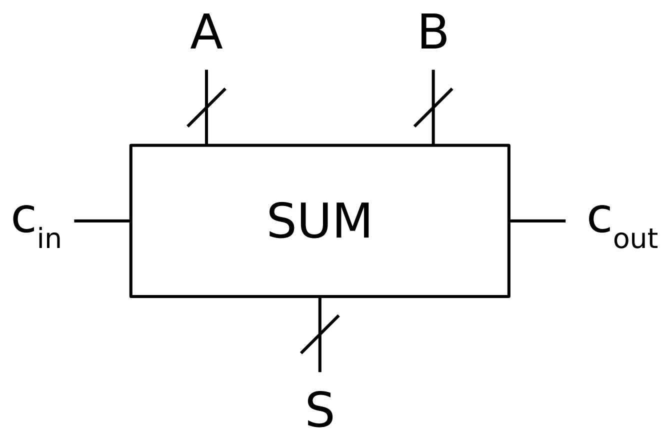
c_{out} es un indicador de “desbordamiento” (*overflow*): el resultado no puede represetnarse con los bits disponibles (n).

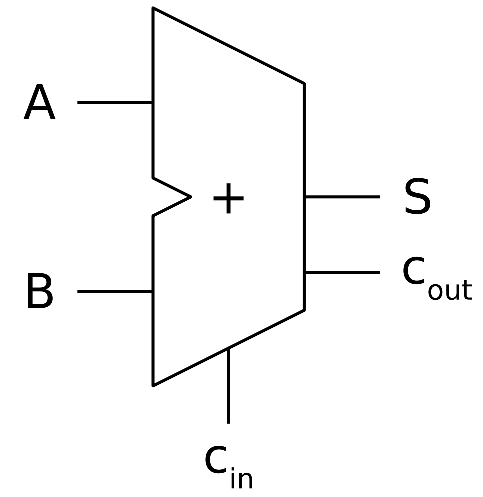
Sumador de magnitudes de n bits con entrada de acarreo

$$
\begin{array}{rccccc}
 & c_{out} & c_3 & c_2 & c_1 & c_{in} \\
 & & a_3 & a_2 & a_1 & a_0 \\
+ & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & s_3 & s_2 & s_1 & s_0
\end{array}
$$



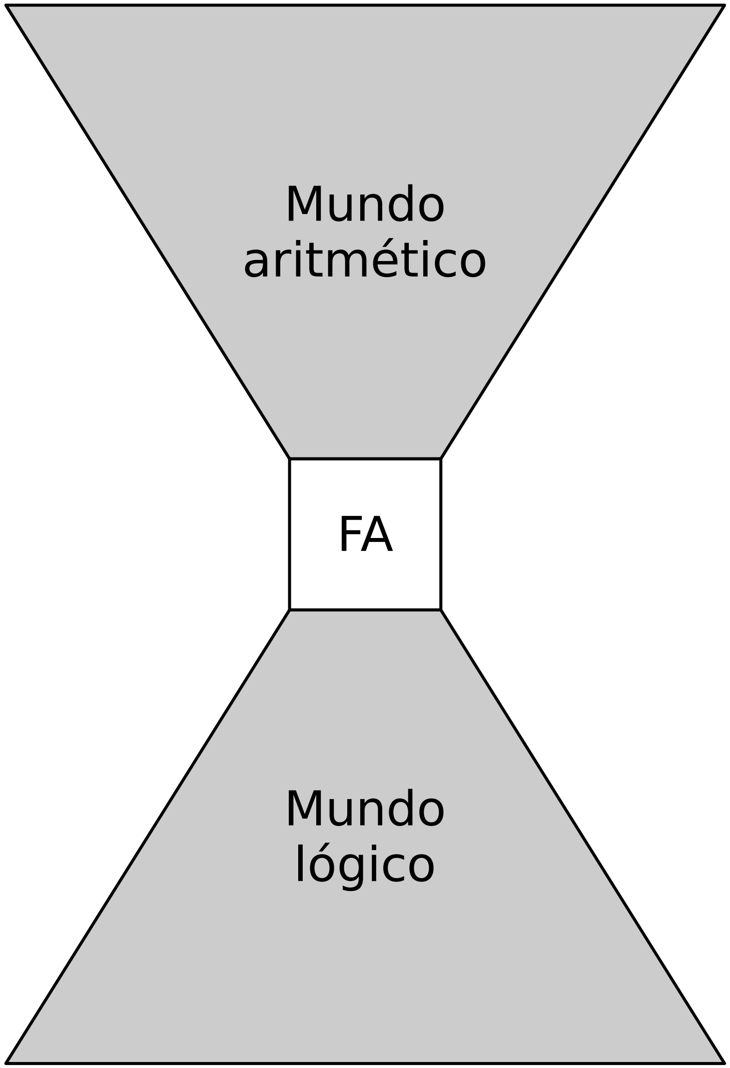
$$\boxed{S = (A + B + C_{in}) \bmod 2^n}$$





c_{in} es útil para conectar varios sumadores y sumar números de más de n bits.

Importancia del sumador completo (FA)

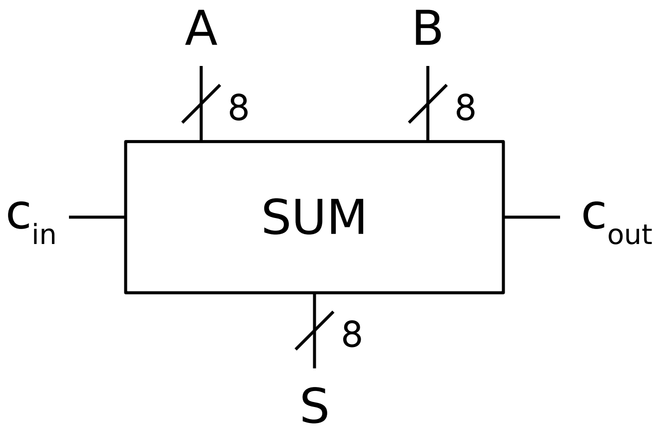


- El sumador completo hace la operación aritmética más básica (sumar 3 bits) usando sólo operadores lógicos.
- ”El sumador completo es el punto de encuentro entre el mundo lógico de los sistemas digitales y el mundo aritmético de los computadores” J. Juan

Ejemplos Verilog

Usando sumadores completos (FA)

```
module adder8_fa(
    input wire [7:0] a,
    input wire [7:0] b,
    input wire cin,
    output wire [7:0] s,
    output wire cout
    );

    // auxiliary signal
    wire [7:1] c;

    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]);
    fa fa7 (a[7], b[7], c[7], s[7], cout);

endmodule   // adder8_fa
```



Usando operadores aritméticos

```
module adder8(
    input [7:0] a,
    input [7:0] b,
    input cin,
    output [7:0] s,
    output cout
    );

    assign
        {cout, s} = a+b+cin;

endmodule // adder8
```

Ejemplos

- Ejemplo 1: Diseña un circuito con una entrada x de 8 bits y una salida z de 8 bits, de forma que z = x + 73. Emplea circuitos sumadores básicos (FA y HA).
- Ejemplo 2: Diseña un circuito con una entrada x de 8 bits y una salida z de 8 bits, de forma que z = 2 * x. Emplea circuitos sumadores básicos (FA y HA).
- Ejemplo 3: Diseña un circuito con una entrada x de 8 bits y una salida z de 8 bits, de forma que z = 5 * x. Proporciona dos soluciones:
  - Empleando circuitos sumadores básicos (FA y HA).
  - Empleando sumadores de magnitud

Ejemplo 4

- Un termómetro en una cámara frigorífica proporciona la temperatura de la cámara mediante un número x de 4 bits. Diseña un circuito que proporcione una lectura de la temperatura de la cámara en formato decimal para ser representada en dos visores de 7 segmentos.
  - Entradas:
    - x[3:0]
  - Salidas:
    - seg1[1:7]: cifra decimal más significativa.
    - seg0[1:7]: cifra decimal menos significativa.

Contenidos

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de magnitudes
- **Números binarios con signo**
- Sumador con signo. Desbordamiento
- Sumador/restador
- ALU

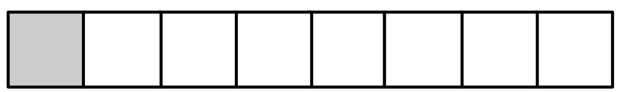
¿Y los números negativos? Números binarios con signo

- En los circuitos digitales no hay "signo", sólo ceros y unos.
- El signo debe codificarse mediante bits junto con la palabra que representa al número.
- Hay varias alternativas para codificar números con signo:
  - Representación signo-magnitud
  - Representación en exceso
  - Representaciones en complemento
- Representación en complemento a 2: usada por la práctica totalidad de computadores actuales para números enteros.

Representación signo-magnitud (s-m) con n bits

- Emplea un bit para el signo y el resto para la magnitud:
  - Signo: 0(+), 1(-)
  - Números representables: 2^n-1
  - Dos representaciones del "0": 00000000, 10000000

sign magnitude



0 1 0 1 1 0 1 0 $\rightarrow$ $+90_{(10}$

1 0 1 0 1 0 0 1 $\rightarrow$ $-41_{(10}$

$$-\left(2^{n-1}-1\right)\leqslant x\leqslant 2^{n-1}-1$$

Representación signo-magnitud con n bits

- Ventajas
  - Fácil de entender
  - Fácil de obtener el opuesto
- Inconvenientes
  - Para operar con números en s-m hay que determinar previamente su signo.
  - La operación a realizar depende del signo de los operandos.
  - Se requieren circuitos complejos para operar con número representados en s-m.
- Usos
  - No usado en la práctica para números enteros.
  - Un concepto similar se usa en la representación de números reales (punto flotante).

Binario	Positivo	s-m
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	0
1001	9	-1
1010	10	-2
1011	11	-3
1100	12	-4
1101	13	-5
1110	14	-6
1111	15	-7

Representación en exceso o sesgada

- Dado un número x, y un exceso e, la representación en exceso-e con n bits consiste en representar x mediante la codificación en binario natural de la magnitud x+e con n bits.
- Para que la representación sea correcta, el resultado debe ser un entero positivo representable con n bits.
- Con n bits, un valor frecuente para el exceso es 2^{n-1}.
  - Aproximadamente mismo número de positivos y negativos
  - El primer bit de la palabra indica el signo: 0-negativo, 1-positivo.
- Ej: exceso-2^{n-1} (n=8 → 2^{n-1}=128)
  - $-35_{(10} \rightarrow -35+128 = 93 = 01011101_{(2}$
  - $-35_{(10} = 01011101_{exc-128}$

$$0 \leqslant x+e < 2^n$$

$$e = 2^{n-1}$$

$$0 \leqslant x+2^{n-1} < 2^n$$

$$-2^{n-1} \leqslant x < 2^{n-1}$$

Representación en exceso o sesgada

- Ventajas
  - Fácil de convertir entre el número y la representación
- Inconvenientes
  - No es fácil obtener el opuesto
  - Se necesitan circuitos complejos para operar con números en representación en exceso.
- Ejemplo de uso
  - Exponente en la representación de números reales (punto flotante)

Binario	Positivo	Exceso-8
0000	0	-8
0001	1	-7
0010	2	-6
0011	3	-5
0100	4	-4
0101	5	-3
0110	6	-2
0111	7	-1
1000	8	0
1001	9	1
1010	10	2
1011	11	3
1100	12	4
1101	13	5
1110	14	6
1111	15	7

Representaciones en complemento con n bits

- Las representaciones en complemento usan una operación de transformación (complemento) para representar los números negativos.
- La transformación se construye de forma que el bit más significativo (msb) sea 0 para positivos y 1 para negativos.
- Representación de números positivos:
  - Se representan en binario natural
  - El bit más significativo debe ser 0
- Representación de números negativos
  - Se representan haciendo la operación “complemento” al opuesto (positivo)
- Cambio de signo
  - La representación del opuesto de un número se obtiene haciendo la operación “complemento” a la representación del número.
- Operaciones complemento típicas
  - Complemento a dos, complemento a uno, complemento a la base, etc.

Representación en complemento a 1 (RC1)

- Emplea la operación de complemento a 1 (complementar todos los bits).
- El primer bit indica el signo.
- Hace años la emplearon algunos ordenadores, pero hoy está en desuso.
- Ventajas
  - Facilidad para obtener el opuesto
  - Circuitos más simples que con s-m
- Inconvenientes
  - Dos representaciones del cero
  - Circuitos para operar con RC1

Binario	Positivo	RC1
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-7
1001	9	-6
1010	10	-5
1011	11	-4
1100	12	-3
1101	13	-2
1110	14	-1
1111	15	0

$$-(2^{n-1}-1) \leqslant x \leqslant 2^{n-1}-1$$

Complemento a 1

- **Definición (Operación Complemento a 1 con n bits)**: Dado un entero positivo $x < 2^n$, se define el complemento a 1 con n bits de x, $C1_n(x)$, a la magnitud que resulta de complementar todos los bits de x expresado en base 2.
- **Definición (Representación en Complemento a 1 con n bits -RC1n-)**: Dado un entero x tal que $-2^{n-1} < x < 2^{n-1}$, la representación en complemento a 1 de x con n bits (RC1n) es una palabra binaria de n bits de magnitud $RC1_n(x)$ tal que:
  - $RC1_n(x) = x$, si $0 \leq x < 2^{n-1}$
  - $RC1_n(x) = C1_n(-x)$, si $-2^{n-1} < x < 0$
- **Definición (Representabilidad en complemento a 1)**: Si $x < -(2^{n-1} - 1)$ o $x > 2^{n-1}-1$ se dice que x no es representable en complemento a 1 con n bits.

Representación en complemento a 2

- Aprovecha la "forma" de sumar del sumador de magnitudes
  - $s = (a+b) \bmod 2^n$
- Se puede sumar un número negativo utilizando uno positivo:
  - $5 + (-3) = 2$
  - $(5 + 13) \bmod 16 = 18 \bmod 16 = 2$
- En general, si $x<0$, basta sustituirlo por $x+2^n$
- Se imponen límites para distinguir positivos de negativos
  - $0 \ldots 2^{n-1}-1 \rightarrow$ positivos (msb=0)
  - $2^{n-1} \ldots 2^n-1 \rightarrow$ negativos (msb=1)
- Funciona para todos las combinaciones de a y b, salvo ¡desbordamiento!

Binario	Positivo	Negativo	RC2
0000	0	0	0
0001	1	-15	1
0010	2	-14	2
0011	3	-13	3
0100	4	-12	4
0101	5	-11	5
0110	6	-10	6
0111	7	-9	7
1000	8	-8	-8
1001	9	-7	-7
1010	10	-6	-6
1011	11	-5	-5
1100	12	-4	-4
1101	13	-3	-3
1110	14	-2	-2
1111	15	-1	-1

Complemento a 2

- **Definición (Representación en Complemento a 2 con n bits -RC2n-)**: Dado un entero x tal que $-2^{n-1} \leq x < 2^{n-1}$, la representación en complemento a 2 de x con n bits (RC2n) es una palabra binaria de n bits de magnitud $RC2_n(x)$ tal que:
  - $RC2_n(x) = x$, si $0 \leq x < 2^{n-1}$
  - $RC2_n(x) = 2^n + x$, si $-2^{n-1} \leq x < 0$
- **Definición (Representabilidad en complemento a 2)**: Se dice que x es representable en complemento a 2 con n bits si $-2^{n-1} \leq x \leq 2^{n-1}-1$
- **Definición (Operación Complemento a 2 con n bits)**: Dado un entero positivo $x < 2^n$, se define el complemento a 2 con n bits de x, $C2_n(x)$, como:
  - $C2_n(x) = 2^n - x$
- Representación en Complemento a 2 reformulada:
  - $RC2_n(x) = x$, si $0 \leq x < 2^{n-1}$
  - $RC2_n(x) = C2_n(-x)$, si $-2^{n-1} \leq x < 0$

Demostraciones

Complemento a 2

- **Teorema (Bit de signo)**: Si x es representable en complemento a 2 con n bits, el bit más significativo de la representación en complemento a 2 de x es 0 si $x \geq 0$, y 1 si $x < 0$.
- **Teorema (Cálculo del opuesto)**: Dado un entero x representable en complemento a 2 con n bits, la magnitud de la RC2n de -x es el complemento a 2 de la magnitud de la RC2n de x, siempre que -x sea representable en complemento a 2, esto es:
  - $RC2_n(-x) = C2_n(RC2_n(x))$
  - En RC2n el opuesto se calcula aplicando la operación complemento a 2 sobre la representación.

Complemento a 2

- **Teorema (Regla de la suma)**: Dados dos enteros a y b tales que a, b y a+b son representables en complemento a 2 con n bits, la magnitud de la RC2n de a+b se puede calcular como:
  - $RC2_n(a+b) = [RC2_n(a) + RC2_n(b)] \bmod 2^n$

  Esto es: la $RC2_n$ de a+b se obtiene sumando las RC2n de a y de b y despreciando posibles bits de acarreo.
- **Corolario**: Un sumador de magnitudes de n bits cuyos operandos son las RC2n de a y b produce la RC2n de a+b, siempre que ésta sea representable con n bits.

Complemento a 2

- **Definición (Desbordamiento en complemento a 2)**: Dados dos enteros a y b representables en complemento a 2 con n bits, se dice que la suma de a y b produce desbordamiento en complemento a 2 con n bits si a+b no es representable en complemento a 2 con n bits.
- **Corolario (Regla del desbordamiento)**: Dados dos enteros a y b representables en complemento a 2 con n bits, la suma a+b es representable en complemento a 2 con n bits si y sólo si:
  - a y b tienen distinto signo o al menos uno de ellos es cero, o bien
  - a y b tienen el mismo signo y el resultado de la suma de las RC2n de a y b, módulo 2^n, tiene el mismo bit de signo que las RC2n de a y b.

Complemento a 2
Suma y desbordamiento

```
 1001 = -7
 0101 = +5
-----
 1110 = -2
```

```
 1100 = -4
 0100 = +4
-----
10000 =  0
```

```
 0011 = +3
 0100 = +4
-----
 0111 = +7
```

```
 1100 = -4
 1111 = -1
-----
11011 = -5
```

```
 0101 = +5
 0100 = +4
-----
 1001 = -7
```

```
 1001 = -7
 1010 = -6
-----
10011 = +3
```

¡Desbordamiento!

Complemento a 2
Relación con el complemento a 1

- **Teorema**: el complemento a 1 con n bits de x puede calcularse como:
  - $C1_n(x) = 2^n - x - 1$
- Relación:
  - $C1_n(x) = C2_n(x) - 1$
  - $C2_n(x) = C1_n(x) + 1$
- Regla 1 (cálculo rápido del C2)
  - El $C2_n(x)$ puede obtenerse complementando todos los bits de x y sumando 1 al resultado.
- Regla 2 (cálculo super-rápido del C2)
  - El $C2_n(x)$ puede obtenerse conservando todos los bits de x que sean '0' comenzando por el menos significativo hasta el primer '1' inclusive y complementando el resto de bits.

Complemento a 2
Ejemplos

- Ejemplo 1: representar las siguientes cantidades en C2 con 8 bits.
  - 32, -13, 115, -140, 128, -128
- Ejemplo 2: obtener el número mínimo de bits necesarios para representar las cantidades anteriores en C2.
- Ejemplo 3: calcular el valor decimal de las siguientes representaciones en C2.
  - 01001100, 11110000

Complemento a 2
Extensión del signo

- **Teorema (Extensión del signo en complemento a 2)**: Sea x un entero representable en complemento a 2 y $RC2_n(x)$ la magnitud de su representación en complemento a 2 con n bits y s el bit de signo de dicha magnitud. Se cumple que:
  - $RC2_{n+1}(x) = s\,2^n + RC2_n(x)$

  Esto es, la representación en complemento a 2 de x con n+1 bits coincide con la representación con n bits añadiendo un bit de signo igual al bit de signo de la representación con n bits.
- **Corolario**: Un entero x representable en complemento a 2 con n bits será representable en complemento a 2 con n-1 bits si los dos bits más significativos de RC2n(x) son iguales (el signo no cambia al reducir el número de bits).

Complemento a 2
C2 como código pesado

- **Teorema**: Sea x un entero representable en complemento a 2 con n bits, $RC2_n(x)$ la magnitud de su representación en complemento a 2 con n bits formada por las cifras binarias $\{x_0,x_1,\ldots,x_{n-1}\}$. Se tiene que:

$$X = -2^{n-1}x_{n-1} + 2^{n-2}x_{n-2} + \ldots + 2x_1 + x_0$$

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
-128	64	32	16	8	4	2	1		
1	0	1	1	0	1	1	0	$\longrightarrow$	-74
1	1	1	1	1	1	1	0	$\longrightarrow$	-2
0	1	0	0	0	0	0	1	$\longrightarrow$	65

Resumen de números con signo

x	S-M	Exc-2^{n-1}	RC1	RC2
-8	-	0000	-	1000
-7	1111	0001	1000	1001
-6	1110	0010	1001	1010
-5	1101	0011	1010	1011
-4	1100	0100	1011	1100
-3	1011	0101	1100	1101
-2	1010	0110	1101	1110
-1	1001	0111	1110	1111
0	0000/1000	1000	0000/1111	0000
1	0001	1001	0001	0001
2	0010	1010	0010	0010
3	0011	1011	0011	0011
4	0100	1100	0100	0100
5	0101	1101	0101	0101
6	0110	1110	0110	0110
7	0111	1111	0111	0111

Contenidos

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de magnitudes
- Números binarios con signo
- **Sumador con signo. Desbordamiento**
- Sumador/restador
- ALU

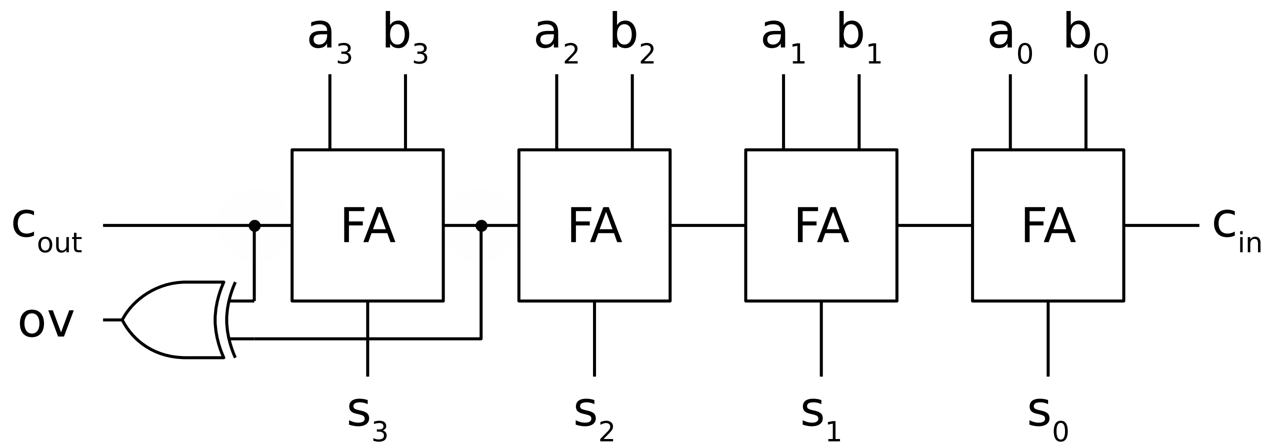
Sumador con signo: desbordamiento

- Mismo sumador que para magnitudes (propiedades de rep. C2)
- El bit de acarreo NO indica desbordamiento en C2.
- Se necesita un indicador de desbordamiento para la suma en C2. Basado en la regla del desbordamiento.
  - Signo de operandos y resultado

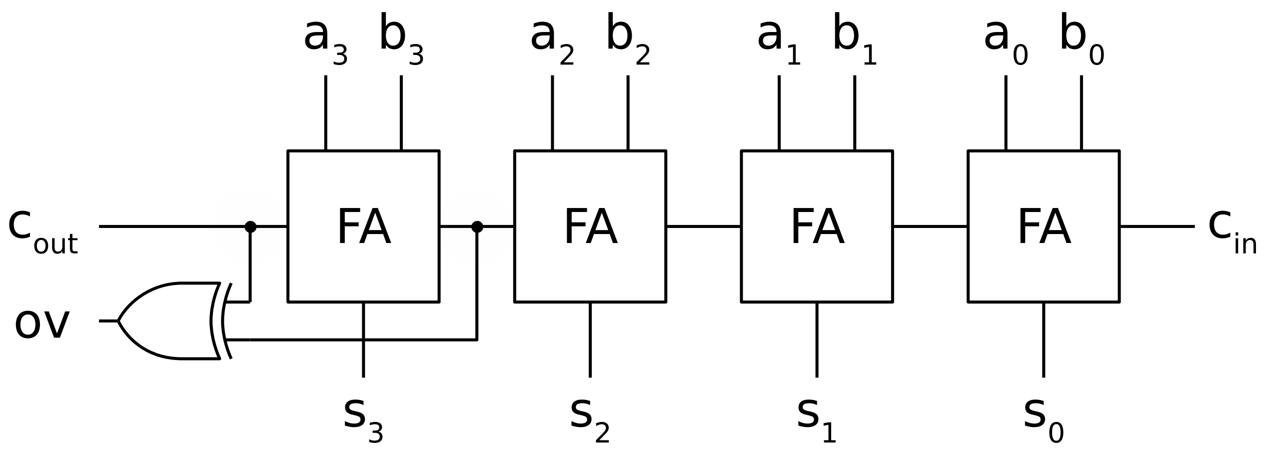
```
   0  1
      0  ...
+     0  ...
---------
      1  ...

   1  0
      1  ...
+     1  ...
---------
      0  ...
```

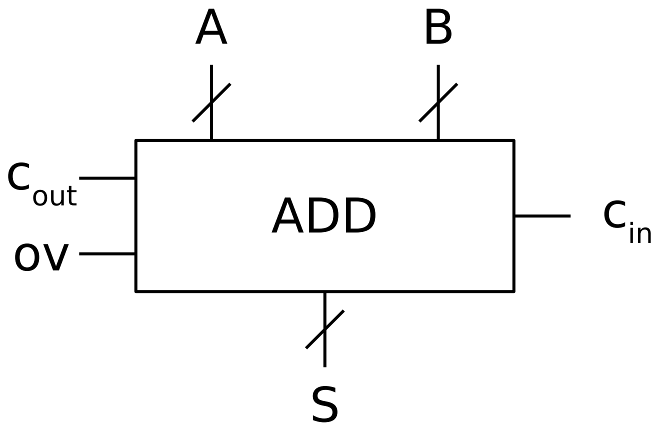
$$ov = \overline{a}_{n-1}\,\overline{b}_{n-1}\,s_{n-1} + a_{n-1}\,b_{n-1}\,\overline{s}_{n-1}$$

$$ov = c_n \oplus c_{n-1}$$



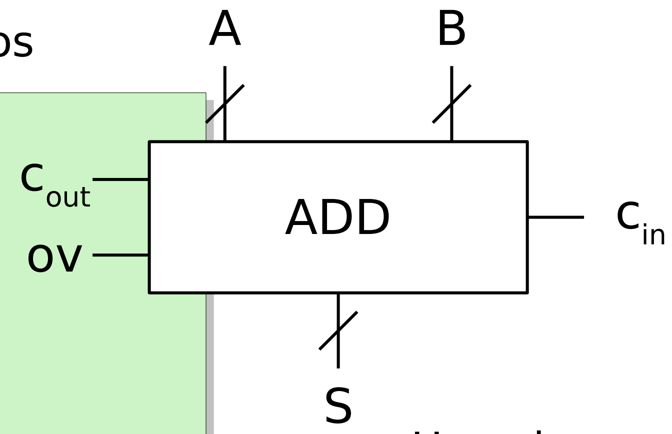
Sumador con signo: desbordamiento





Sumador con/sin signo Ejemplos Verilog

Usando sumadores completos




```
module adder8_fa(
    input [7:0] a,
    input [7:0] b,
    input cin,
    output [7:0] s,
    output cout, ov
    );

    // auxiliary signal
    wire [7:1] c;

    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]);
    fa fa7 (a[7], b[7], c[7], s[7], cout);

    assign ov = c[7] ^ cout;

endmodule   // adder8_fa
```

Usando operadores aritméticos

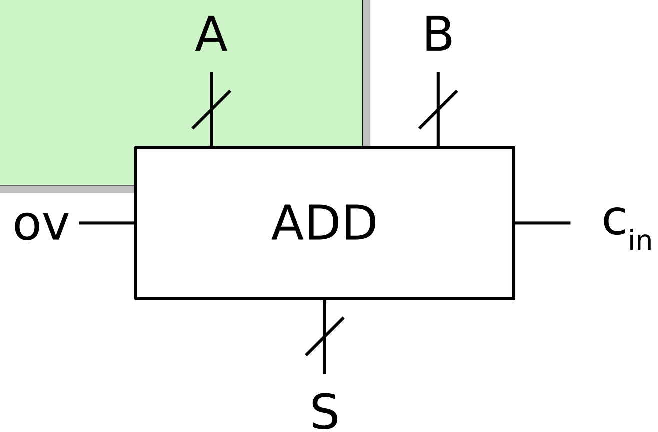
```
module adder8(
    input [7:0] a,
    input [7:0] b,
    input cin,
    output [7:0] s,
    output cout, ov
    );

    assign {cout, s} = a+b+cin;

    assign ov = ~a[7] & ~b[7] & s[7]
              | a[7] & b[7] & ~s[7];

endmodule // adder8
```

Sumador con signo
Ejemplos Verilog

```
module adder8 #(parameter N = 8)(
    input wire signed [N-1:0] a,
    input wire signed [N-1:0] b,
    output reg signed [N-1:0] z,
    output reg ov
    );

    reg signed [N:0] f;

    always @* begin
        f = a + b;

        if (f[N] != f[N-1])
            ov = 1;
        else
            ov = 0;

        z = f[N-1:0];
    end
endmodule // adder8
```

- Tipo “signed”. Maneja automáticamente representación en C2.
  - Constantes negativas
  - Extensión de signo
  - Etc.

```
if (f[N] != f[N-1])
    ov = 1;
else
    ov = 0;
```



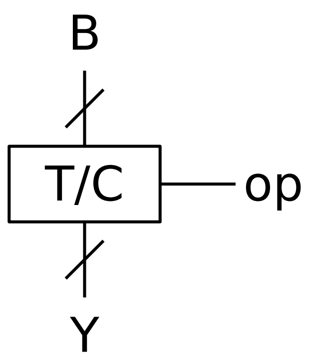

```
ov = (f[N]==f[N-1])? 0: 1;
```

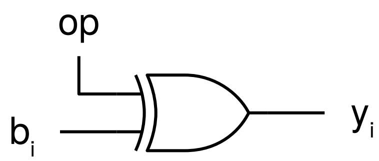
```
ov = f[N] ^ f[N-1]);
```

Contenidos

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de magnitudes
- Números binarios con signo
- Sumador con signo. Desbordamiento
- **Sumador/restador**
- ALU

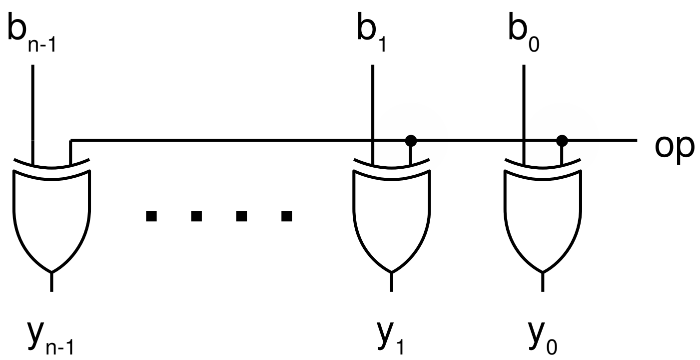
Sumador/restador
Bloque transfiere/complementa





b_i \ op	0	1
0	0	1
1	1	0

y_i



$Y = \overline{B} = C1_n(B) = 2^n\text{-}B\text{-}1 = C2_n(B)\text{-}1$

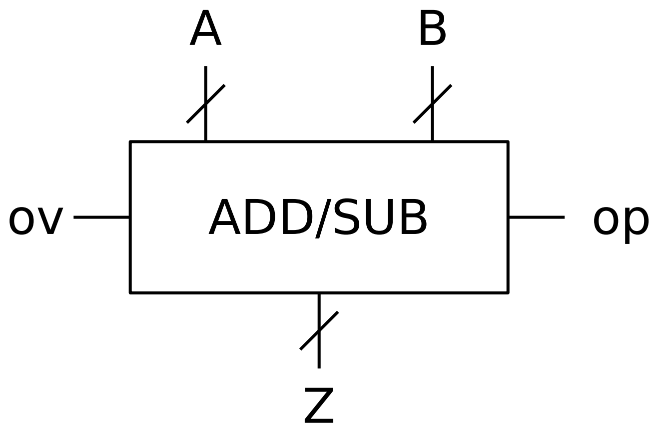
Si $B=RC2_n(b)$, entonces:

$Y+1 = C2_n(B) = RC2_n(-b)$

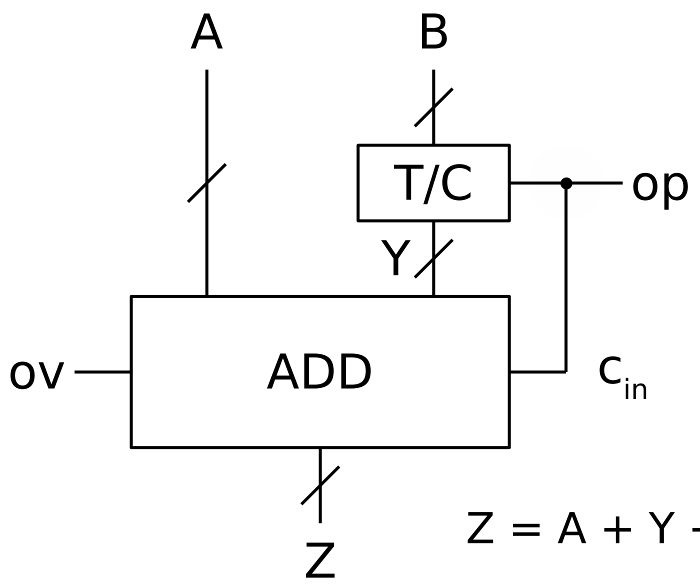
$\boxed{\text{Notación: } X = RC2_n(x)}$

Sumador/restador

- Sumador/restador en complemento a 2
  - $A=RC2_n(a)$, $B=RC2_n(b)$, $Z=RC2_n(z)$
  - ov: salida de desbordamiento (z no representable en C2)



op	z	Z
0	a + b	A + B
1	a - b	A+C2(B)

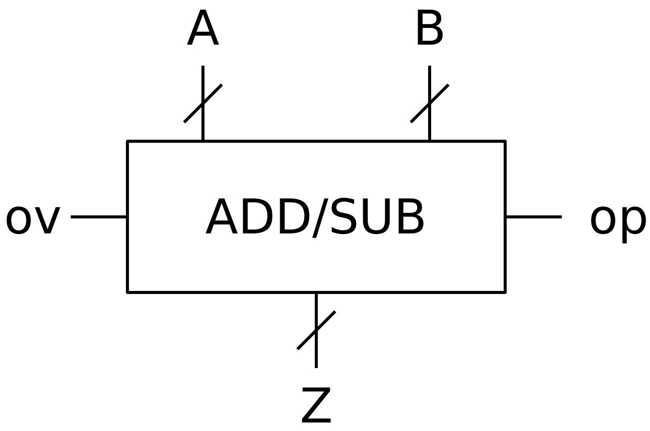


Z = A + Y + cin

op	Y	c_{in}	Z	z
0	B	0	A + B	a + b
1	$\overline{B}$	1	$A+\overline{B}+1$	a - b

$\overline{B}+1 = C1_n(B)+1 = C2_n(B)$

Sumador/restador
Descripción Verilog



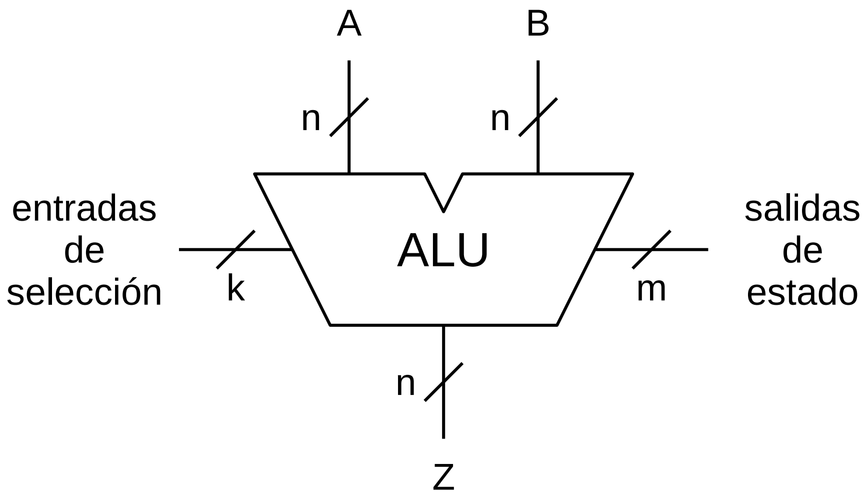

```
module addsub #(parameter N = 8)(
    input wire signed [N-1:0] a,
    input wire signed [N-1:0] b,
    input wire op,
    output reg signed [N-1:0] z,
    output reg ov
    );

    reg signed [N:0] f;

    always @* begin
        case (op)
          0:
            f = a + b;
          default:
            f = a - b;
        endcase

        ov = f[N] ^ f[N-1]);

        z = f[N-1:0];
    end
endmodule // addsub
```

Contenidos

- Introducción
- Aritmética binaria
- Circuitos sumadores básicos
- Sumador de magnitudes
- Números binarios con signo
- Sumador con signo. Desbordamiento
- Sumador/restador
- **ALU**

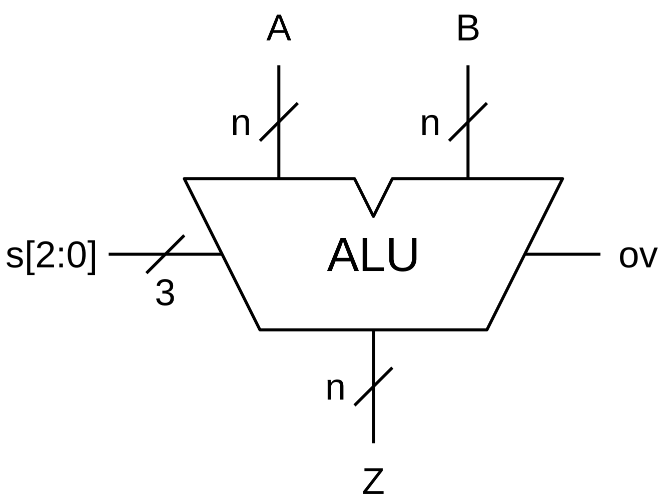
ALU

- Conjunto de operaciones de procesado de datos agrupadas en un mismo dispositivo
  - Operaciones lógicas
  - Operaciones aritméticas
- Uno de los componentes más importantes del computador



ALU de ejemplo

- Unidad Lógico-Aritmética en complemento a 2
  - $A = RC2_n(a)$, $B = RC2_n(b)$, $Z = RC2_n(z)$
  - ov: salida de desbordamiento (z no representable en C2)

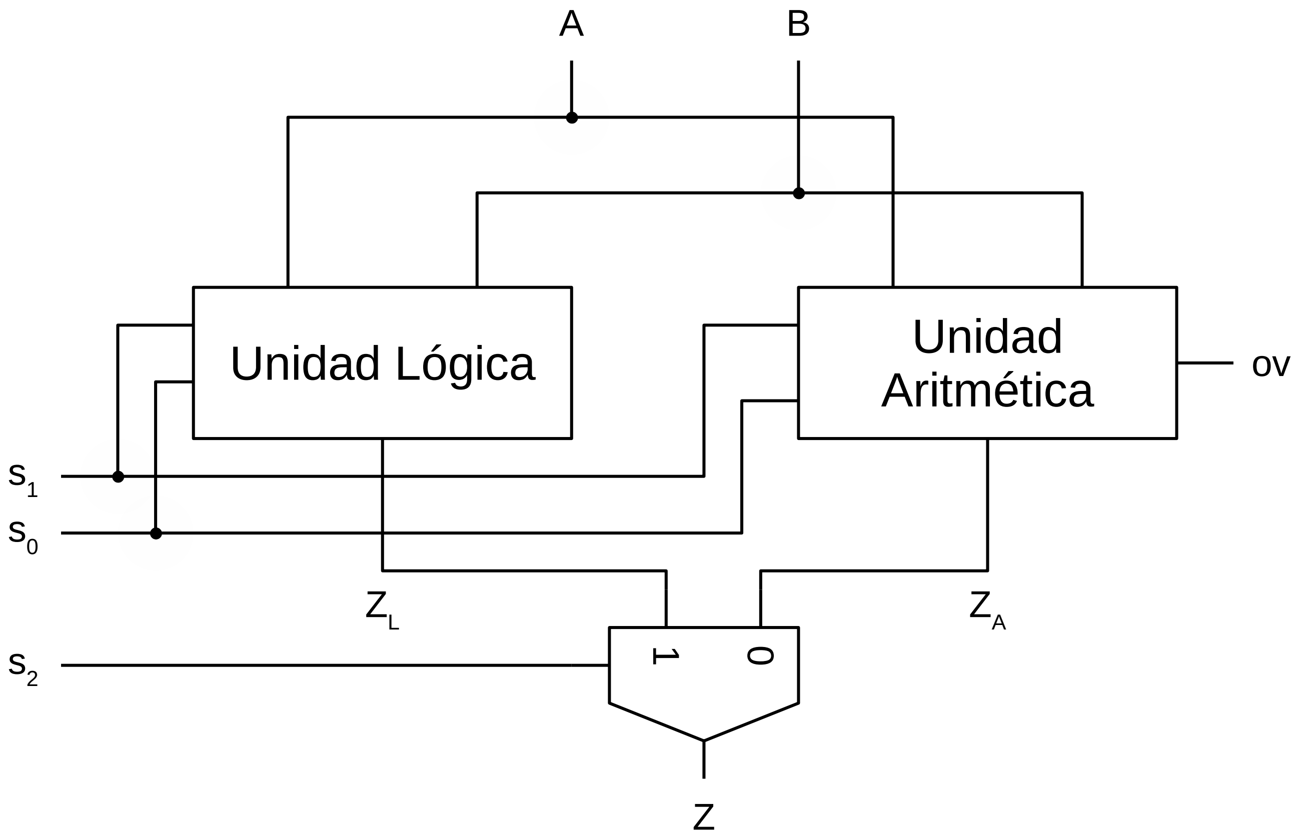


$s_2s_1s_0$	z	Z	
000	a + b	A + B	Aritméticas (s_2=0)
001	a - b	A+C2(B)	
010	a + 1	A + 1	
011	a - 1	$A+2^n-1$	
100	A AND B		Lógicas (s_2=1)
101	A OR B		
110	A XOR B		
111	NOT A		

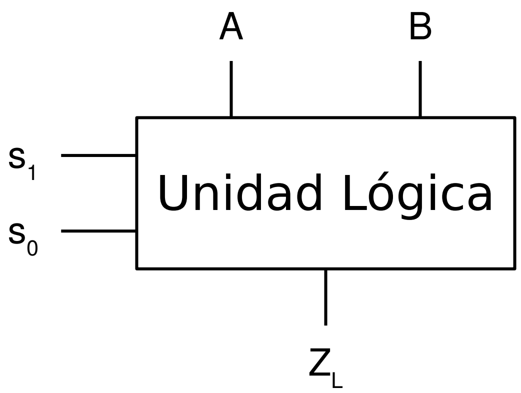
ALU
Estrategia de diseño

- ¡Divide y vencerás! (otra vez)
  - Diseñar una unidad lógica y una unidad aritmética independientes controladas por s_2 (¿multiplexor?)
- Unidad lógica
  - Seleccionar la operación adecuada con s_1 y s_0 (¿multiplexor?)
- Unidad aritmética
  - Usar un sumador de magnitudes como base.
  - Calcular las entradas del sumador (B y Cin) para obtener el resultado deseado.
  - Seleccionar los valores apropiados de B y Cin con s_1 y s_0 (¿multiplexor?)

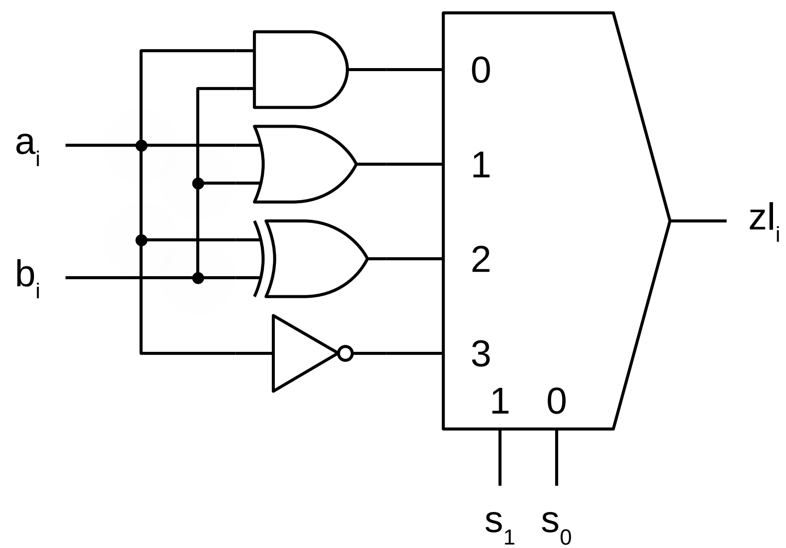
ALU
Diseño



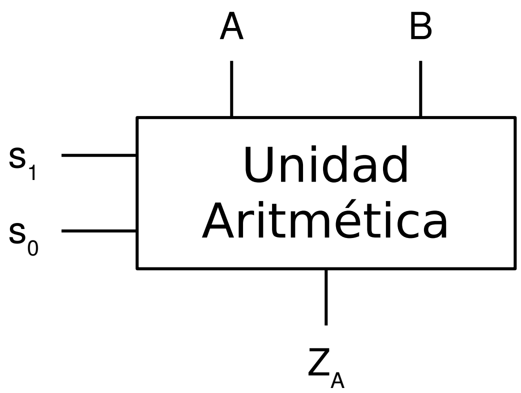
ALU
Diseño de la unidad lógica

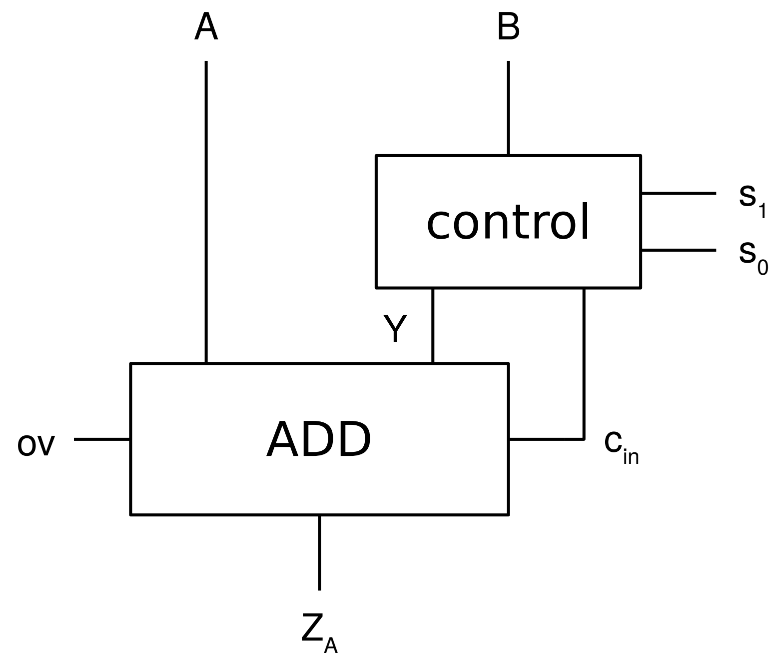


s_1s_0	zl	zl_i
00	A AND B	a_i AND b_i
01	A OR $\overline{B}$	a_i OR $\overline{b}_i$
10	A XOR B	a_i XOR b_i
11	NOT A	NOT a_i



ALU
Diseño de la unidad aritmética



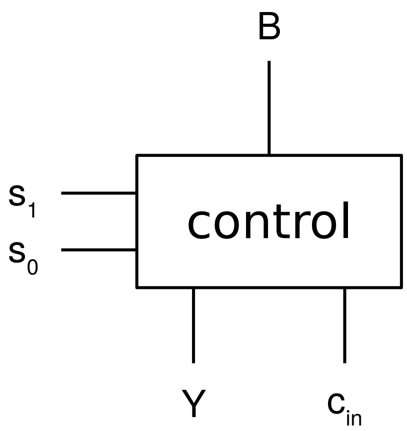


s_1s_0	z_A	Z_A
00	a + b	A + B
01	a – b	$A + \overline{B} + 1$
10	a + 1	A + 1
11	a – 1	$A + 2^n-1$

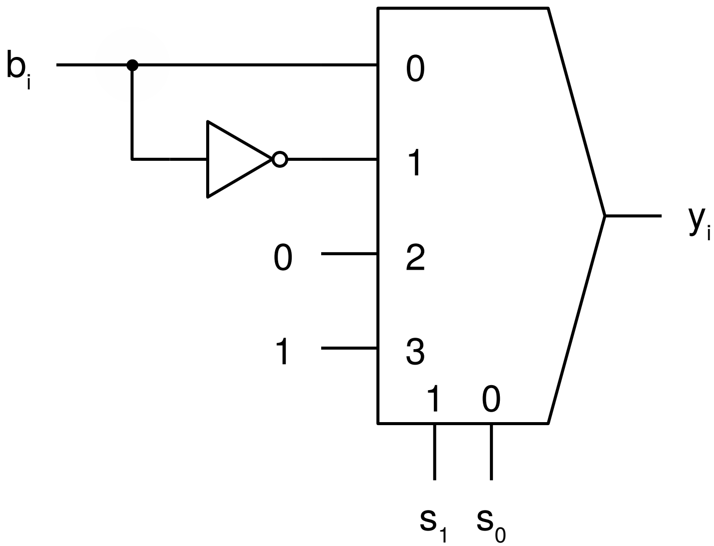
$$Z_A=A+Y+c_{in}$$

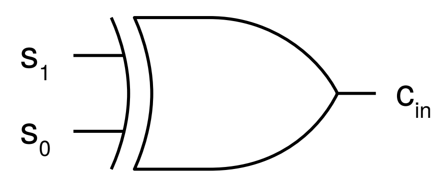
s_1s_0	Y	y_i	c_{in}
00	B	b_i	0
01	$\overline{B}$	$\overline{b}_i$	1
10	0	0	1
11	2^n-1	1	0

ALU
Diseño de la unidad aritmética

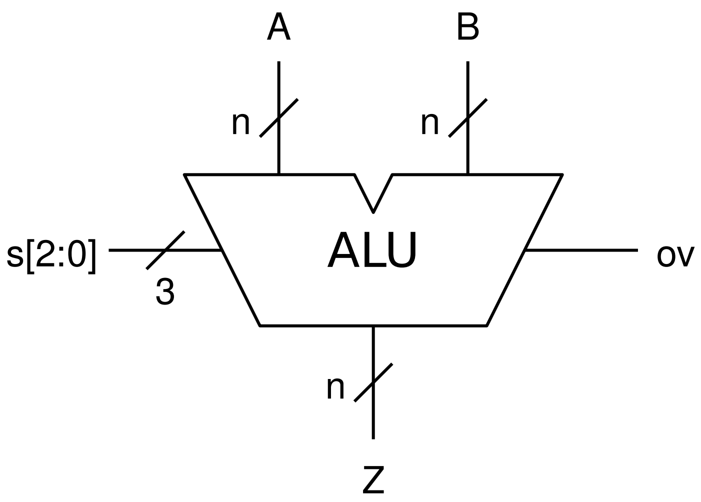


s_1s_0	Y	y_i	c_{in}
00	B	b_i	0
01	$\overline{B}$	$\overline{b}_i$	1
10	0	0	1
11	2^n-1	1	0





ALU
Descripción Verilog




```
module alu #(parameter N = 8)(
  input signed [N-1:0] a,
  input signed [N-1:0] b,
  input [2:0] s,
  output reg signed [N-1:0] z,
  output reg ov
  );

  reg signed [N:0] f;
```

```
  always @* begin
    ov = 0;

    if (s[2] == 0) begin  // Arithmetic
      case (s[1:0])
        2'b00:  f = a + b;
        2'b01:  f = a - b;
        2'b10:  f = a + 1;
        2'b11:  f = a - 1;
      endcase

      ov = (f[N] == f[N-1])? 0: 1;

      z = f[N-1:0];

    end else              // Logic
      case (s[1:0])
        2'b00:  z = a & b;
        2'b01:  z = a | b;
        2'b10:  z = a ^ b;
        2'b11:  z = ~a;
      endcase
  end // always

endmodule // alu
```