Fundamentos de Computadores

1º curso del Grado en Ingeniería Informática

Tema 5

Aritmética binaria

Universidad de Huelva

ETSI ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Departamento de Ingeniería Electrónica, de Sistemas Informáticos y Automática

Representación de números con signo

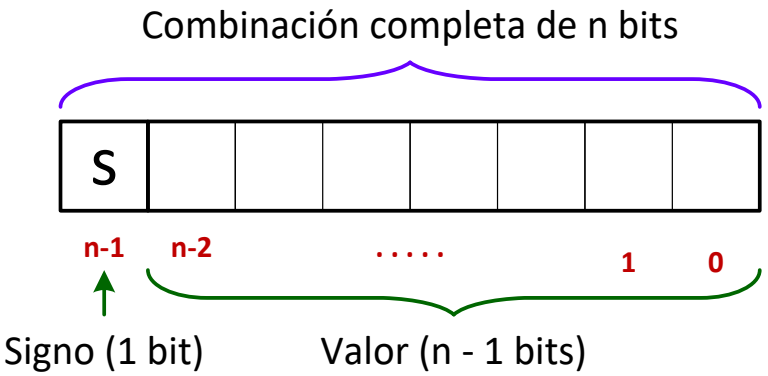
En un computador se efectúan múltiples operaciones aritméticas. En este tema se tratará la forma de realizar algunas de ellas, tales como la suma y la resta, así como los dispositivos y circuitos que las llevan a cabo. Las unidades lógicas y aritméticas (ALU), como parte integrante de cualquier procesador, también se estudian aquí.

Hasta el momento, se han estudiado una serie de códigos que únicamente permiten representar cantidades positivas. No obstante, en las operaciones realizadas en los computadores tanto los operandos como los resultados pueden tener signo positivo o negativo. Así pues se hace necesario el estudio de nuevos sistemas de representación de números que incluyan el signo de las cantidades.

A continuación se tratan algunos formatos de representación de números con signo:

- **Signo-magnitud.**
- **Complemento a 1.**
- **Complemento a 2.**

En los tres formatos a estudiar, para representar las cantidades se utilizan combinaciones de **n** bits, de los cuales el bit de mayor peso se usa para indicar el signo (S) y los **n - 1** bits restantes para expresar el valor.



Representación de números con signo (Signo magnitud)

En el formato de **signo-magnitud**, la representación de números con signo mediante combinaciones de **n** bits posee el siguiente rango:

$$[-2^{n-1} + 1, +2^{n-1} - 1]$$

- **Si se usan combinaciones de 4 bits (n = 4) el rango de representación será : [-7, +7].**
- **El rango de representación es simétrico y posee dos combinaciones para la representación del valor 0 (000…00 y 100…00).**
- **Para invertir el signo de una combinación basta con cambiar el valor del bit de signo de 0 a 1, o viceversa.**

Las cantidades se representan de la siguiente forma.

- **Bit de signo:**
  - **Números positivos:** S = 0.
  - **Números negativos:** S = 1.
- **n - 1 bits restantes:**
  - **Números positivos:** Se codifica el valor en binario natural.
  - **Números negativos:** Se codifica el valor en binario natural.

Ejemplos de representación de cantidades para n = 8.

- **00100101:** + 37.
- **01110010:** + 114.
- **01000100:** + 68.
- **00000000:** + 0.
- **10000000:** - 0.
- **11111000:** - 120.
- **10100001:** - 33.

D	C	B	A	VALOR DECIMAL
0	0	0	0	+ 0
0	0	0	1	+ 1
0	0	1	0	+ 2
0	0	1	1	+ 3
0	1	0	0	+ 4
0	1	0	1	+ 5
0	1	1	0	+ 6
0	1	1	1	+ 7
1	0	0	0	- 0
1	0	0	1	- 1
1	0	1	0	- 2
1	0	1	1	- 3
1	1	0	0	- 4
1	1	0	1	- 5
1	1	1	0	- 6
1	1	1	1	- 7

Representación de números con signo (Complemento a 1)

En el formato de **complemento a 1**, la representación de números con signo mediante combinaciones de n bits posee el siguiente rango:

$$[-2^{n-1} + 1, +2^{n-1} - 1]$$

- **Si se usan combinaciones de 4 bits (n = 4) el rango de representación será : [-7, +7].**
- **El rango de representación es simétrico y posee dos combinaciones para la representación del valor 0 (000…00 y 111…11).**
- **Para invertir el signo de una combinación se calcula su complemento a 1, es decir, se cambian ceros por unos, y viceversa.**

Las cantidades se representan de la siguiente forma.

- **Bit de signo:**
  - **Números positivos:** S = 0.
  - **Números negativos:** S = 1.
- **n - 1 bits restantes:**
  - **Números positivos:** Se codifica el valor en binario natural.
  - **Números negativos:** Se codifica el valor en complemento a 1.

Ejemplos de representación de cantidades para n = 8.

- **00100101:** + 37.
- **01110010:** + 114.
- **01000100:** + 68.
- **00000000:** + 0.
- **11111111:** - 0.
- **10000111:** - 120.
- **11011110:** - 33.

D	C	B	A	VALOR DECIMAL
0	0	0	0	+ 0
0	0	0	1	+ 1
0	0	1	0	+ 2
0	0	1	1	+ 3
0	1	0	0	+ 4
0	1	0	1	+ 5
0	1	1	0	+ 6
0	1	1	1	+ 7
1	0	0	0	- 7
1	0	0	1	- 6
1	0	1	0	- 5
1	0	1	1	- 4
1	1	0	0	- 3
1	1	0	1	- 2
1	1	1	0	- 1
1	1	1	1	- 0

Representación de números con signo (Complemento a 2)

En el formato de **complemento a 2**, la representación de números con signo mediante combinaciones de n bits posee el siguiente rango:

$$[-2^{n-1}, +2^{n-1} - 1]$$

- **Si se usan combinaciones de 4 bits (n = 4) el rango de representación será : [-8, +7].**
- **El rango de representación no es simétrico y posee una sola combinación para la representación del valor 0 (000…00.**
- **Para invertir el signo de una combinación se calcula su complemento a 2, del modo que se describirá posteriormente.**

Las cantidades se representan de la siguiente forma.

- **Bit de signo:**
  - **Números positivos:** S = 0.
  - **Números negativos:** S = 1.
- **n - 1 bits restantes:**
  - **Números positivos:** Se codifica el valor en binario natural.
  - **Números negativos:** Se codifica el valor en complemento a 2.

Ejemplos de representación de cantidades para n = 8.

- **00100101:** + 37.
- **01110010:** + 114.
- **01000100:** + 68.
- **00000000:** + 0.
- **11111111:** - 1.
- **10001000:** - 120.
- **11011111 :** - 33.

D	C	B	A	VALOR DECIMAL
0	0	0	0	+ 0
0	0	0	1	+ 1
0	0	1	0	+ 2
0	0	1	1	+ 3
0	1	0	0	+ 4
0	1	0	1	+ 5
0	1	1	0	+ 6
0	1	1	1	+ 7
1	0	0	0	- 8
1	0	0	1	- 7
1	0	1	0	- 6
1	0	1	1	- 5
1	1	0	0	- 4
1	1	0	1	- 3
1	1	1	0	- 2
1	1	1	1	- 1

Formas de calcular el complemento a 2 de una combinación

- **Restar el valor expresado en complemento a 2 de 2^n-1:**

  Ejemplo: Dado el valor **+ 2**, que codificado en complemento a 2 con cuatro bits es **0010**, restamos esta combinación de $2^4 = 16$, que expresado en binario es **10000**.

$$\begin{array}{r} 10000 \\ -\ 0010 \\ \hline 1110 \end{array}$$

  Como resultado se obtiene la combinación **1110**, que es la codificación del valor **- 2** en formato de complemento a 2 con 4 bits.

- **Calcular el complemento a 1 de la combinación y a continuación sumarle 1:**

  Ejemplo: Dado el valor **+ 2**, que codificado según el formato del complemento a 2 con cuatro bits es **0010**, comenzamos por obtener su complemento a 1, que es **1101**. A continuación sumamos 1 a la combinación obtenida.

$$\begin{array}{r} 1101 \\ +\quad 1 \\ \hline 1110 \end{array}$$

  Como resultado se obtiene la combinación **1110**, que es la codificación del valor **- 2** en formato de complemento a 2 con 4 bits. Este es el método que se emplea para realizar la resta en los circuitos digitales, como se verá posteriormente en este tema.

- **Mantener intactos todos los 0 situados a la derecha de la combinación y el primer 1, e invertir el resto de los bits:**

  Ejemplo: Dado el valor **+ 2**, que codificado según el formato del complemento a 2 con cuatro bits es **0010**, se opera de la forma siguiente.

$$0010 \rightarrow 1110$$

  Como se puede apreciar, este método proporciona el mismo resultado que los anteriores.

La **suma binaria** es una operación muy parecida a la suma decimal que conocemos, salvo que en ésta las combinaciones poseen sólo dos dígitos, el 0 y el 1.

La suma de dos combinaciones de n bits puede producir un resultado de n+1 bits, debido al acarreo que puede producirse en el bit de mayor peso (MSB).

Ejemplo: Suma de los valores 11 y 9 expresados en binario.

$$\begin{array}{r} 1011 \\ +\ 1001 \\ \hline 10100 \end{array}$$

Como se puede apreciar, la suma de números binarios de 4 bits puede dar como resultado un número de 5 bits.

Para llevar a cabo la suma de dos combinaciones **A** y **B**, es necesario comenzar por los bits de menos peso e ir sumado en cada etapa los bits correspondientes a ambas combinaciones (A_n y B_n) con el acarreo de la etapa anterior (C_{n-1}), lo cual dará como resultado el resultado de la suma correspondiente a dicha etapa (S_n) mas el acarreo a sumar a la etapa siguiente (C_n).

Sumador completo de 1 bit

Un **sumador completo** de un bit es un circuito combinacional capaz de realizar la suma de dos bits más un acarreo de entrada para producir un bit de suma mas un acarreo de salida.

A	B	CI	CO	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

$$S = \bar{A}\bar{B}C_I + \bar{A}B\bar{C}_I + A\bar{B}\bar{C}_I + ABC_I = \bar{A}\,(\bar{B}C_I + B\bar{C}_I) + A\,(\overline{\bar{B}C_I + BC_I}) =$$

$$= \bar{A}\,(B \oplus C_I) + A\,(\overline{B \oplus C_I}) = A \oplus B \oplus C$$

$$C_O = \bar{A}BC_I + A\bar{B}C_I + AB\bar{C}_I + ABC_I =$$

$$= AB\,(\bar{C}_I + C_I) + AC_I\,(\bar{B} + B) + BC_I\,(\bar{A} + A) = AB + (A + B)\,C_I$$

Las expresiones obtenidas anteriormente permite implementar el sumador completo como sigue.

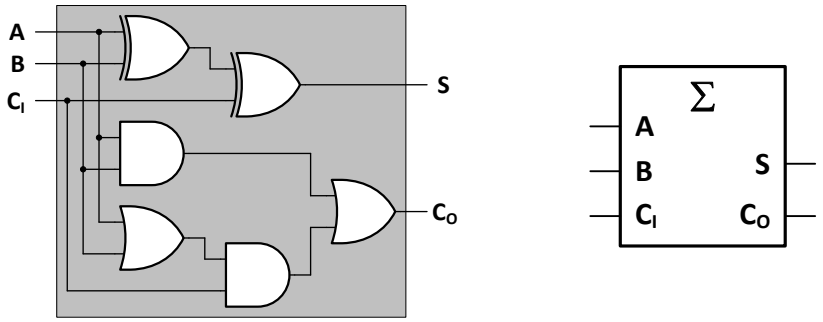
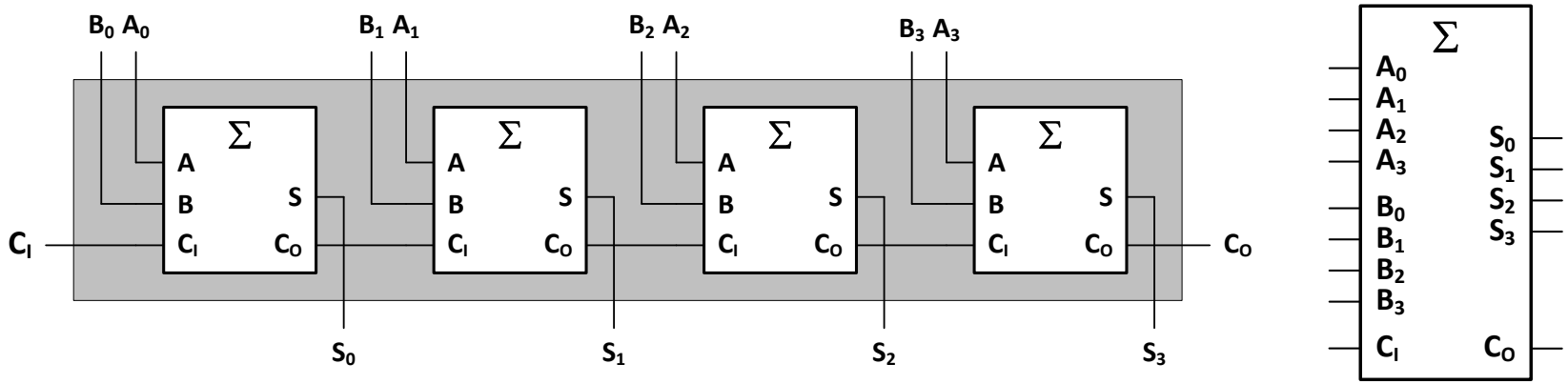


Diagrama lógico y símbolo de un sumador completo de 1 bit

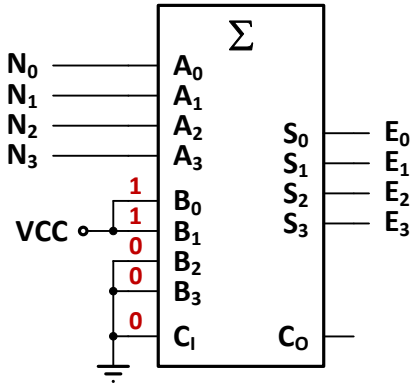
Sumador paralelo de 4 bits

Mediante la conexión de cuatro sumadores completos de 1 bit con propagación del acarreo en serie se puede obtener un sumador paralelo de cuatro bits, que permite realizar la suma entre dos combinaciones de 4 bits mas un acarreo de entrada.



Ejemplos de usos de los sumadores

Ejemplo 1: Conversión de BCD Natural a BCD exceso 3.

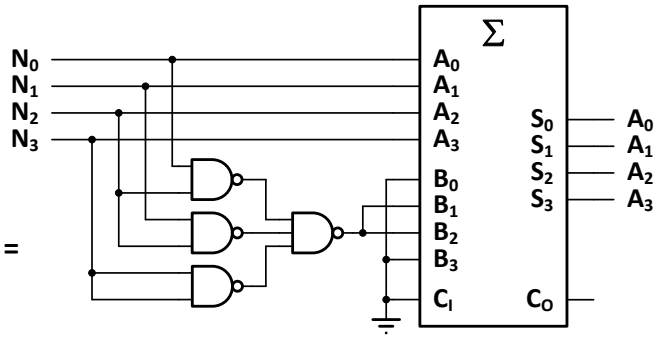


Ejemplo 2: Conversión de BCD Natural a BCD Aiken.

	BCD NATURAL				CORRECCIÓN				BCD AIKEN			
Valor	N_3	N_2	N_1	N_0	C_3	C_2	C_1	C_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	1	0
3	0	0	1	1	0	0	0	0	0	0	1	1
4	0	1	0	0	0	0	0	0	0	1	0	0
5	0	1	0	1	0	1	1	0	1	0	1	1
6	0	1	1	0	0	1	1	0	1	1	0	0
7	0	1	1	1	0	1	1	0	1	1	0	1
8	1	0	0	0	0	1	1	0	1	1	1	0
9	1	0	0	1	0	1	1	0	1	1	1	1

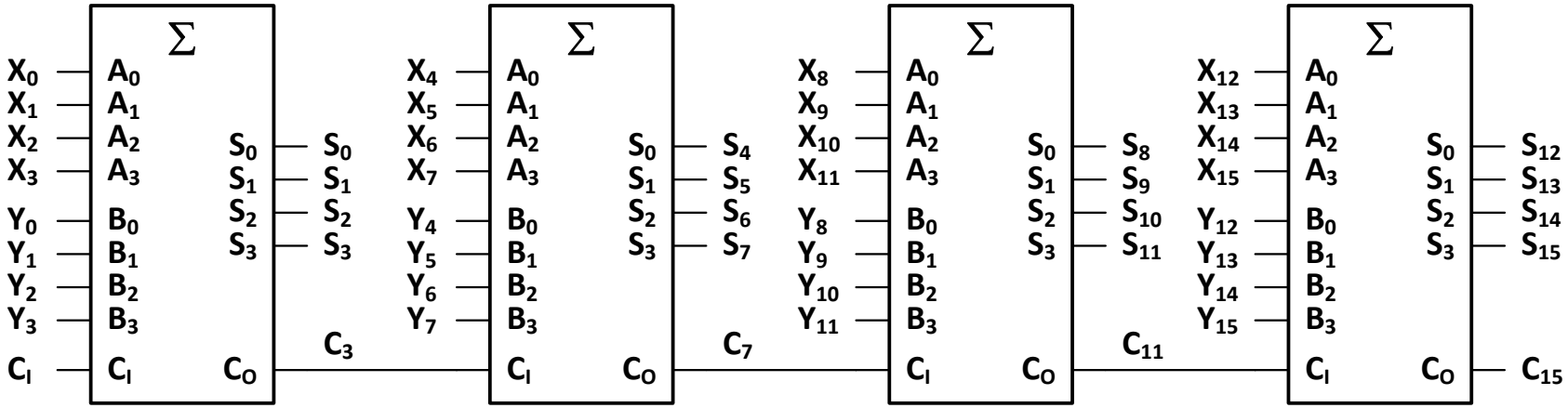
$$C_3 = C_0 = 0$$

$$C_2 = C_1 = N_3 + N_2N_1 + N_2N_0 =$$
$$= \overline{\overline{N_3} \cdot \overline{N_2N_1} \cdot \overline{N_2N_0}}$$



Mediante la conexión de sumadores paralelos de 4 bits con propagación del acarreo en serie se puede obtener un sumador paralelo para combinaciones de tantos bits como sea necesario. El inconveniente de este tipo de conexión consiste en que cuantos más circuitos se añadan a la cadena mayor será el retardo del circuito obtenido.

Ejemplo: Obtener un sumador paralelo para combinaciones de 16 bits mediante el uso de sumadores paralelos de 4 bits.



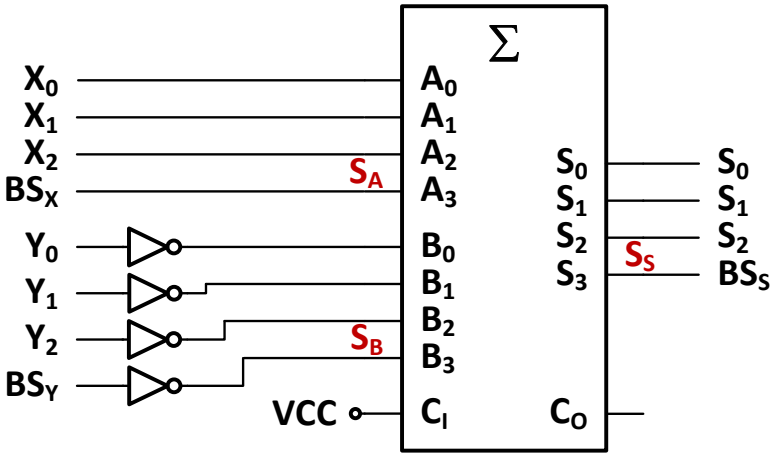
En los sistemas digitales, la **resta binaria** se suele llevar a cabo en la práctica mediante circuitos implementados con sumadores.

Para ello, la resta binaria se trata como una suma algebraica, de la siguiente manera:

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

Si se representan los números en formato de complemento a 2 (n-1 bits más signo), como se indicó anteriormente, para cambiarlos de signo bastará con hacerles el complemento a 1 (invertir todos sus bits) y sumarle 1 al resultado.

Así pues, para implementar un restador de cuatro bits se aplicará el minuendo directamente a uno de los operandos de un bloque sumador y el sustraendo al otro operando a través de cuatro inversores (que realizan el complemento a 1), mientras que al acarreo de entrada se le aplica el valor 1 (para sumar 1 al resultado).



Al realizar una resta binaria hay que asegurarse de que el resultado no sobrepase la capacidad de representación de la combinación de salida $\Rightarrow [-2^{n-1}, +2^{n-1} - 1]$. Si el valor obtenido queda fuera de este rango se produce un desbordamiento (*overflow*), y por tanto no será válido.

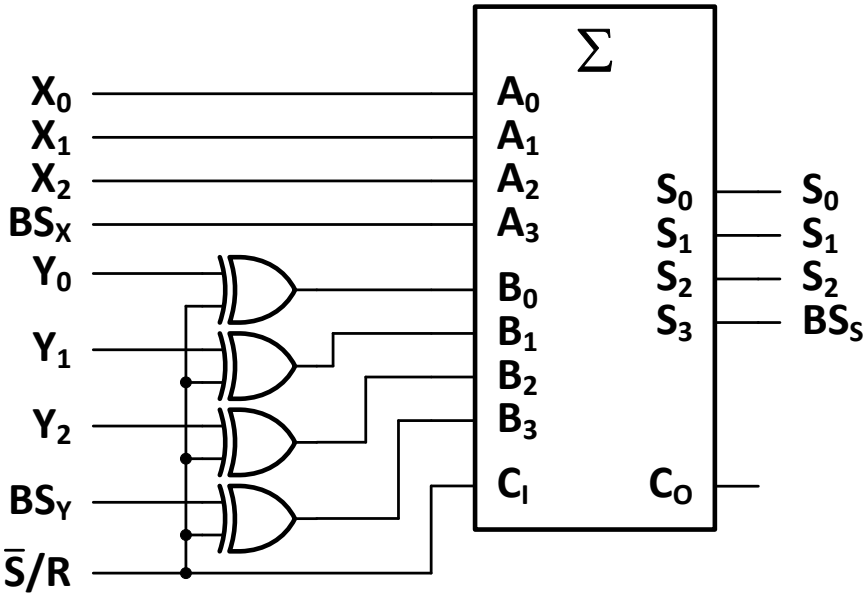
Este desbordamiento puede ser detectado fácilmente mediante la siguiente función:

$$D = \overline{S_A}\ \overline{S_B}\ S_S + S_A\ S_B\ \overline{S_S}$$

Resta binaria (Circuito sumador/restador en complemento a 2)

Con frecuencia, en un sistema digital deben realizarse tanto sumas como restas, por lo que se hace necesario disponer de un circuito capaz de realizar ambas operaciones.

En la figura se muestra el modo de implementar un circuito sumador/restador capaz de operar sobre dos combinaciones de tres bits mas signo.



Cuando se aplica a la señal $\bar{S}/R$ el valor **0**, al operando B del sumador llega el número Y sin complementar y el acarreo de entrada también adopta el valor **0**, por lo que el circuito realiza la suma de los dos números. Por el contrario, si es puesto a **1**, se aplica al operando B del sumador el complemento a uno del número Y, y un **1** al acarreo de entrada, por lo que el circuito realiza la resta X - Y.

Operaciones aritméticas en VHDL (Suma y resta)

Para realizar la **suma** de dos números en VHDL:

```
S <= A + B;
```

- **Los números A y B pueden ser vectores y enteros.** Los operandos **A** y **B** pueden ser de distinta longitud. Si **S** se declara de la misma longitud que el operando de mayor longitud, el acarreo no se tiene en cuenta. Si se quiere conservar el acarreo, **S** debe tener un bit más que el operando de mayor longitud, y en ese caso la suma se escribirá:

```
S <= '0' & A + B;
```

- **Si operamos con números positivos y negativos,** deben declararse **S**, **A** y **B** de la misma longitud, y la operación **R** <= **A** + **B** usará el complemento a 2 para representar números negativos. Se puede añadir un circuito que detecte el *overflow* comprobando los bits de signo.
- Cuando A o B son de menor longitud que S, el paquete ***std_logic_unsigned*** añade ceros hasta completar la longitud de la suma. Si se emplea el paquete ***std_logic_signed*,** la longitud se completa repitiendo el bit de signo.

Para realizar la **resta** de dos números en VHDL:

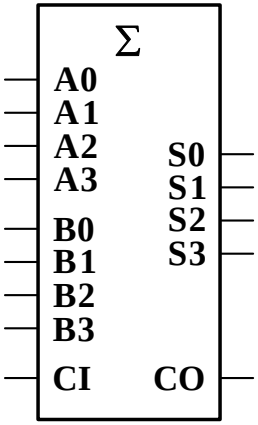
```
R <= A – B;
```

Los operandos pueden ser de distinta longitud, y el resultado R ha de ser de longitud igual o mayor que el operando de mayor longitud.

Dependiendo de si trabajamos con números positivos y negativos, o sólo positivos, emplearemos uno de estos paquetes:

```
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
```

Ejemplo: Realizar una descripción VHDL de un sumador completo de 4 bits.



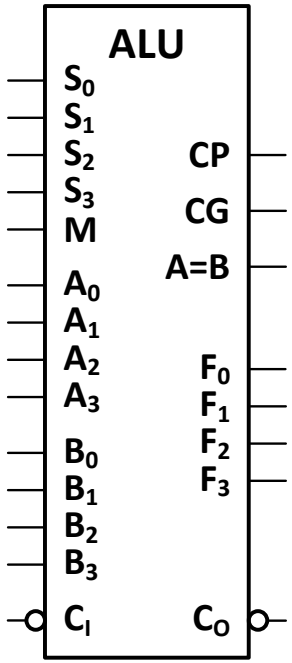

```
-- Sumador de 4 bits
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity sumador_4 is
      port (B, A : in std_logic_vector (3 downto 0);
            C0 : in std_logic;
            S : out std_logic_vector (3 downto 0);
            C4: out std_logic);
end sumador_4;

architecture suma of sumador_4 is
      signal sum : std_logic_vector (4 downto 0);
      begin
            sum <= ('0' & A) + ('0' & B) + ("0000" & C0);
            C4 <= sum(4);
            S <= sum(3 downto 0);
end suma;
```

Las **unidades aritmético-lógicas** o **ALUs** (*Arithmetic Logic Units*) son circuitos combinacionales que, como su nombre indica, son capaces de realizar un conjunto de operaciones tanto lógicas como aritméticas.

Estos circuitos reciben dos operandos A y B (normalmente de 4 bits) más un acarreo de entrada (C_I) y proporcionan un salida F de la misma longitud que los operandos y un acarreo de salida (C_O). Además, suelen incorporar salidas de generación (CG) y propagación (CP) de acarreos, que se usan para asociar varios circuitos mediante la generación de acarreos en paralelo, y una salida A = B (de colector abierto) que permite detectar si las combinaciones de entrada son iguales.



74181

SELECCIÓN				DATOS ACTIVOS A NIVEL ALTO (TABLA 2)		
				M = 1	M = 0; OPERACIONES ARITMÉTICAS	
S_3	S_2	S_1	S_0	FUNCIONES LÓGICAS	$\overline{C_n} = 1$ (Sin acarreo)	$\overline{C_n} = 0$ (Con acarreo)
0	0	0	0	F = A	F = A	F = A MAS 1
0	0	0	1	F = $\overline{A + B}$	F = A + B	F = (A + B) MAS 1
0	0	1	0	F = $\overline{A}B$	F = A + $\overline{B}$	F = (A + $\overline{B}$) MAS 1
0	0	1	1	F = 0	F = MENOS 1 (COMPL. 2)	F = 0
0	1	0	0	F = $\overline{AB}$	F = A MAS A$\overline{B}$	F = A MAS A$\overline{B}$ MAS 1
0	1	0	1	F = $\overline{B}$	F = (A + B) MAS A$\overline{B}$	F = (A + B) MAS A$\overline{B}$ MAS 1
0	1	1	0	F = A ? B	F = A MENOS B MENOS 1	F = A MENOS B
0	1	1	1	F = A$\overline{B}$	F = A$\overline{B}$ MENOS 1	F = A$\overline{B}$
1	0	0	0	F = $\overline{A}$ + B	F = A MAS AB	F = A MAS AB MAS 1
1	0	0	1	F = $\overline{A ? B}$	F = A MAS B	F = A MAS B MAS 1
1	0	1	0	F = B	F = (A + $\overline{B}$) MAS AB	F = (A + $\overline{B}$) MAS AB MAS 1
1	0	1	1	F = AB	F = AB MENOS 1	F = AB
1	1	0	0	F = 1	F = A MAS A	F = A MAS A MAS 1
1	1	0	1	F = A + $\overline{B}$	F = (A + B) MAS A	F = (A + B) MAS A MAS 1
1	1	1	0	F = A + B	F = (A + $\overline{B}$) MAS A	F = (A + $\overline{B}$) MAS A MAS 1
1	1	1	1	F = A	F = A MENOS 1	F = A