Sistemas de numeración y aritmética binaria

Sistemas de numeración posicionales

Un sistema de numeración se define como un conjunto de símbolos y reglas que permiten construir todos los números de dicho sistema.

Dentro de los sistemas de numeración se pueden distinguir los posicionales, en los cuales el valor del símbolo depende no solo del valor en sí mismo, sino también de la posición en la que se encuentra dentro del número. Un ejemplo es el *sistema de numeración decimal*, que es el que habitualmente utilizamos. Se dice que el sistema de numeración es decimal debido a que diez son la cantidad de símbolos que tiene el sistema, por lo que se dice que la base de ese sistema es 10. Esos símbolos son $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Habitualmente, a estos símbolos se los conoce como dígito.

Un ejemplo es el número 347. En este, el dígito que se encuentra en la posición más a la izquierda (en este caso, el 3) es el dígito con mayor peso dentro del número, mientras que el dígito que se encuentra más a la derecha (el 7) es el de menor peso.

En la siguiente secuencia se descompone el número elegido:

$$347 = 300 + 40 + 7$$

$$347 = 3 \times 100 + 4 \times 10 + 7 \times 1$$

$$347 = 3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

Indica la base

Indica la posición

(ambos indican el peso del dígito)

Base: cantidad de símbolos o dígitos que conformar dicho sistema de numeración.

Peso: valor del dígito en función a su posición.

El sistema de numeración decimal no es el único sistema posicional. Todos los sistemas de numeración posicional se estructuran de la misma manera que el sistema de numeración en base 10, cambiando la base y, por lo tanto, sus símbolos.

Los sistemas de numeración que se estudiarán son los utilizados en los sistemas digitales:

- *Sistema decimal*, es decir, de base 10 $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$.
- *Sistema binario*, base 2 $(0, 1)$.
- *Sistema octal*, base 8 $(0, 1, 2, 3, 4, 5, 6, 7)$.

- *Sistema hexadecimal*, base 16 $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)$.

Para indicar en qué sistema de numeración o base se está trabajando se agrega un número o una letra como subíndice. Para el decimal, 10 o la d (ejemplo: 25_{10} *o* 25_d), para el binario, 2 o la b, para octal, 8 u o, y para el hexadecimal 16 o H.

En el sistema binario, al dígito binario se lo conoce como *bit* (*binary digit*), a la agrupación de 4 bits se le llama *nibble,* mientras que a la de 8 bits se la conoce como *byte.*

Representación de números en diferentes bases

En la siguiente tabla se muestra la representación de los primeros números enteros positivos en las diferentes bases:

Decimal		**Binario**					**Octal**		**Hexadecimal**	
10^1	10^0	2^4	2^3	2^2	2^1	2^0	8^1	8^0	16^1	16^0
	0					0		0		0
	1					1		1		1
	2				1	0		2		2
	3				1	1		3		3
	4			1	0	0		4		4
	5			1	0	1		5		5
	6			1	1	0		6		6
	7			1	1	1		7		7
	8		1	0	0	0	1	0		8
	9		1	0	0	1	1	1		9
1	0		1	0	1	0	1	2		A
1	1		1	0	1	1	1	3		B
1	2		1	1	0	0	1	4		C
1	3		1	1	0	1	1	5		D
1	4		1	1	1	0	1	6		E
1	5		1	1	1	1	1	7		F
1	6	1	1	1	1	1	2	0	1	0

La segunda fila muestra el peso según la posición del dígito. Si se observa cuidadosamente, se puede ver que la combinación de los dígitos '10' en las diferentes bases corresponde en decimal al número de la base en la que está escrito el dicho número.

Del mismo modo que en el sistema de numeración decimal, una combinación de dígitos agregando más dígitos indica números más grandes. Sucede lo mismo en los sistemas de numeración binario, octal y hexadecimal.

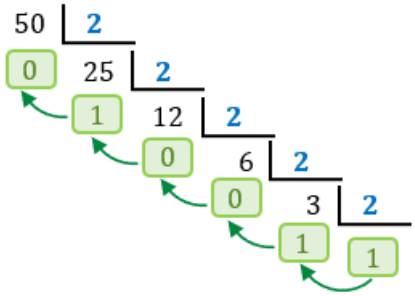
Conversión entre bases

En la tabla anterior se ve cómo escribir los primeros diecisiete números positivos en los diferentes sistemas de numeración. ¿Si se quiere escribir números más grandes en otras bases hay que seguir completando la tabla? No, no es necesario. Por eso, se verá cómo convertir un número a otras bases.

Del sistema decimal al binario

Se quiere pasar el número 50*, que está en decimal, a binario (base 2).*

Para esto, se debe tomar el número 50 y dividirlo por la base a la que se quiere llegar, en este caso 2, hasta llegar a que el resultado de la división entera es menor a la base. Al realizar las divisiones, estas deberán ser de resultados enteros y se deberá anotar los restos. El último resultado y los restos concatenados darán el equivalente del número decimal en binario.



$$50_d = 110010_b$$

Del sistema binario al decimal

Se quiere pasar el número 110010*, que está en binario, a decimal.*

En este caso, se debe escribir el número en forma polinomial. Es decir, se toma cada dígito del número y se lo multiplica por la base elevada a la posición en que se encuentra de derecha a izquierda, empezando en cero y aumentado su posición en uno. El resultado de dicha cuenta corresponderá al número en decimal.

$$110010_b = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$110010_b = 1 \times 32 + 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$$

$$110010_b = 32 + 16 + 0 + 0 + 2 + 0 = 50_d$$

$$\boxed{110010_b = 50_d}$$

Se verifica, de esta forma, que el resultado obtenido anteriormente es el correcto.

Del sistema decimal al binario con parte fraccionaria

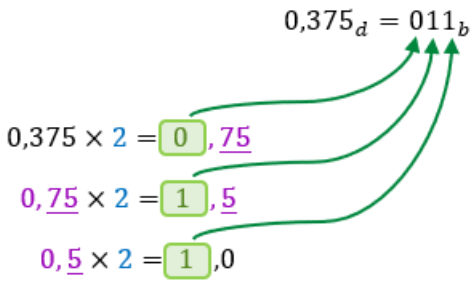
Se quiere pasar el siguiente número $50{,}375_{\mathrm{d}}$ *a binario.*

Para realizarlo, se separa el número en dos partes: por un lado, la parte entera (la que se encuentra a la izquierda de la coma) y, por el otro, la parte fraccionaria (a la derecha de la coma).

La conversión de parte entera (50_d) se realizó anteriormente y se obtuvo:

$$50_d = 110010_b$$

Para realizar la conversión de la parte fraccionaria $(0{,}375_d)$, se debe multiplicar dicho número por la base (en este caso 2), sucesivamente, hasta obtener la cantidad de dígitos fraccionarios que se quiera o hasta que el resultado sea un número entero. En cada multiplicación se deberá guardar la parte entera del resultado y solo utilizar para multiplicar la parte fraccionaria de este. Las partes enteras obtenidas en cada cuenta darán como resultado la conversión de la parte fraccionaria en binario.



Al unir los dos resultados obtenidos (parte entera y fraccionaria) el resultado final es:

$$50{,}375_d = 110010{,}011_b$$

Del sistema binario al decimal con parte fraccionaria

Se quiere pasar el número $110010{,}011_b$ *a decimal*

Para realizarlo, se puede separar en dos partes (entera y fraccionaria).

La parte entera se hizo anteriormente y se obtuvo como resultado: $110010_b = 50_d$

Para la parte fraccionaria, el procedimiento es el mismo, pero al estar del lado derecho de la coma la posición va disminuyendo en uno de izquierda a derecha, empezando en -1.

$0{,}011_b = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$

$0{,}011_b = 0 \times \frac{1}{2} + 1 \times \frac{1}{4} + 1 \times \frac{1}{8}$

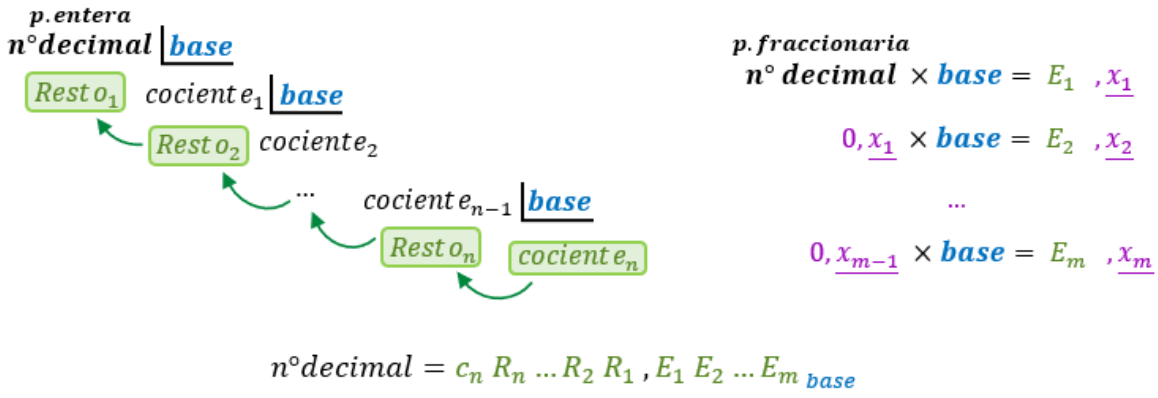
$0{,}011_b = 0 + 0{,}25 + 0{,}125 = 0{,}375_d$

Uniendo ambos resultados se obtiene como resultado final:

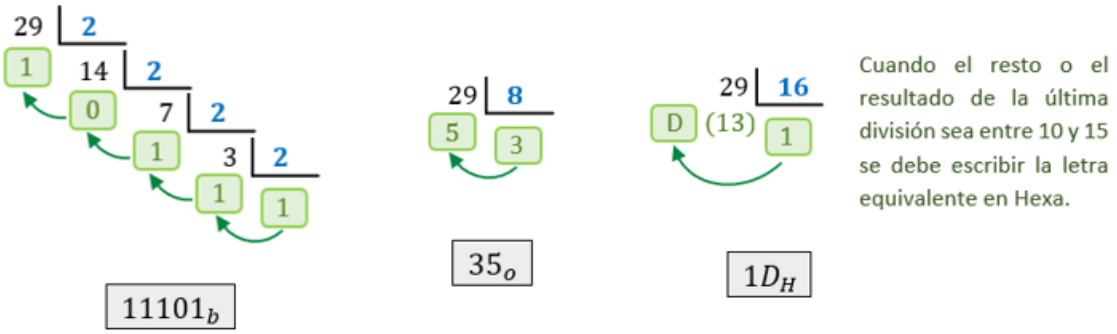
$$110010{,}011_b = 50{,}375_d$$

De base 10 a otra base

Del mismo modo que se pasó de un número decimal a binario, se puede pasar de un número decimal a hexadecimal u octal. Se deberá, en ese caso, cambiar el número 2 por el número 16 u 8 según corresponda. Generalizando, se obtiene lo siguiente:



Ejemplo: *pasar el número* 29_{d} *a binario, octal y hexadecimal.*



Decimal	Binario	Octal	Hexadecimal
29_d	11101_b	35_o	$1D_H$

De otra base a base 10

Del mismo modo que se pasó de un número binario a decimal, se puede pasar de un número hexadecimal u octal a decimal. Se deberá, en ese caso, cambiar el número 2 por el número 16 u 8 según corresponda. Generalizando, se obtiene lo siguiente:

$$número_{\ otra\ base} = x_n \dots x_1\, x_0\, , x_1 \dots x_{-m}$$

$$n°decimal = x_n \times base^{n} + \cdots + x_1 \times base^{1} + x_0 \times base^{0} + x_{-1} \times base^{-1} + \cdots + x_{-m} \times base^{-m}$$

Ejemplo: *verificar que los resultados obtenidos en el ejemplo anterior son* 29 *en decimal.*

$$11101_b = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 4 + 1 = 29_d$$

$$35_o = 3 \times 8^1 + 1 \times 8^0 = 3 \times 8 + 5 \times 1 = 24 + 5 = 29_d$$

$$1D_H = 1 \times 16^1 + 13(D) \times 16^0 = 1 \times 16 + 13 \times 1 = 16 + 13 = 29_d$$

Decimal	Binario	Octal	Hexadecimal
29_d	11101_b	35_o	$1D_H$

Obsérvese que habiendo partido de un número en la base 10, si se quiere saber su equivalente en una base menor (2 u 8) se espera que los dígitos sean mayores o se utilicen más dígitos; por el contrario, si se quiere su equivalente en una base mayor (16) se espera que los dígitos utilizados sean más chicos o menor cantidad.

¿Es posible pasar de una base directamente a otra sin tener que pasar por la base 10?

Sí, siempre que se cumpla que una base se pueda escribir como potencia de otra, por ejemplo, $2^3 = 8$, entonces se podrá pasar del sistema binario al octal directamente, y viceversa. Lo mismo sucede con $2^4 = 16$, por lo que se puede pasar directamente de binario a hexadecimal, y viceversa.

¿Y de octal a hexadecimal?

En ese caso, se deberá optar por pasar primero a binario o a decimal y luego a la otra base.

Del sistema binario al octal

Se quiere pasar el número $110010{,}011_{\mathrm{b}}$ *a octal.*

Para esto, se debe tener en cuenta que $2^{\mathbf{3}} = 8$, por lo que cada **3** *bits* se debe escribir el equivalente en octal. La agrupación de los **3** bits se realiza partiendo de la coma, para un lado y para el otro. En caso de que falte algún *bit,* se agregarán los ceros necesarios a la izquierda, si es antes de la coma, o a la derecha, si es después de esta.

Binario: $110 \quad 010 \quad , \quad 011$

Octal: $6 \quad 2 \quad , \quad 3$

$$110010{,}011_b = 62{,}3_o$$

Del sistema octal al binario

Se quiere pasar el número $62{,}3_{\mathrm{o}}$ *a binario.*

Para esto, debemos tener en cuenta que $2^{\mathbf{3}} = 8$, por lo cual cada dígito se deberá escribir como su equivalente en binario con **3** *bits.*

Octal: $6 \quad 2 \quad , \quad 3$

Binario: $110 \quad 010 \quad , \quad 011$

$$62{,}3_o = 110010{,}011_b$$

Del sistema binario al hexadecimal

Se quiere pasar el número $110010{,}011_{\mathrm{b}}$ *a hexadecimal.*

Para esto, se debe tener en cuenta que $2^{\mathbf{4}} = 16$, por lo que cada **4** *bits* se debe escribir el equivalente en hexadecimal. La agrupación de los **4** *bits* se realiza partiendo de la coma, para un lado y para el otro. En caso de que falte algún *bit,* se agregarán los ceros necesarios a la izquierda, si es antes de la coma, o a la derecha, si es después de esta.

Binario:	0011	0010	,	0110
Hexadecimal:	3	2	,	6

$$110010{,}011_b = 32{,}6_H$$

Del sistema hexadecimal al binario

Se quiere pasar el número $32{,}6_{\mathrm{H}}$ *a binario.*

Para esto, debemos tener en cuenta que $2^{\mathbf{4}} = 16$, por lo cual cada dígito se deberá escribir como su equivalente en binario con **4** *bits.*

Hexadecimal:	3	2	,	6
Binario:	0011	0010	,	0110

$$32{,}6_H = 00110010{,}0110_b = 110010{,}011_b$$

Operaciones aritméticas en sistema de numeración binario

Así como en el sistema de numeración decimal podemos realizar sumas aritméticas, también es posible hacerlo en el sistema de numeración binario, tal como lo hacen las computadoras.

Suma

Reglas de suma en binario:

$0 + 0 = 0$

$1 + 0 = 1$

$1 + 1 = 10$ (el 1 corresponde al *carry* o acarreo, “me llevo” 1 a la columna izquierda).

Se quiere realizar la siguiente operación $13_{\mathrm{d}} + 7_{\mathrm{d}}$ *en binario.*

Es conveniente recodar cómo se opera en el sistema de numeración decimal. De la misma forma, se puede hacer en binario. Primero, se deben pasar los números que están en decimal al binario (como vimos antes). Luego, se suman los *bits* columna a columna. En caso de que aparezca un *carry,* se debe llevar dicho *bit* a la columna izquierda inmediata.

Si se quiere verificar que el resultado obtenido de la cuenta es el correcto, se debe pasar dicho resultado a decimal.

$13_d = 1101_b$

$7_d = 111_b$

Decimal	**Binario**
$\begin{array}{rr} & {\scriptstyle 1} \\ & 1\ 3 \\ + & \ \ 7 \\ \hline & 2\ 0 \end{array}$	$\begin{array}{rrrrr} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 1 & 1 & 0 & 1 \\ + & & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0\ 0 \end{array}$

Verificación

$$\begin{array}{|ccccc|} \hline 1 & 0 & 1 & 0 & 0 \\ \hline 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 16 & 8 & 4 & 2 & 1 \end{array} = 20_d$$

16+4= 20

Resta

Reglas de resta en binario:

$0 - 0 = 0$

$1 - 0 = 1$

$10 - 1 = 1$

En el caso de que aparezca el “le pido prestado” a la columna izquierda, este estaría haciendo referencia al *borrow.*

Se quiere realizar la siguiente operación $37_d - 10_d$ *en binario.*

Primero, se deben pasar los números decimales al binario (como vimos antes). Luego, se debe restar columna a columna los *bits.* En caso de que no alcance, se buscará en la columna izquierda para “pedir prestado” (*borrow*).

Si se quiere verificar que el resultado obtenido es el correcto, se debe pasar dicho resultado a decimal.

$37_d = 100101_b$

$10_d = 1010_b$

Decimal	**Binario**
3 7	0 1 0 1 10 10 1 10 1
− 1 0	− 1 0 1 0
2 7	0 1 1 0 1 1

Verificación

1	1	0	1	1
2^4	2^3	2^2	2^1	2^0
16	8	4	2	1

$= 27_d$

16+8+2+1= 27

Se sabe que $37_d - 10_d = 27_d$ es lo mismo que $+37_d + (-10)_d = 27_d$. La diferencia entre la primera expresión y la segunda es que la segunda utiliza números con signo.

¿Es posible escribir números con signo en el sistema de numeración binario? Sí, se verán tres formas de escribir los números binarios con signo.

Números binarios con signo

Se verá cómo escribir el número $+10_d$ y -10_d en binario con signo en las diferentes formas de representación.

a) Signo y magnitud (SyM)

Cuando se quiere escribir un número en binario con signo, primero se debe pasar el número a binario sin signo y luego se deberá agregar un *bit* adelante, que corresponde al *bit* de signo. Si se quiere indicar que es positivo, el *bit* de signo debe ser '0', mientras que si se quiere indicar que es negativo el *bit* de signo debe ser '1'. En el caso de la representación de SyM, bastará con agregar el *bit* de signo.

$$+10 = \underset{\text{Signo}}{0}\ \underset{\text{Magnitud}}{1010}$$

$$-10 = \underset{\text{Signo}}{1}\ \underset{\text{Magnitud}}{1010}$$

b) Complemento a 1 (Ca1)

El complemento a 1 hace referencia al complemento a la base (2) menos 1, es decir, lo que le falta al número (N) para llegar a la potencia de la base de n cantidad de *bits* menos 1.

$$C_1(N) = 2^n - N - 1$$

N: número que se quiere complementar, n: cantidad de bits que tiene el número N.

Un número binario positivo en Ca1 se escribe igual en SyM, es decir, solo habrá que agregarle el *bit* de signo $'0'$ adelante. En caso de querer escribir un número negativo, habrá que agregarle el bit de signo '1' y, en la parte correspondiente a la magnitud, se deberán cambiar todos los unos por ceros y todos los ceros por unos.

$$+10 = \underset{\text{Signo}}{0}\ 1010$$

$$-10 = \underset{\text{Signo}}{1}\ 0101$$

c) Complemento a 2 (Ca2)

El complemento a 2 hace referencia al complemento a la base (2), es decir lo que le falta al número (N) para llegar a la potencia de la base de n cantidad de *bits.*

$$C_2(N) = 2^n - N$$

N: número que se quiere complementar, n: cantidad de bits que tiene el número N.

Un número binario positivo en Ca2 se escribe igual en SyM y que en Ca1, es decir, solo habrá que agregarle el *bit* de signo adelante $'0'$. En caso de querer escribir un número negativo, habrá que agregarle el *bit* de signo '1' y, en la parte correspondiente a la magnitud, se deberán cambiar todos los unos por ceros y todos los ceros por unos y luego sumarle 1 (es decir que se pasa a complemento a 1 y luego se le suma 1).

$$+10 = \underset{\text{Signo}}{0}\ 1010$$

$$-10 = \underset{\text{Signo}}{1}\ 0110$$

	SyM	Ca1	Ca2
$+10_d$	01010	01010	01010
-10_d	11010	10101	10110

Para hacer las cuentas, solo se utilizará la representación de complemento (tanto Ca1 como Ca2).

Sumas y restas mediante sumas en complemento a 1 (Ca1)

Primero, hay pasar las magnitudes (es decir, los valores absolutos) a binario. Se debe verificar que la cantidad de *bits* de ambos números sean iguales, caso contrario, se deberán agregar adelante tantos ceros como sean necesarios.

Una vez que ambos números tengan la misma cantidad de *bits,* se los deberá escribir en complemento a 1. Luego, se debe sumar columna a columna.

Es importante diferenciar la columna del signo y no confundir esta con la del último *carry*. En caso de haber *carry* en complemento a 1, será necesario sumárselo al resultado, para así obtener el resultado final de la cuenta.

Para verificar el resultado final, se debe tener en cuenta que si este es positivo se puede escribir directamente su correspondiente en decimal. En caso de ser negativo, será conveniente escribirlo en SyM (volviendo a complementar), para poder pasarlo a decimal.

Ejemplo: *se quiere realizar la siguiente operación* $37_d - 10_d$ *en binario con signo en Ca1*

1. Escribir el número en binario y completar con ceros.

$37_d = 100101_b$

$10_d = 001010_b$

2. Escribir el número en complemento a 1.

$37_d = 100101_b \rightarrow +37_d = 0100101_b$

$10_d = 001010_b \rightarrow -10_d = 1110101_b$

3. Realizar la suma y si hay *carry* se lo sumo al resultado.

		1		1		1	
		0	1	0	0	1	0	1
	+	1	1	1	0	1	0	1
carry →	1	0	0	1	1	0	1	0
							+	1
signo →		0	1	1	1	0	1	1

Verificación

Como es positivo, directamente se pasa a decimal.

0	0	1	1	0	1	1	$= +27_d$
	2^5	2^4	2^3	2^2	2^1	2^0	
	32	16	8	4	2	1	
+	16+8+2+1 = 27						

El resultado obtenido es el esperado: $37_d - 10_d = +27_d$

Ejemplo: se quiere realizar la siguiente operación $-117_d + 46_d$ en binario, con signo en Ca1.

1. Escribir el número en binario y completar con ceros.

$117_d = 1110101_b$

$46_d = 0101110_b$

2. Escribir el número en complemento a 1.

$117_d = 1110101_b \rightarrow -117_d = 10001010_b$

$46_d = 0101110_b \rightarrow +46_d = 00101110_b$

3. Realizar la suma y si hay *carry* se lo sumo al resultado.

				1	1	1		
+	1	0	0	0	1	0	1	0
	0	0	1	0	1	1	1	0
	1	0	1	1	1	0	0	0

Verificación

Como es negativo, primero se debe complementar y luego pasar a decimal.

1	0	1	1	1	0	0	0	$= -71_d$
1	1	0	0	0	1	1	1	
	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	64	32	16	8	4	2	1	
−	64+4+2+1 = 71							

El resultado obtenido es el esperado: $-117_d + 46_d = -71_d$

Sumas y restas mediante sumas en complemento a 2 (Ca2)

Primero hay que pasar las magnitudes (es decir, los valores absolutos) a binario. Se debe verificar que la cantidad de *bits* de ambos números sean iguales, caso contrario, se deberán agregar adelante tantos ceros como sean necesarios.

Una vez que ambos números tengan la misma cantidad de *bits,* se los deberá escribir en complemento a 2. Luego, se debe sumar columna a columna.

Es importante diferenciar la columna del signo y no confundir esta con la del último *carry*. En caso de haber *carry*, en complemento a 2, este se descarta.

Para verificar el resultado final, se debe tener en cuenta que si este es positivo se puede escribir directamente su correspondiente en decimal. En caso de ser negativo, será conveniente escribirlo en SyM (volviendo a complementar Ca2), para poder pasarlo a decimal.

Ejemplo: *se quiere realizar la siguiente operación* $37_d - 10_d$ *en binario, con signo en Ca2.*

1. Escribir el número en binario y completar con ceros.

   $37_d = 100101_b$

   $10_d = 001010_b$

2. Escribir el número en complemento a 2.

   $37_d = 100101_b \rightarrow +37_d = 0100101_b$

   $10_d = 001010_b \rightarrow -10_d = 1110101_b \rightarrow -10_d = 1110110_b$ (paso a Ca1 y sumo 1)

3. Realizar la suma.

	1			1			
	0	1	0	0	1	0	1
+	1	1	1	0	1	1	0
1	0	0	1	1	0	1	1

Verificación

Como es positivo, se pasa directamente a decimal.

0	0	1	1	0	1	1	$= +27_d$
	2^5	2^4	2^3	2^2	2^1	2^0	
	32	16	8	4	2	1	
+	16+8+2+1 = 27						

El resultado obtenido es el esperado: $37_d - 10_d = 27_d$

Ejemplo: se quiere realizar la siguiente operación $-117_d + 46_d$ en binario, con signo en Ca2.

1. Escribir el número en binario y completar con ceros.

$117_d = 1110101_b$

$46_d = 0101110_b$

2. Escribir el número en complemento a 2.

$117_d = 1110101_b \rightarrow -117_d = 10001010_b \rightarrow -117_d = 10001011_b$ (Ca1 y sumo 1)

$46_d = 0101110_b \rightarrow +46_d = 00101110_b$

3. Realizar la suma.

				1	1	1		
+	1	0	0	0	1	0	1	0
	0	0	1	0	1	1	1	1
	1	0	1	1	1	0	0	1

Verificación

Como es negativo, primero se debe complementar Ca2 y luego pasarlo a decimal.

1	0	1	1	1	0	0	1	$= -71_d$
1	1	0	0	0	1	1	0	
						+	1	
1	1	0	0	0	1	1	1	
	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	64	32	16	8	4	2	1	
−	64+4+2+1 = 71							

El resultado obtenido es el esperado: $-117_d + 46_d = -71_d$

Ejemplo: realizar la siguiente cuenta en binario, con signo $7_d + 6_d$ (Ca2).

$7_d = 111_b \rightarrow +7_d = 0111_b$

$6_d = 110_b \rightarrow +6_d = 0110_b$

		1	1		
	0	1	1	1	
+	0	1	1	0	
	1	1	0	1	

Se esperaba obtener un resultado positivo, sin embargo, al observar la cuenta, el *bit* de signo es 1, lo que nos indica que el resultado sería negativo. ¿Qué sucedió?

Lo que ocurre es que hubo un *overflow* (desbordamiento), es decir, que nos fuimos del rango de representación que la cantidad de *bits* utilizados para hacer la cuenta permite. Esto puede suceder cuando se realizan sumas con signo entre dos números del mismo signo. En este caso, al utilizar 4 *bits* el rango de representación es [-8 ; 7]. Como el resultado esperado es mayor a 7, necesitamos agregar al menos un *bit* más para que esto no suceda.

$7_d = 111_b \rightarrow 7_d = 0111 \rightarrow +7_d = 00111_b$

$6_d = 110_b \rightarrow 6_d = 0110 \rightarrow +6_d = 00110_b$

			1	1		
	0	0	1	1	1	
+	0	0	1	1	0	
	0	1	1	0	1	

Representación de números en binario con y sin signo

En la siguiente tabla se puede ver equivalencia en decimal del número escrito en binario con tres *bits* según la representación en la que está: valor absoluto (VA), signo y magnitud (SyM), complemento a 1 (Ca1) y complemento a 2 (Ca2).

	Sin signo	**Con signo**		
Binario	**VA**	**SyM**	**Ca1**	**Ca2**
000	0	+0	+0	+0
001	1	+1	+1	+1
010	2	+2	+2	+2
011	3	+3	+3	+3
100	4	−0	−3	−4
101	5	−1	−2	−3
110	6	−2	−1	−2
111	7	−3	−0	−1
Rango 3 bits	$[0; 7]$	$[-3; 3]$	$[-3; 3]$	$[-4; 3]$
Rango genérico	$[0; (2^n - 1)]$		$[-(2^{n-1} - 1); (2^{n-1} - 1)]$	$[-(2^{n-1}); (2^{n-1} - 1)]$

De la tabla se deduce que cuando se quiere utilizar números con signo es más conveniente usar la representación de complemento a 2, dado que es el único caso donde el cero tiene un solo equivalente (el elemento nulo es único), por lo que el rango de representación es mayor.