

Arquitectura de Computadoras

Unidad 0 Sistemas de numeración

Edgardo Gho
Carlos Rodríguez



UM 1082 LA2/3

8225

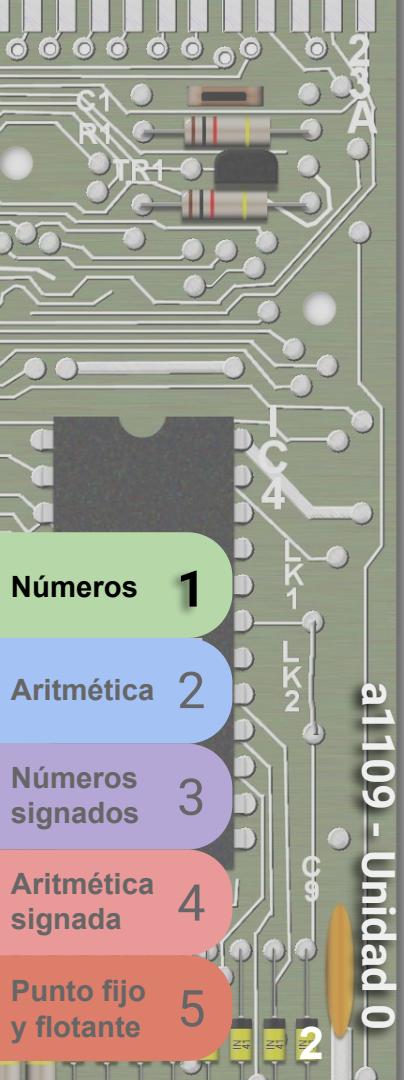
FERRANTI
ULA 2C184E
8214

ZILLOG
Z8400A PS
Z80A CPU
8220

SINCLAIR
RESEARCH 8223Pg
D2364C 649 © 1981

Toshiba
TMM2016P
2-EE4

Sistemas de numeración



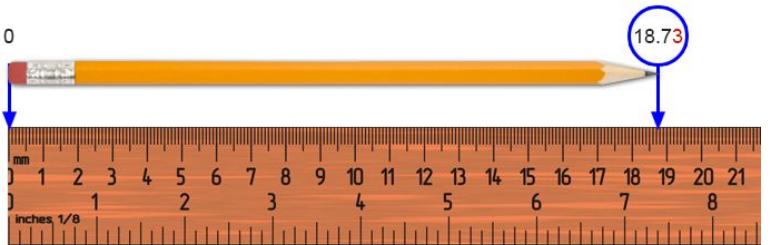
Números 1

Aritmética 2

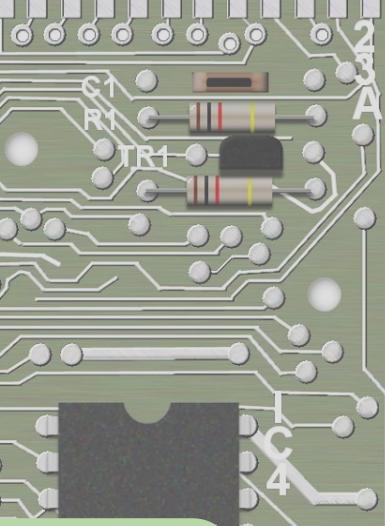
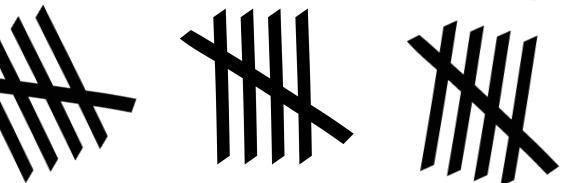
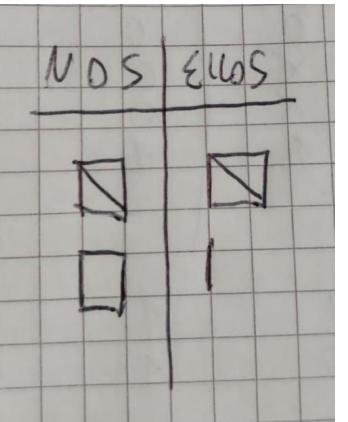
Números
signados 3

Aritmética
signada 4

Punto fijo
y flotante 5



\$1244,56



Números 1

Aritmética 2

Números signados 3

Aritmética signada 4

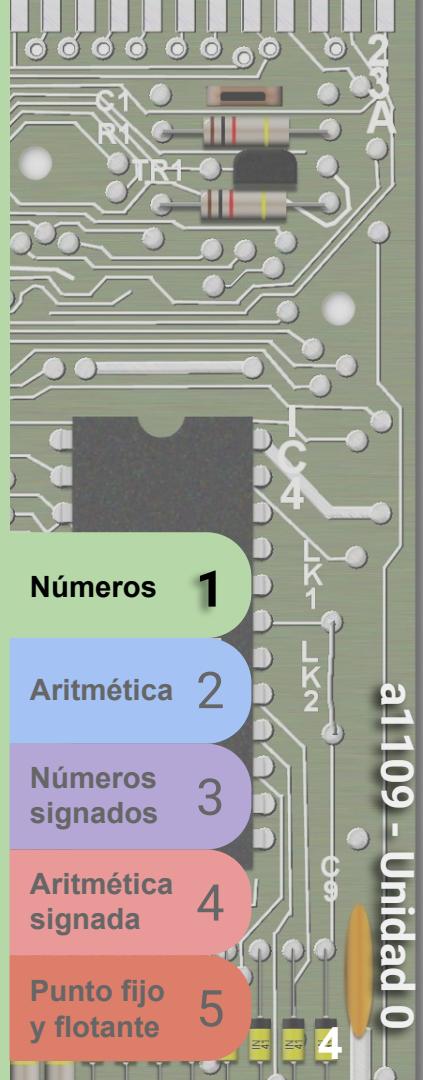
Punto fijo y flotante 5

al109 - Unidad 0

Dígitos

Se define como **dígito** a cada uno de los símbolos diferentes que constituyen al sistema de numeración, y se define asimismo como **base** del sistema de numeración a la cantidad de dígitos que lo forman.

Base	Dígitos
2	0 , 1
4	0 , 1 , 2 , 3
8	0 , 1 , 2 , 3 , 4 , 5 , 6 , 7
10	0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9
16	0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E , F



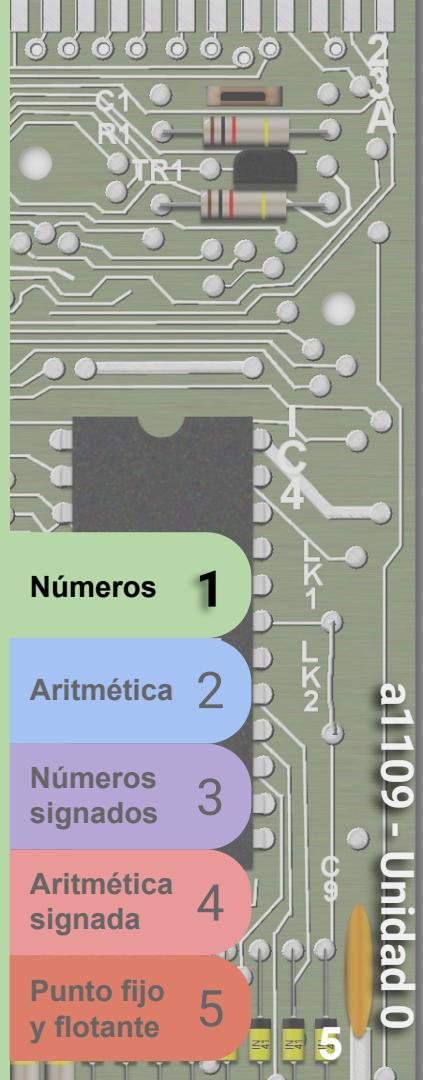
Sistema de numeración posicional

$$13203_{10} = 1 * 10^4 + 3 * 10^3 + 2 * 10^2 + 0 * 10^1 + 3 * 10^0$$

↑ ↑ ↑ ↑ ↑
Decenas Unidades Centenas Decenas Unidades
De Mil De Mil De Mil De Mil De Mil

$$N_B = a_0 * B^0 + a_1 * B^1 + \dots + a_n * B^n$$

$$N_B = \sum_{i=0}^n a_i * B^i$$



Otras bases y su conversión a decimal

$$11010_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$

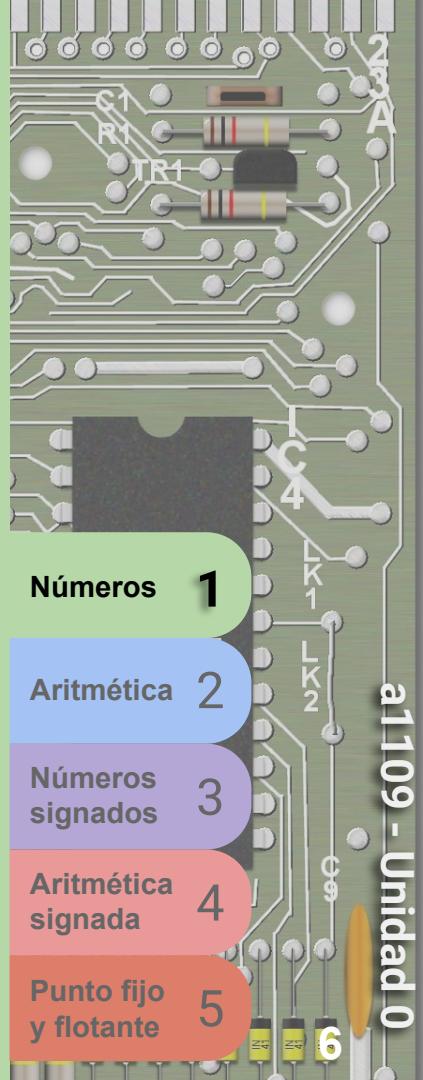


$$11010_2 = 1 * 16 + 1 * 8 + 0 * 4 + 1 * 2 + 0 * 1 = 26_{10}$$

$$03FE_{16} = 0 * 16^3 + 3 * 16^2 + 15 * 16^1 + 14 * 16^0$$

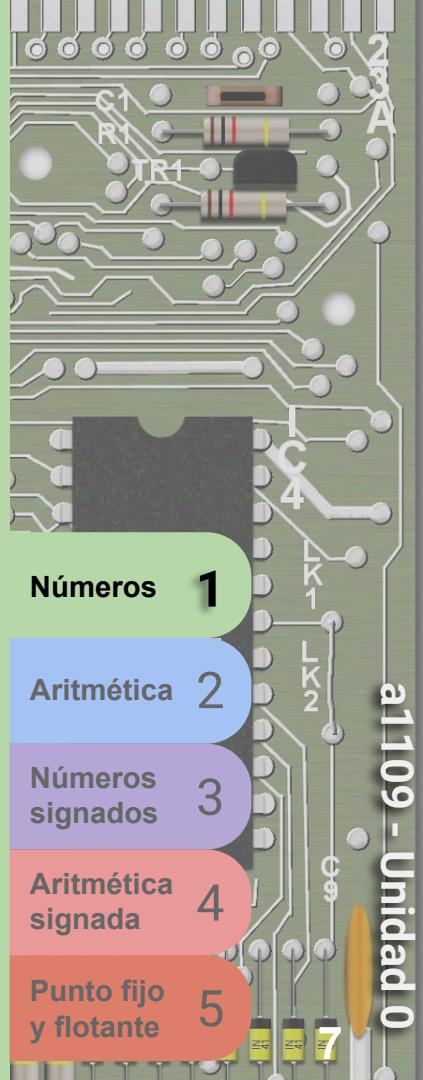


$$03FE_{16} = 3 * 256 + 15 * 16 + 14 = 1022_{10}$$



Conversión rápida entre bases potencias enteras de 2

Valor	Base 2	Base 4	Base 8	Base 16
0	0000	0	0	0
1	0001	1	1	1
2	0010	2	2	2
3	0011	3	3	3
4	0100	10	4	4
5	0101	11	5	5
6	0110	12	6	6
7	0111	13	7	7
8	1000	20	10	8
9	1001	21	11	9
10	1010	22	12	A
11	1011	23	13	B
12	1100	30	14	C
13	1101	31	15	D
14	1110	32	16	E
15	1111	33	17	F

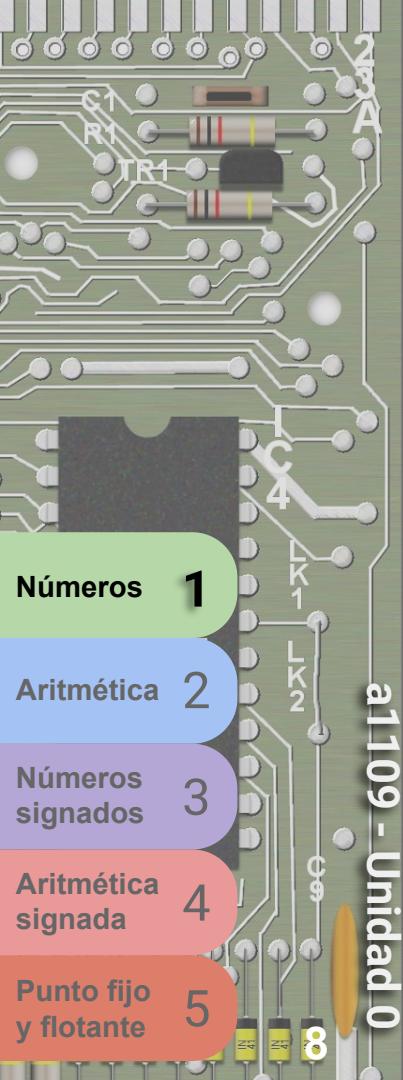


Potencias de 2

2 ^{fila col}	0	1	2	3	4	5	6	7	8	9
0	2 ⁰⁰	2 ⁰¹	2 ⁰²	2 ⁰³	2 ⁰⁴	2 ⁰⁵	2 ⁰⁶	2 ⁰⁷	2 ⁰⁸	2 ⁰⁹
1	2 ¹⁰	2 ¹¹	2 ¹²	2 ¹³	2 ¹⁴	2 ¹⁵	2 ¹⁶	2 ¹⁷	2 ¹⁸	2 ¹⁹
2	2 ²⁰	2 ²¹	2 ²²	2 ²³	2 ²⁴	2 ²⁵	2 ²⁶	2 ²⁷	2 ²⁸	2 ²⁹
3	2 ³⁰	2 ³¹	2 ³²	2 ³³	2 ³⁴	2 ³⁵	2 ³⁶	2 ³⁷	2 ³⁸	2 ³⁹
4	2 ⁴⁰	2 ⁴¹	2 ⁴²	2 ⁴³	2 ⁴⁴	2 ⁴⁵	2 ⁴⁶	2 ⁴⁷	2 ⁴⁸	2 ⁴⁹

2 ^{fila col}	0	1	2	3	4	5	6	7	8	9
0 B	1	2	4	8	16	32	64	128	256	512
10 K	1024	2048	4096	8192	16384	32768	65536	128K	256K	512K
20 M	1M	2M	4M	8M	16M	32M	64M	128M	256M	512M
30 G	1G	2G	4G	8G	16G	32G	64G	128G	256G	512G
40 T	1T	2T	4T	8T	16T	32T	64T	128T	256T	512T

Ejemplo: $2^{16} = 2^6 * 2^{10} = 64 \text{ K} = 65536$



Conversión de base decimal a base 2

Número	Base	Resultado	Resto
125 %	2		

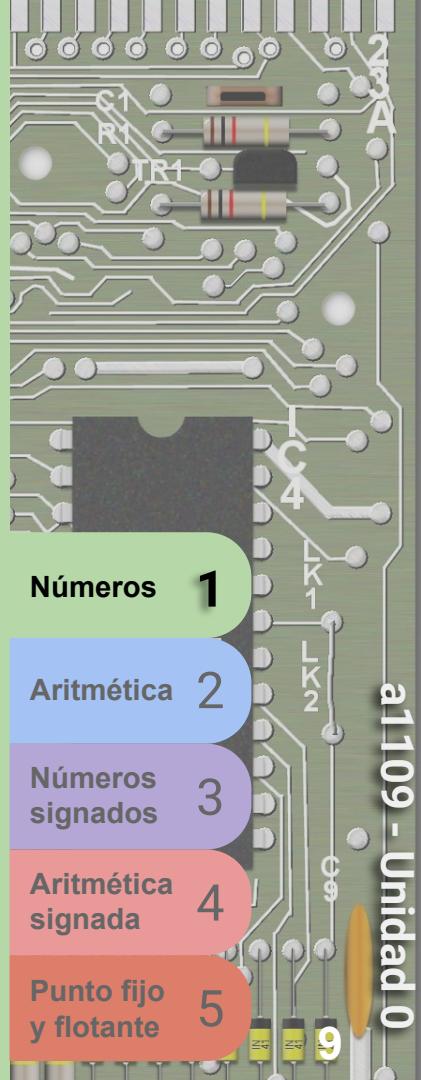
Número	Base	Resultado	Resto
125	2	= 62	= 1
62	2		

Número	Base	Resultado	Resto
125 %	2	= 62	1
62	2	31	0
31	2	15	1
15	2	7	1
7	2	3	1
3	2	1	1
1	2	0	1

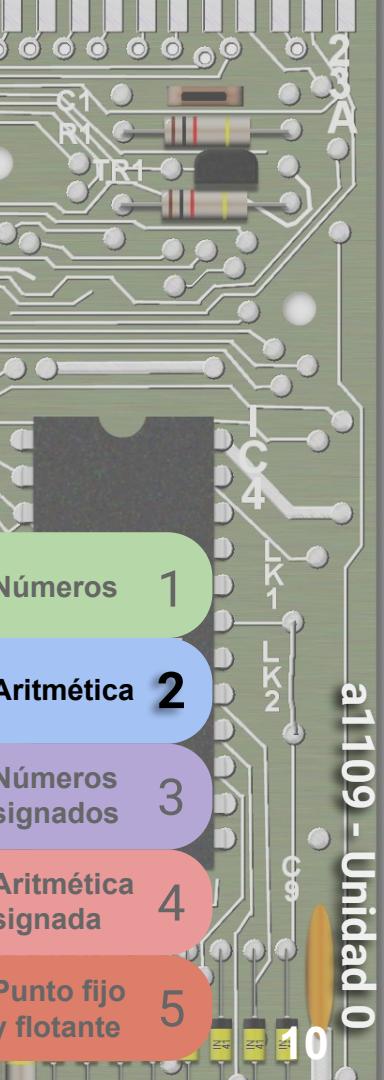
1
2
4
8
16
32
64

1111101

64+32+16+8+4+1



Aritmética (sumas, restas, comparaciones sin signo)



3 bits Valor

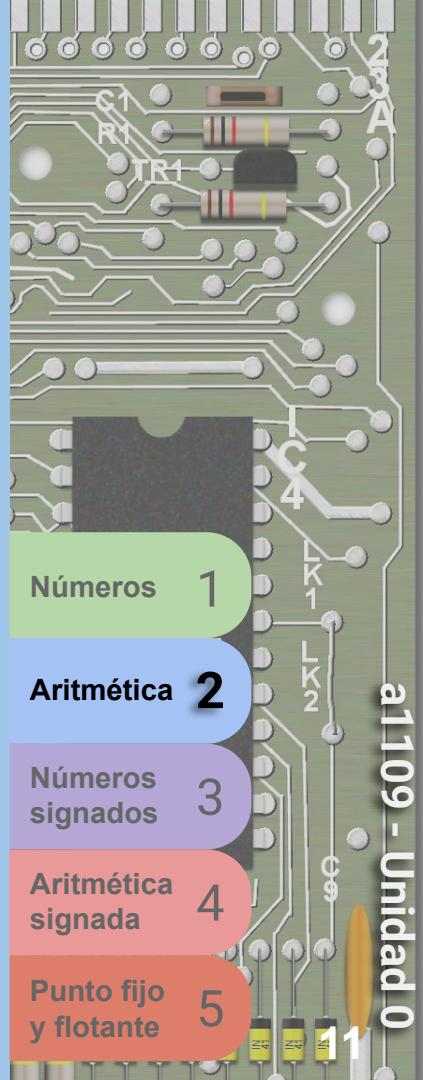
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Si tenemos un mundo donde solo existen números de 3 bits, entonces podemos representar 2^3 elementos distintos. Esto podemos generalizarlo para todas las bases B y todas las cantidades de dígitos N como

$$B^N$$

Dado que el primer elemento va a representar el cero, el último elemento va a representar $B^N - 1$. En este caso de $2^3 - 1 = 7$. El primer elemento es 0 y el último es 7.



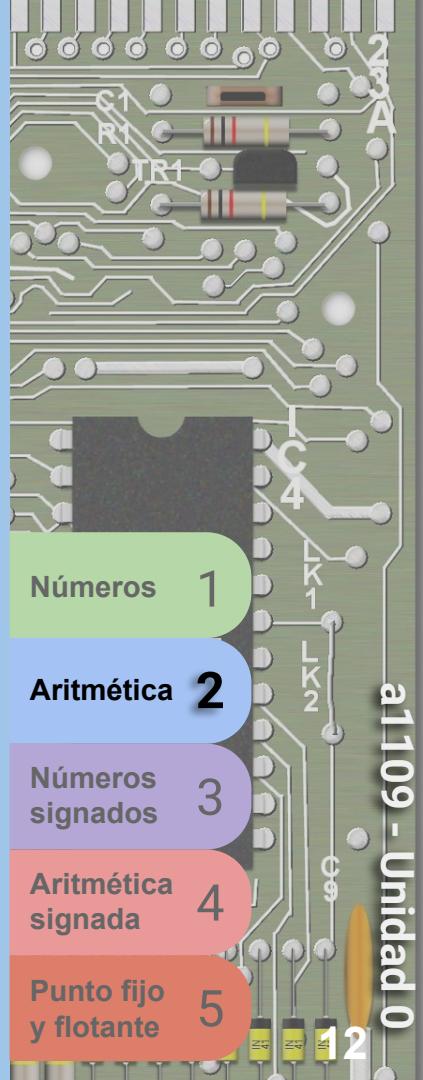
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario			Decimal	
0	1	0	2	
+ 0	1	1	3	

Planteamos la suma de dos números de 3 bits sin signo (010 y 011).



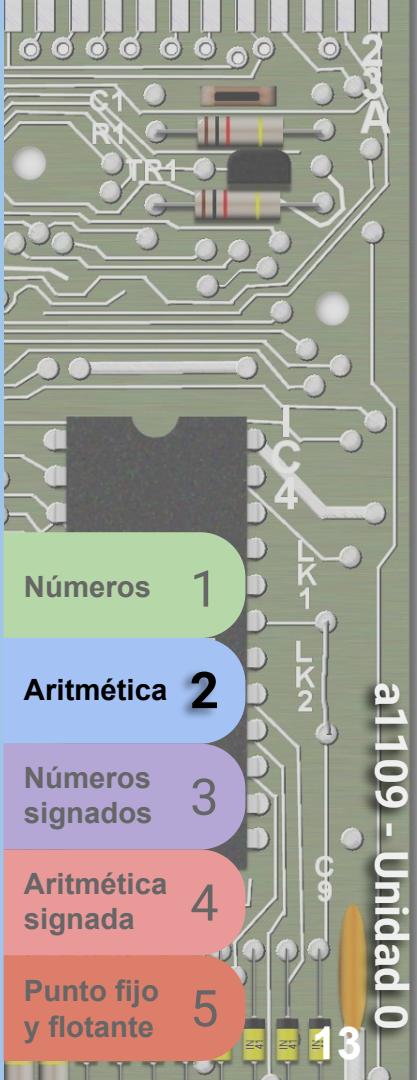
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} & 0 & 1 & 0 & | & 2 \\ + & 0 & 1 & 1 & | & 3 \\ \hline & & & & & 5 \end{array}$$



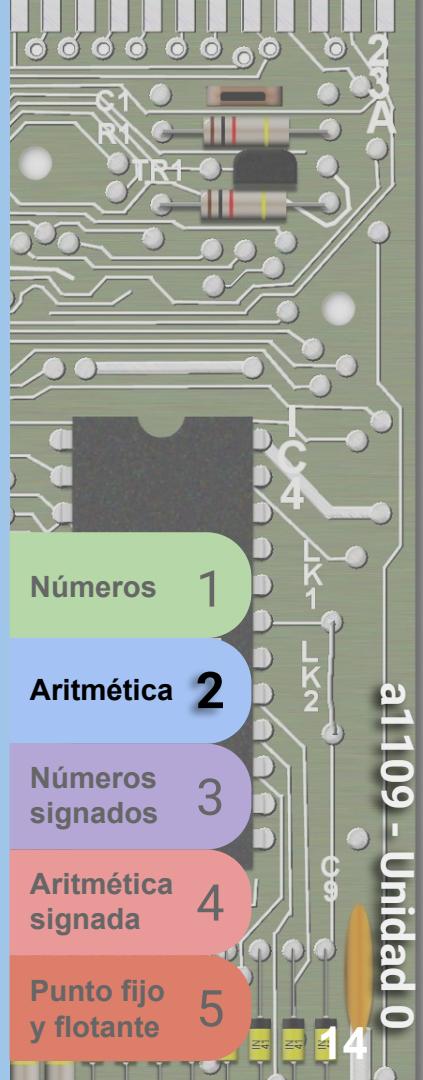
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

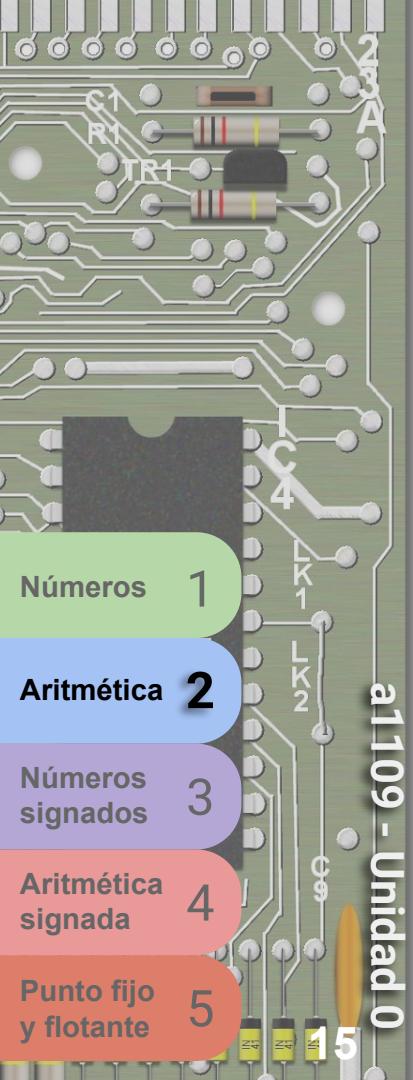
$$\begin{array}{r} & 0 & 1 & 0 & | & 2 \\ + & 0 & 1 & 1 & | & 3 \\ \hline & & & & | & 1 \\ & & & & & 5 \end{array}$$



3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal			
	{ }			
	{ }			
	0	1	0	2
+	0	1	1	3
				10 1 5
↑				
$10_2 = 2_{10}$				



3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario		Decimal
+		
0 1 0	2	3
0 1 1	3	5
0 1 5		

1

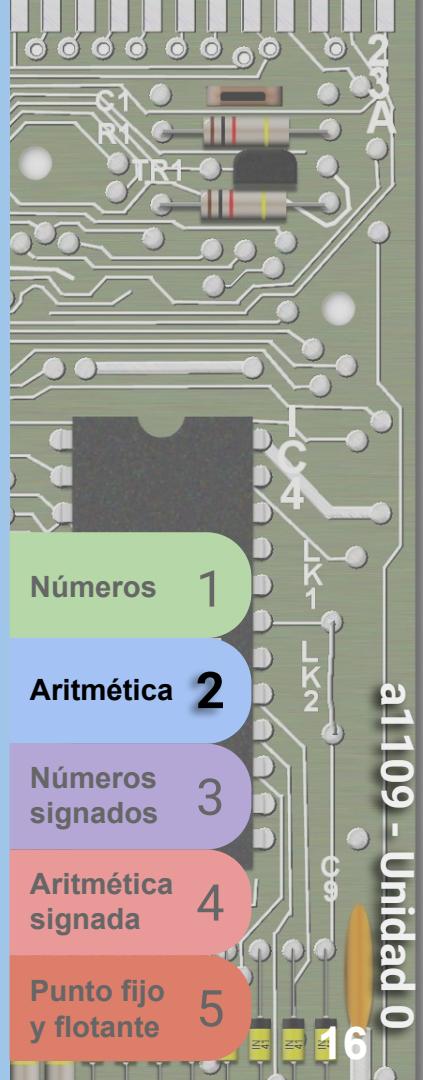
¿Cuál es el peor caso con el acarreo?

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 10_2$$

$$1 + 1 + 1 = 11_2$$

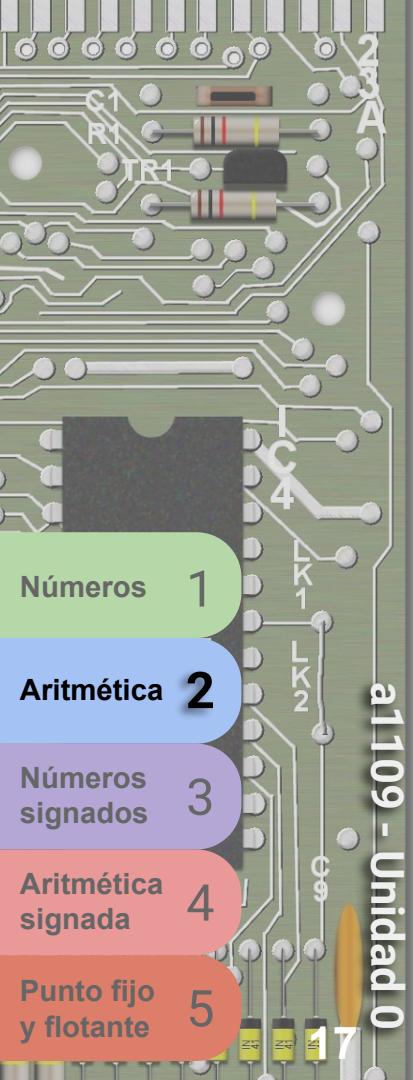


3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario		Decimal
1		
0 1 0		2
+ 0 1 1		3
		1 0 1 5

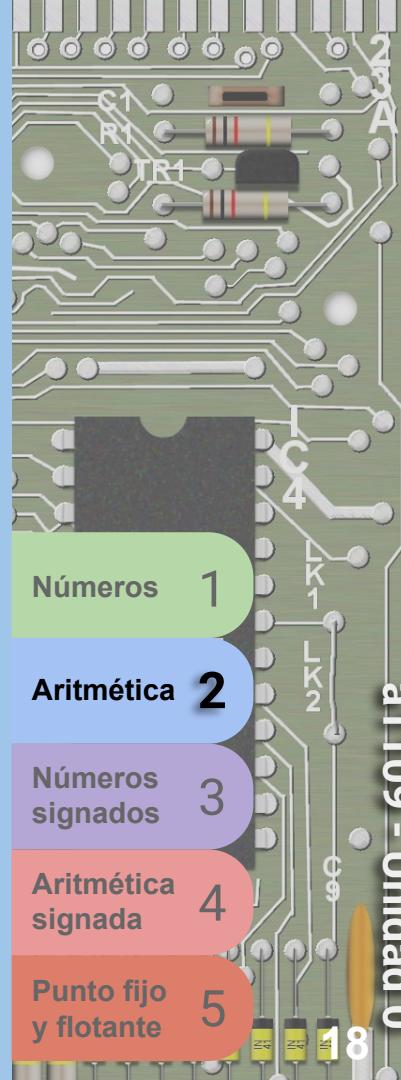
Esta suma de dos números de 3 bits sin signo (010 y 011) existe dentro del mundo de números de 3 bits sin signo (101).



3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
1 1 0	6
+	
0 1 1	3



Planteamos ahora la suma de dos números de 3 bits sin signo ($110 + 011$)

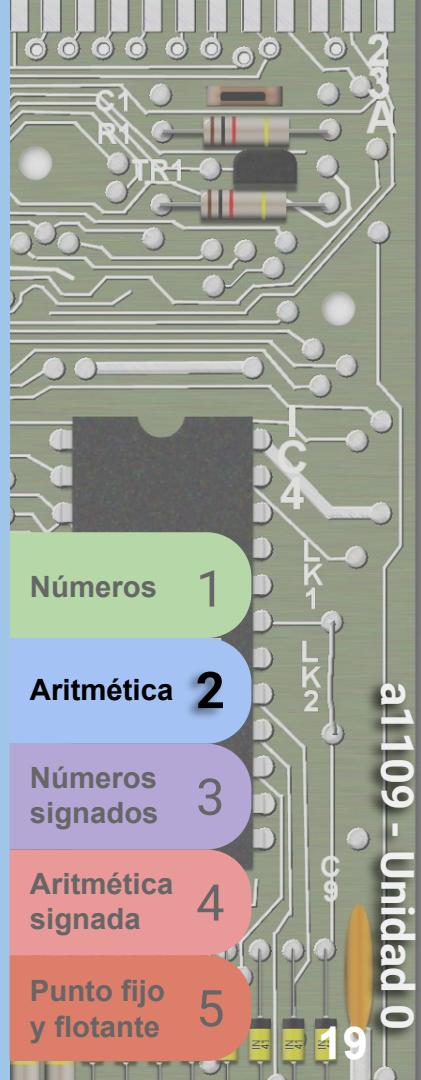
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} & 1 & 1 & 0 & | & 6 \\ + & 0 & 1 & 1 & | & 3 \\ \hline & & & & & 9 \end{array}$$



En decimal, el resultado es un solo dígito. A priori ya vemos que ese valor no existe en el mundo de números de 3 bits sin signo.

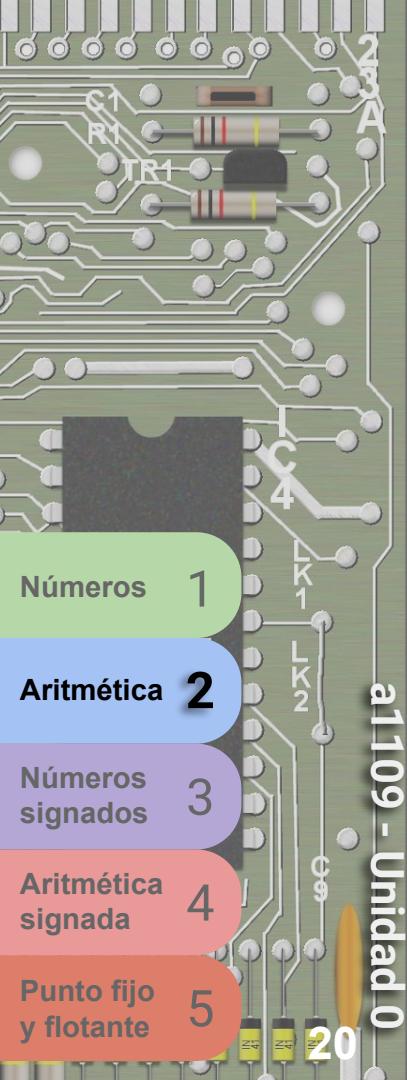
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r}
 & 1 & 1 & 0 & | & 6 \\
 + & 0 & 1 & 1 & | & 3 \\
 \hline
 & & & & | & 1 \quad 9
 \end{array}$$



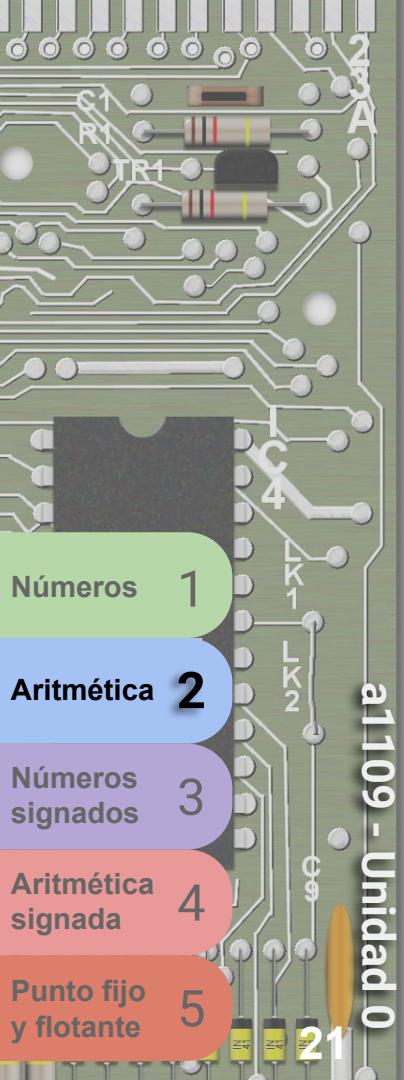
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} & 1 & 1 & 0 & | & 6 \\ + & 0 & 1 & 1 & | & 3 \\ \hline & 10 & 1 & 9 & & \end{array}$$



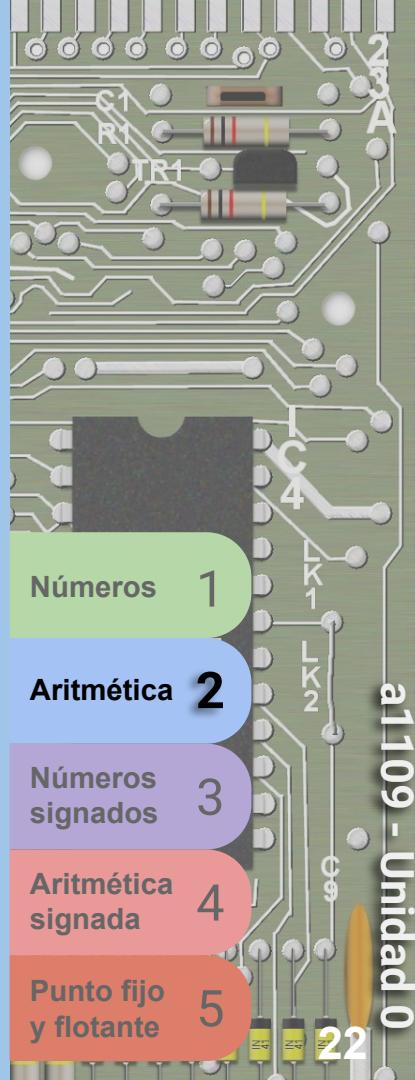
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} & \overset{\text{1}}{+} \\ \begin{array}{r} 1 \quad 1 \quad 0 \\ + \quad 0 \quad 1 \quad 1 \end{array} & \begin{array}{r} | \\ 6 \quad 3 \end{array} \\ \hline & \overset{\text{0}}{=} \quad \begin{array}{r} 1 \quad 9 \end{array} \end{array}$$

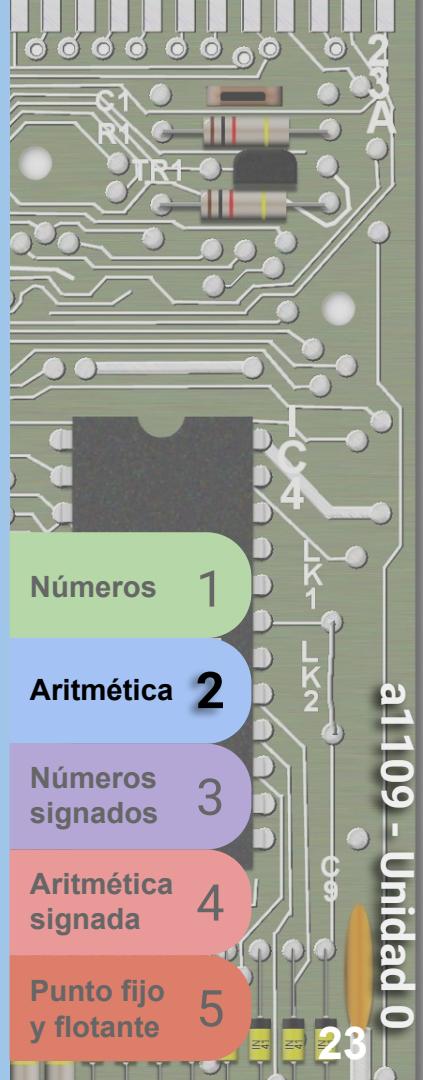


3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
1	
1 1 0	6
+	
0 1 1	3
10 0 1	9

¡¡¡Nos quedamos sin bits!!! no importa, hacemos de cuenta que el número es de 4 bits.



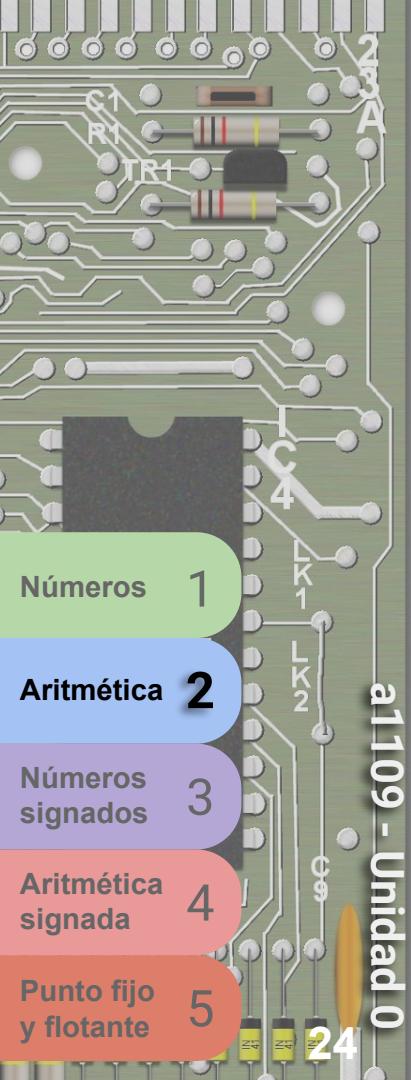
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal			
1	1			
0	1	1	0	6
0	0	1	1	3
0	0	1	9	

Expandimos los números originales a 4 bits agregando un cero delante, y acarreamos el 1. Ahora podemos hacer la suma.



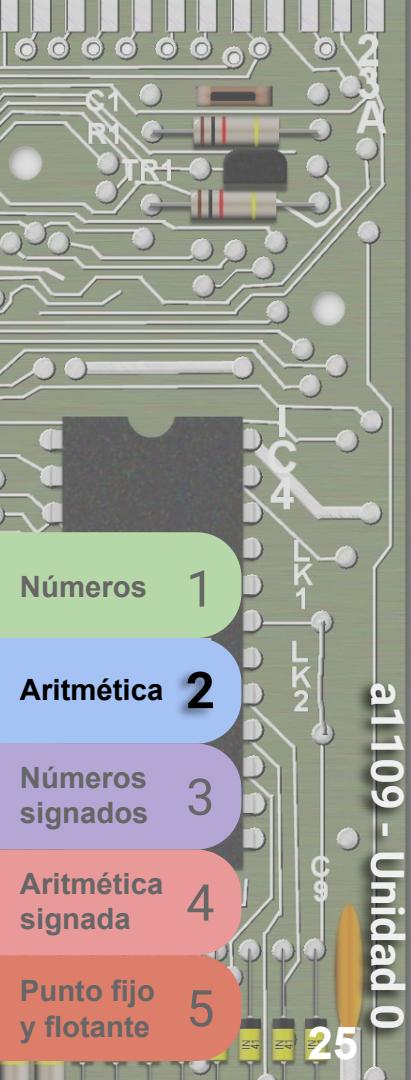
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal			
1	1			
0	1	1	0	6
0	0	1	1	3
1	0	0	1	9

El resultado tiene un bit extra. Ese bit se llama bit de acarreo (carry). Indica que el resultado de la suma excede el mundo de los 3 bits sin signo.

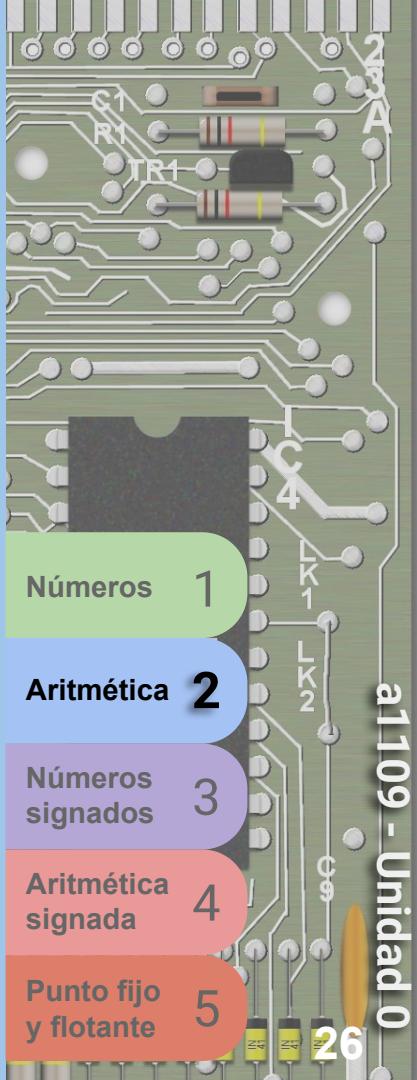


3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
1	1
0 1 1 0	6
0 0 1 1	3
1 0 0 1	9

El resultado 001 no es 9... y tiene sentido ya que 9 no se puede representar con 3 bits sin signo. Este bit de carry indica que el resultado de la operación no existe en este mundo de 3 bits sin signo.



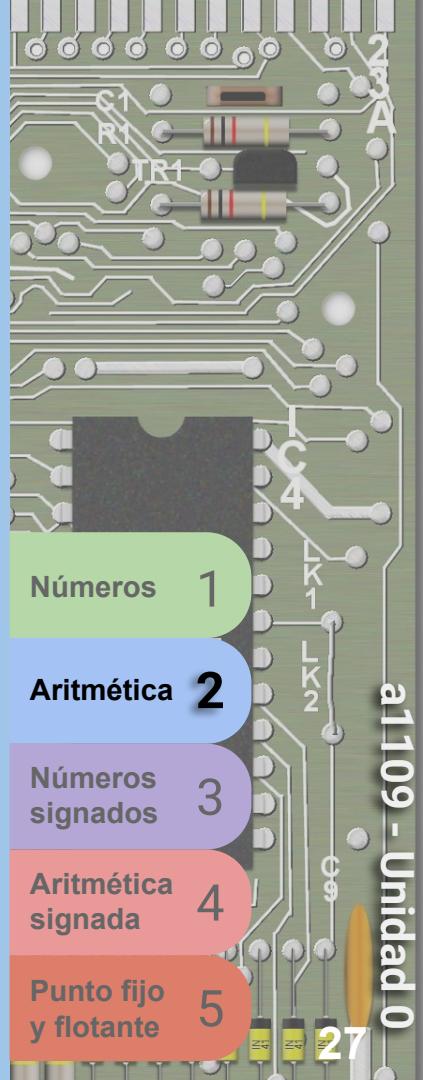
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
1	0	1	5
-	0	1	3

Planteamos ahora la resta de dos números de 3 bits sin signo ($101 + 011$). A 101 (5) lo llamamos **minuendo** y a 011 (3) lo llamamos **sustraendo**.



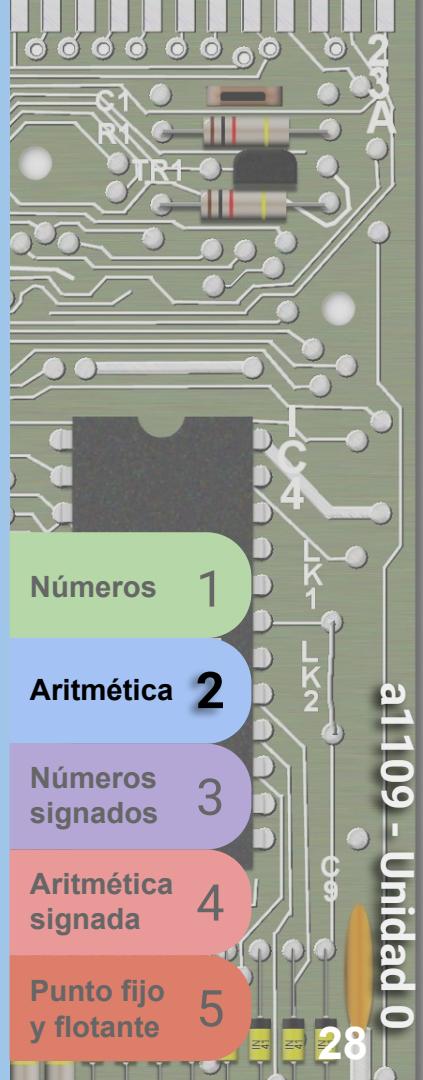
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
1	0	1	5
-	0	1	3
			2

Vemos que en decimal la resta es 2, y que 2 existe en nuestro mundo de 3 bits sin signo (010) por ende tiene buenas chances de funcionar bien.



3 bits Valor

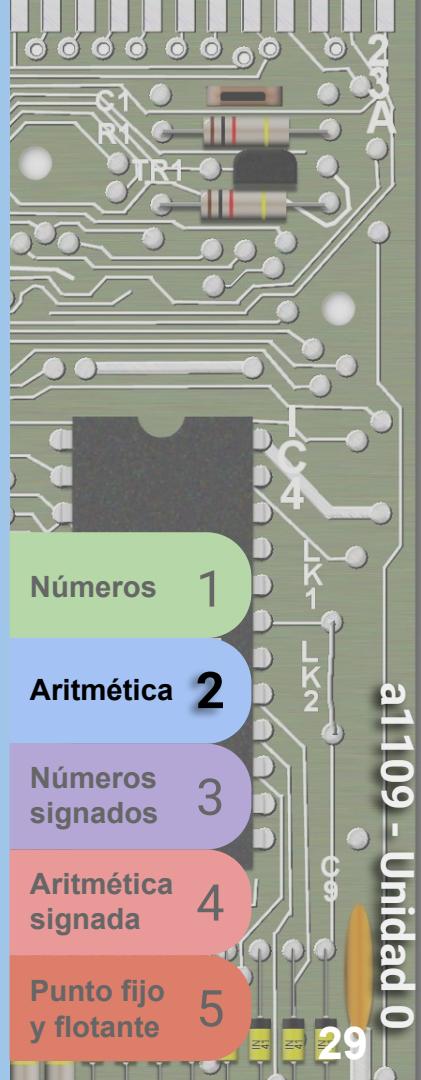
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} 1 & 0 & 1 & | & 5 \\ - & 0 & 1 & 1 & \\ \hline & & & 0 & 2 \end{array}$$

Restamos los bits de la columna menos significativa, $1-1=0\dots$ sin problemas



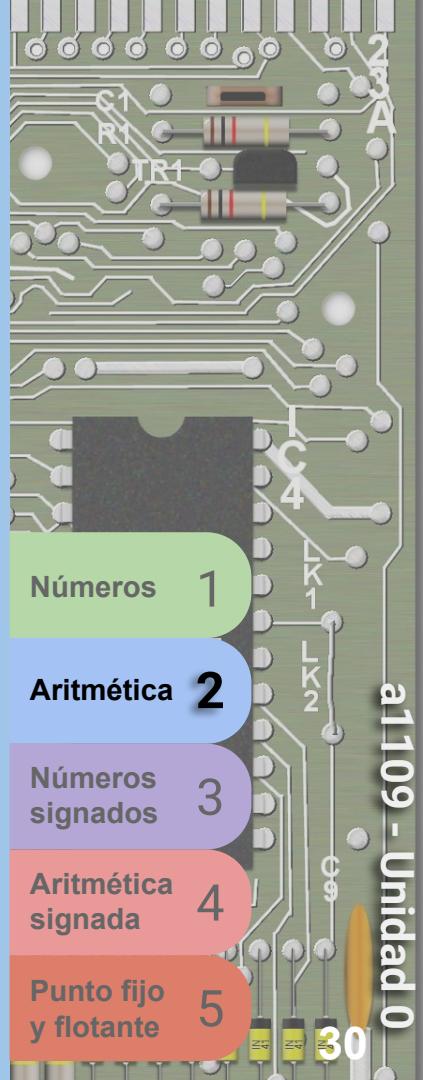
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
1	0	1	5
-	0	1	3
	?	0	2

¡¡¡La siguiente columna es 0-1!!! Para poder restar necesitamos que el minuendo sea mayor o igual al sustraendo, pero en este caso no ocurre. Le “pedimos” prestado a la columna que sigue...



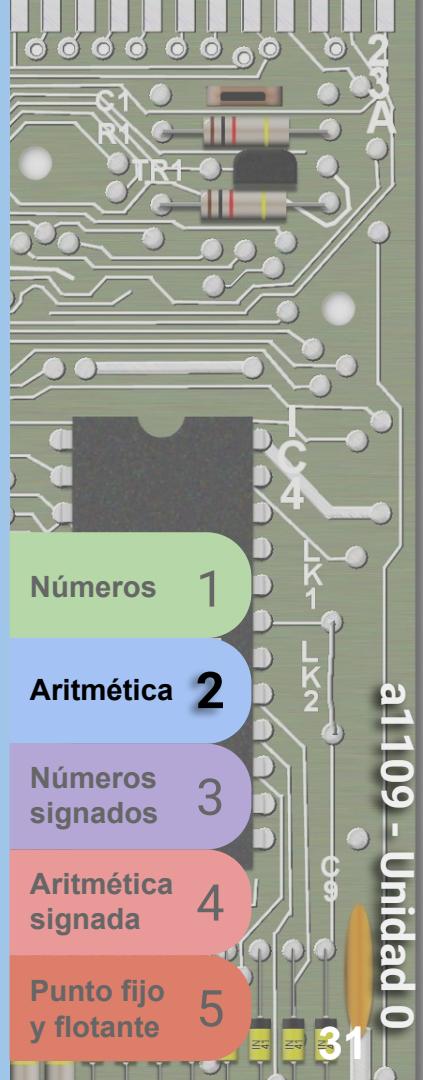
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
1	0	1	5
-	0	1	3
	?	0	2

En la columna que sigue el minuendo tiene un 1, así que se transforma en 0 y pasa a la columna anterior...



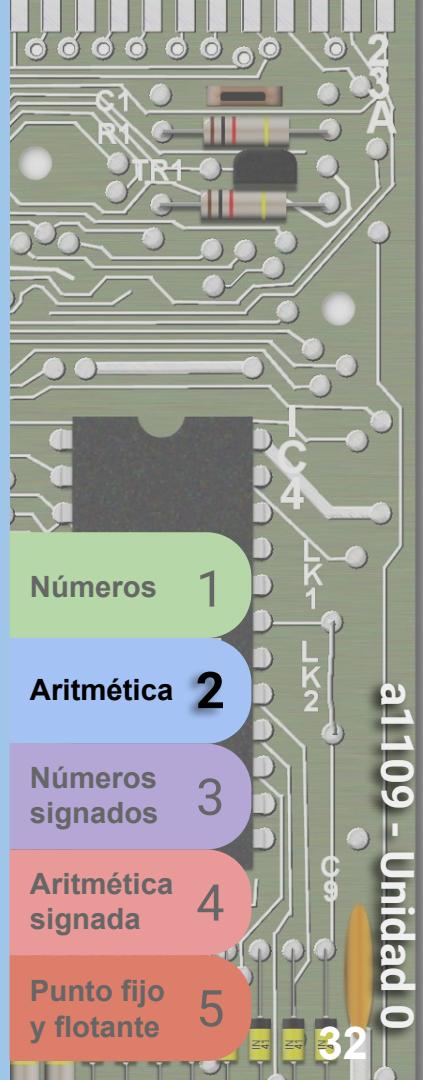
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario			Decimal	
+	1	0	1	5
-	0	1	1	3
?	0	2		

Ahora tenemos $10_2 - 1$ lo que es igual a $2_{10} - 1 = 1$



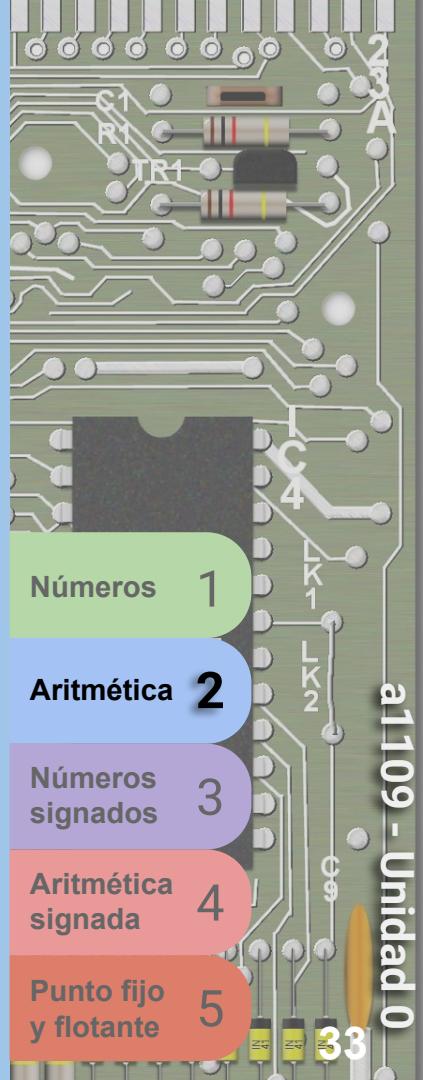
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
0	1	0	5
1	0	1	3
-	0	1	2
0	1	0	2

En la columna de la izquierda el 1 original pasó a ser 0. Nos queda 0-0 que es igual a 0.

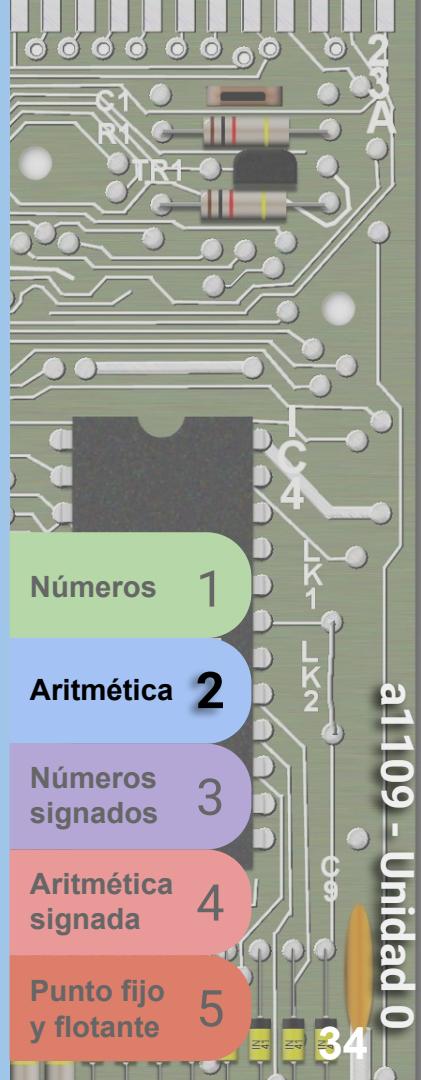


3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

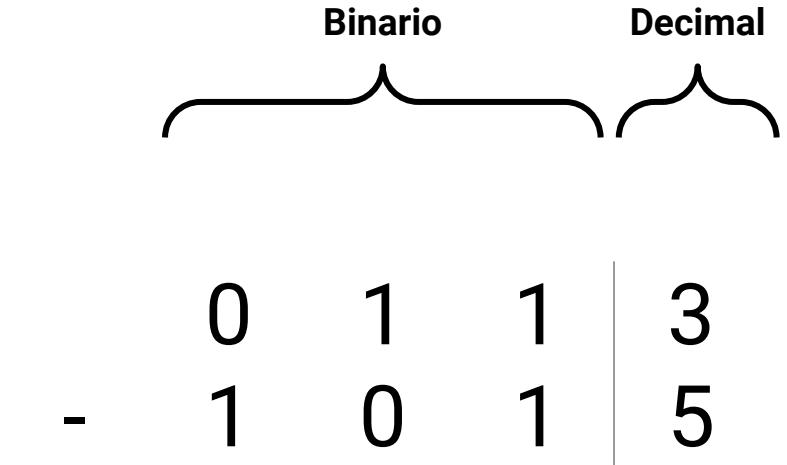
Binario		Decimal
0 1 0 1	-	5
0 1 1		3
<hr/>		
0 1 0		2

Salvo por el “préstamo” (en inglés Borrow), la operación resultó en 010. En nuestro universo de 3 bits sin signo podemos restar 5-3 sin problemas. El Borrow intermedio se resolvió bien.

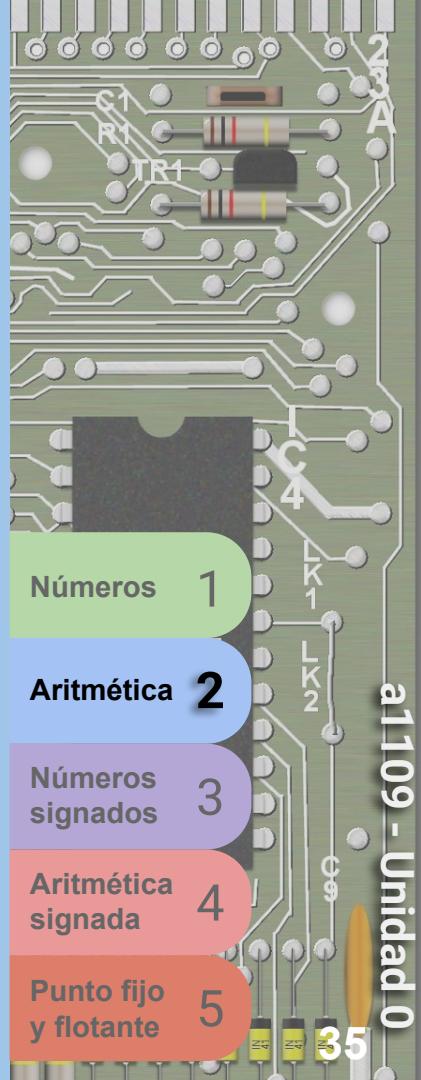


3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo



Ahora invertimos **minuendo** y **sustraendo**. Nótese que el minuendo es menor que el sustraendo.



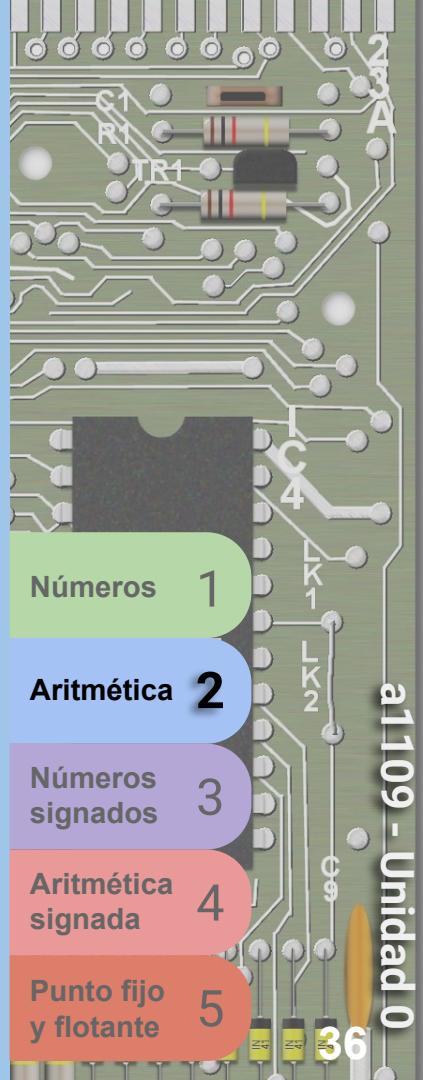
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
0 1 1	3
- 1 0 1	5
	-2

En decimal conocemos los números negativos, pero en nuestro universo de 3 bits sin signo eso no existe... seguro va a haber problemas con esta operación.



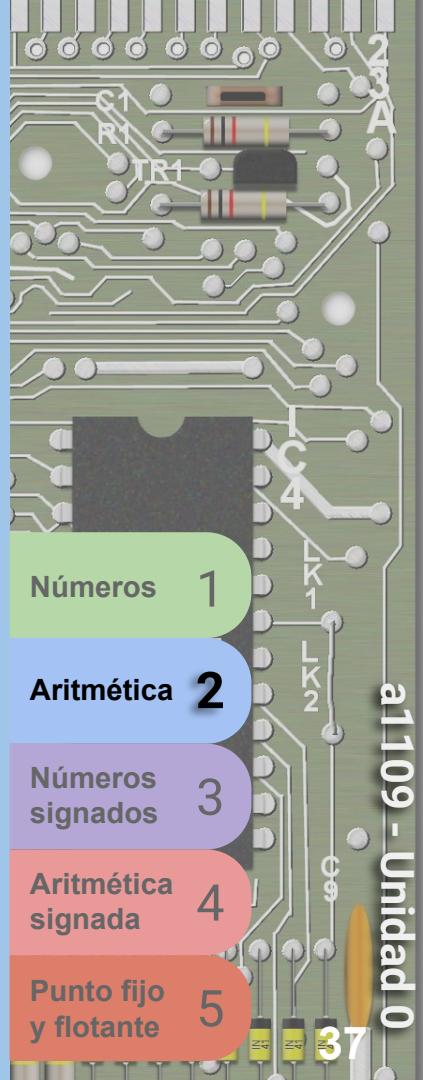
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
0 1 1	3
- 1 0 1	5
0 -2	

Bits menos significativos, 1-1 .. sin problemas por ahora.



3 bits Valor

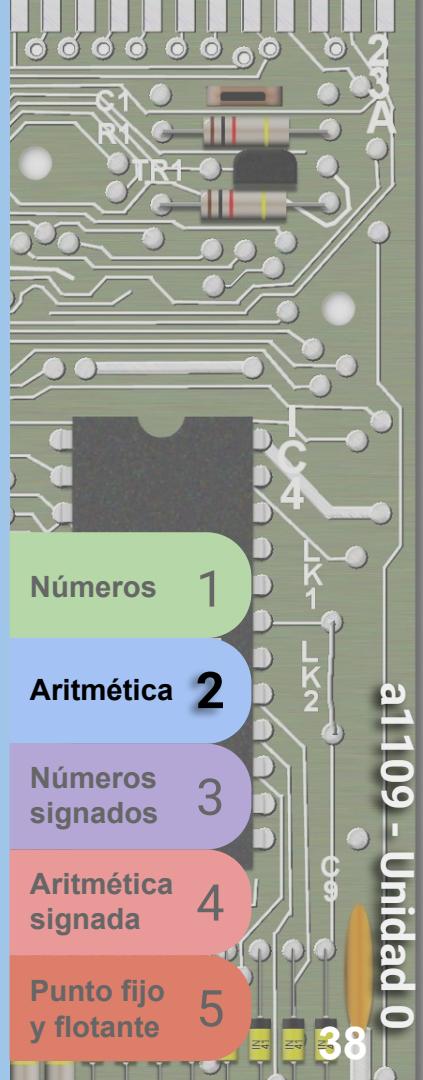
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario Decimal

$$\begin{array}{r} 0 \quad 1 \quad 1 \quad | \quad 3 \\ - \quad 1 \quad 0 \quad 1 \quad | \quad 5 \\ \hline 1 \quad 0 \quad -2 \end{array}$$

Ahora $1-0 = 1$, sin problemas todavía.



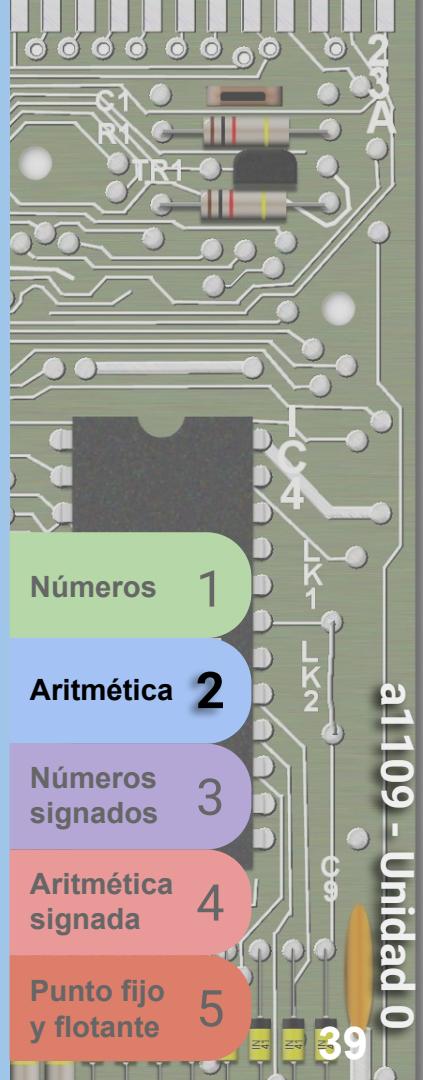
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal
0 1 1	3
- 1 0 1	5
?	1 0 -2

En la columna más significativa tenemos 0-1, y sabemos que para eso tenemos que pedir un préstamo. Pero no tenemos a nadie a quien pedir.



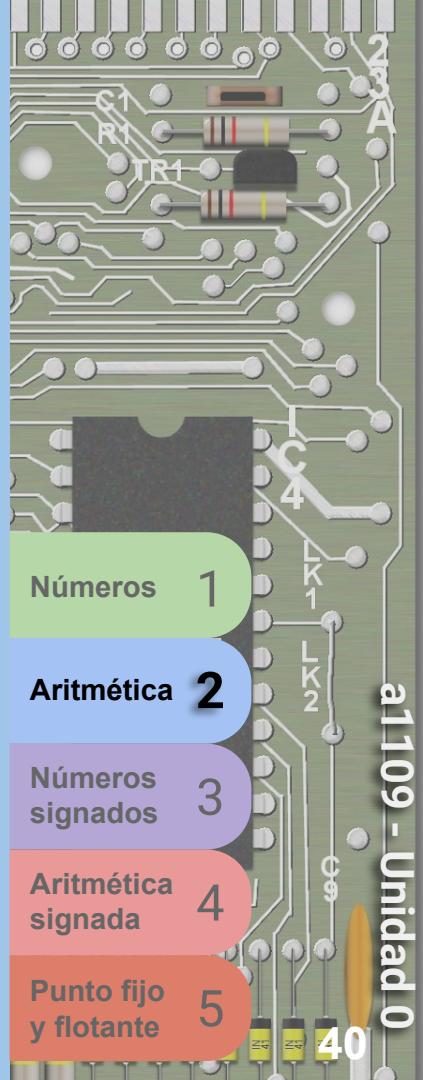
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
1011101010101010	1	0	1
- 1011101010101010	1	0	1
1 ? 1 0 -2	?	1	0

Hacemos algo de “contabilidad creativa” e inventamos un préstamo. Ahora tenemos $10_2 - 1$ por ende podemos resolver.



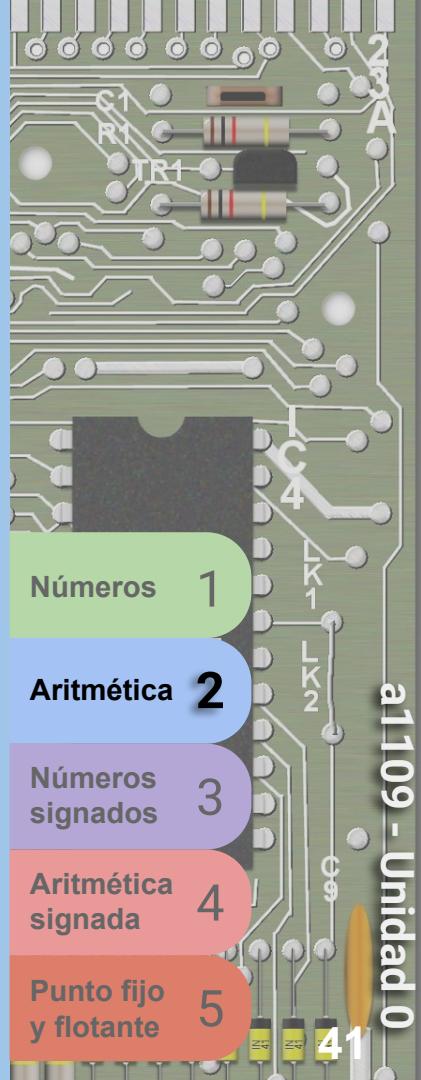
3 bits Valor

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

Binario	Decimal		
101101	10	1	1
-	1	0	1
1	1	1	0
			-2

Hacemos algo de “contabilidad creativa” e inventamos un préstamo. Ahora tenemos $10_2 - 1$ por ende podemos resolver, pero nos queda un 1 en el bit más significativo.



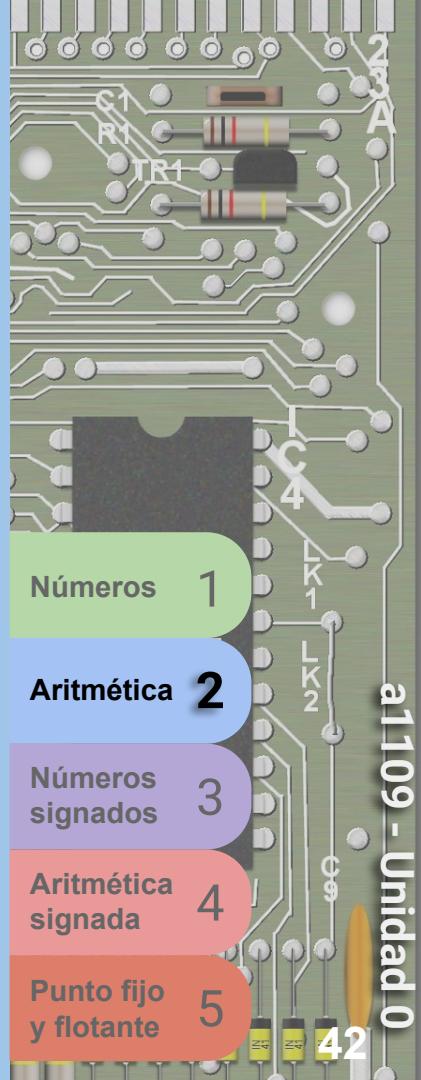
3 bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Un mundo de 3 bits sin signo

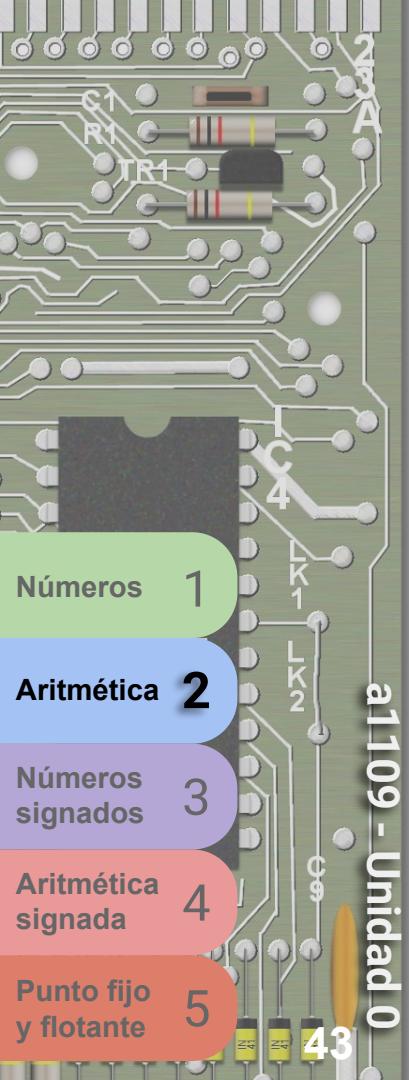
Binario	Decimal		
101	5		
-			
101	5		

1110	10	10	3
	1	0	5
	1	1	
	0	1	
	-2		

Este bit se llama bit de “préstamo”(en inglés Borrow) e indica que el resultado no puede ser representado en 3 bits sin signo. También indica que el minuendo es menor que el sustraendo.



Comparando números sin signo

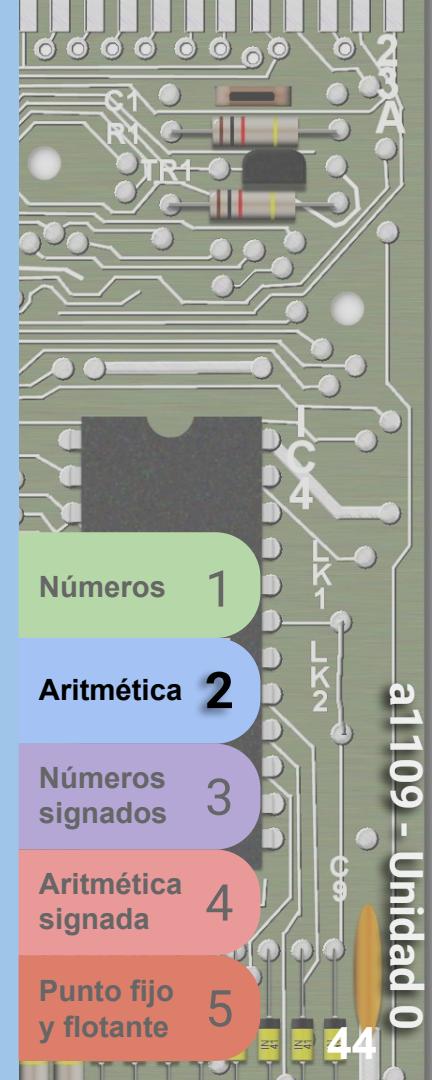


Comparamos haciendo una resta

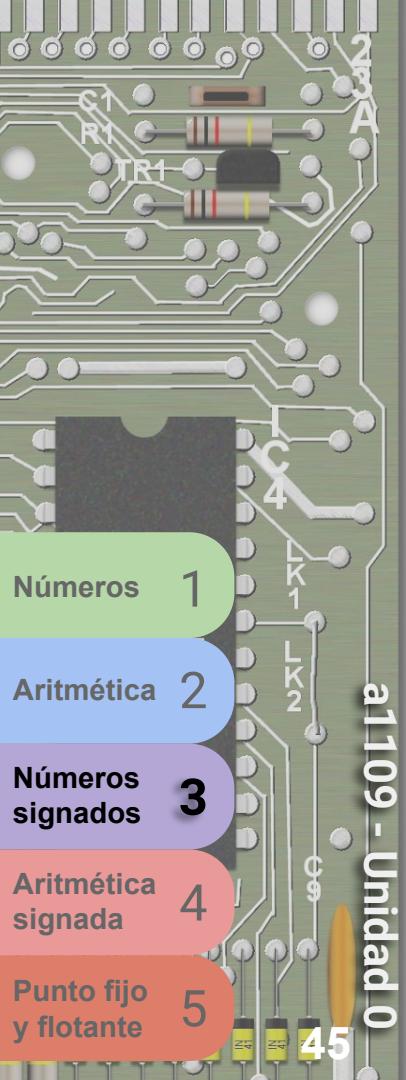
Cuando queremos comparar dos números, por ejemplo A y B, y estos son números sin signo, realizamos la resta entre ambos.

Siendo A el minuendo y B el sustraendo

- Si $A-B = 0$ entonces $A=B$
- Si $A-B$ da Borrow, entonces $A < B$
- Si $A-B$ no da Borrow, entonces $A > B$



Complementos y números signados

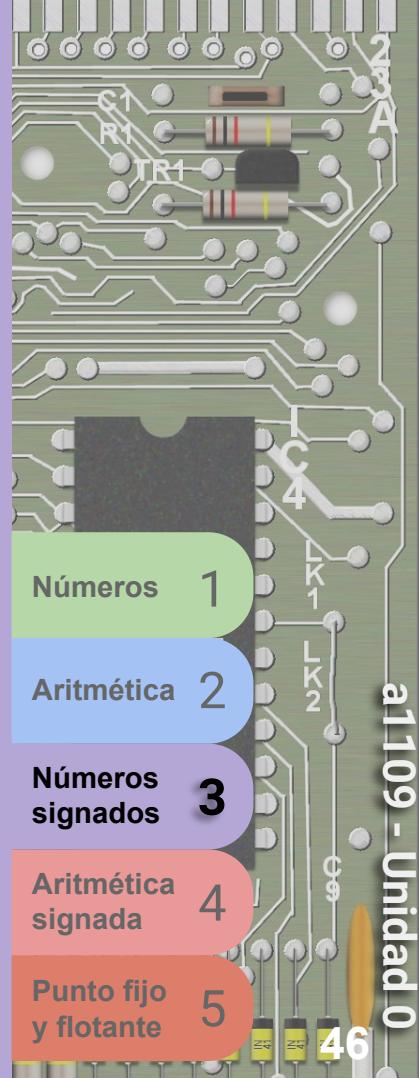


Complemento de un número

Se define como **complemento de un número con respecto a la potencia más cercana de la base** como **la diferencia entre dicha potencia de la base y el número** que se está complementando.

Dado que es muy molesto estar diciendo “*complemento de un número con respecto a la potencia más cercana de la base*” lo resumimos como **“complemento a la base”**.

Se entiende que cuando hablamos de complemento a la base nos referimos a la definición.



Complemento de un número decimal

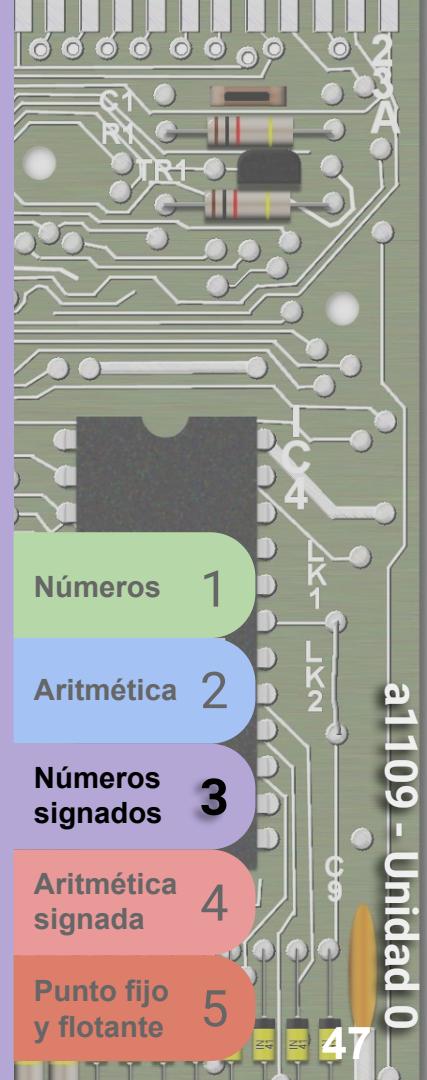
Podemos generalizar el cálculo del complemento para un número N en cualquier base B, donde la cantidad de dígitos que componen a N es X, el complemento de N es:

$$CB(N) = B^X - N$$

Vemos que si calculamos $CB(CB(N))$ obtenemos nuevamente N.

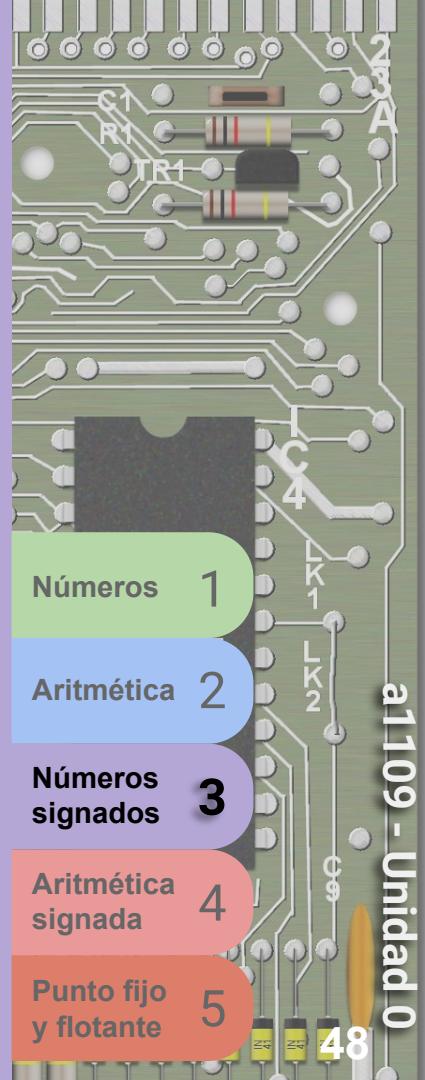
También podemos generalizar el complemento a la base menos uno como:

$$CB_{-1}(N) = B^X - 1 - N$$



Complemento de un número decimal

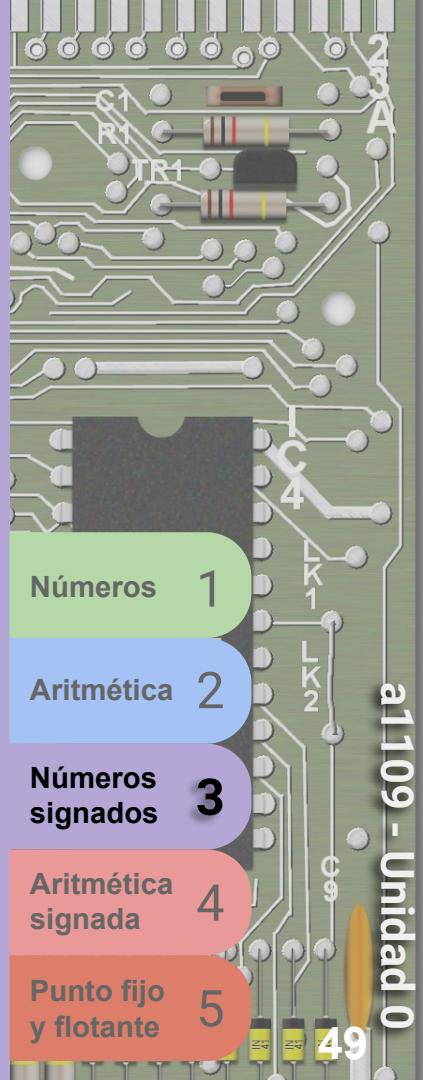
Tomemos un número decimal como el **234**. La cantidad de dígitos que posee es **3**. La potencia más cercana de la base (10) se calcula como la base elevada a la cantidad de dígitos, o sea $10^3 = 1000$.



Complemento de un número decimal

$$\begin{array}{r} 1 0 0 0 \\ - 0 2 3 4 \\ \hline \end{array}$$

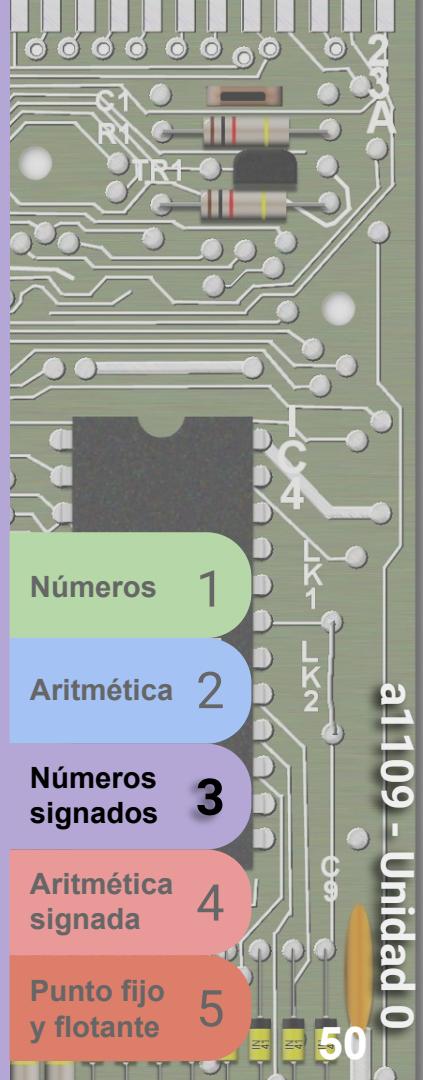
El complemento de un número se calcula como la **base (minuendo) menos el número (sustraendo)**.



Complemento de un número decimal

$$\begin{array}{r} & 1 & 0 & 0 & 0 \\ - & 0 & 2 & 3 & 4 \\ \hline & ? & & & \end{array}$$

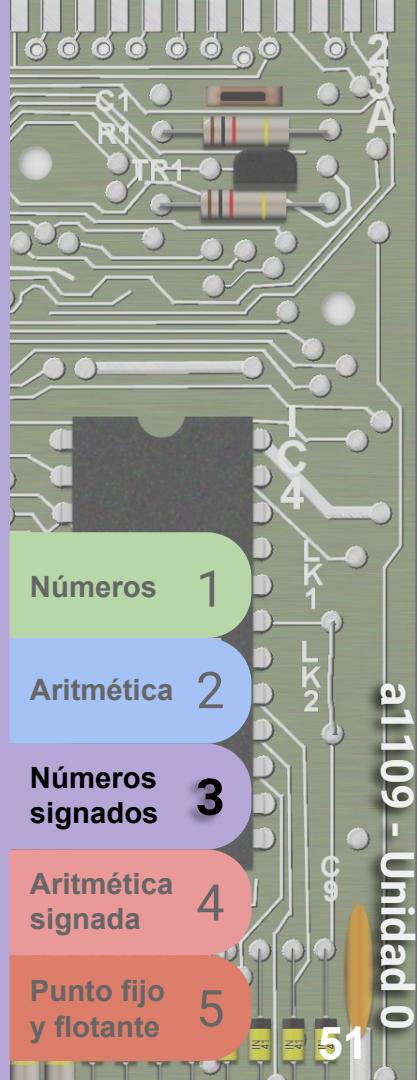
Pero ya en la primer columna tenemos problemas, ya que tenemos $0 - 4$. Tenemos que “pedir préstamos” pero las columnas siguientes también son 0, por ende seguimos hasta el primer 1 del minuendo.



Complemento de un número decimal

$$\begin{array}{r} \textcolor{red}{+} \quad 0 \quad 0 \quad 0 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline \quad \quad \quad \quad \textcolor{red}{?} \end{array}$$

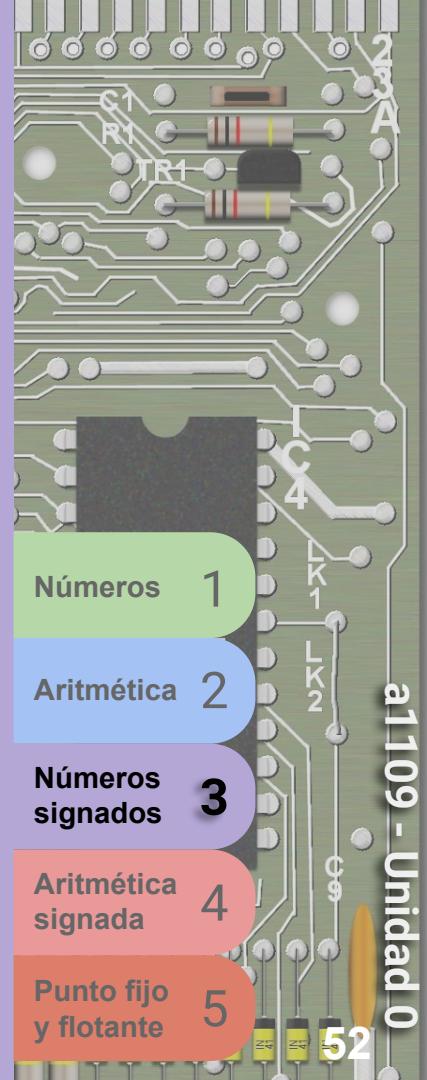
Este 1 pasa a ser 0 y viaja a la columna de la derecha.



Complemento de un número decimal

$$\begin{array}{r} \textcolor{red}{+} \quad 1 \quad 10 \quad 0 \quad 0 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline \quad \quad \quad \quad \textcolor{red}{?} \end{array}$$

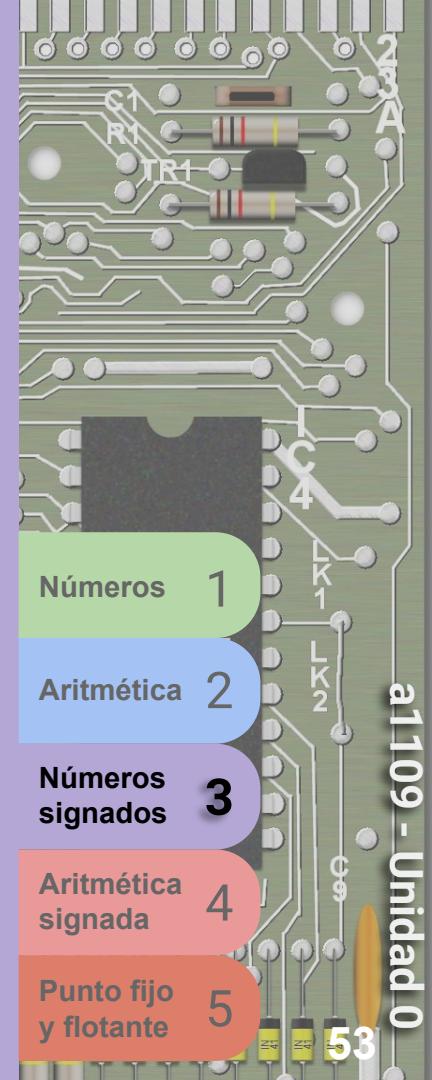
Ahora tenemos 10_{10} en la columna de las centenas. Pero le tenemos que seguir prestando hasta llegar a las unidades. Dado que tenemos 10 y prestamos 1, entonces nos quedamos con 9 en las centenas.



Complemento de un número decimal

$$\begin{array}{r} \textcolor{red}{+} \quad \textcolor{red}{9} \quad \textcolor{black}{10} \quad 0 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline \quad \quad \quad \quad \textcolor{red}{?} \end{array}$$

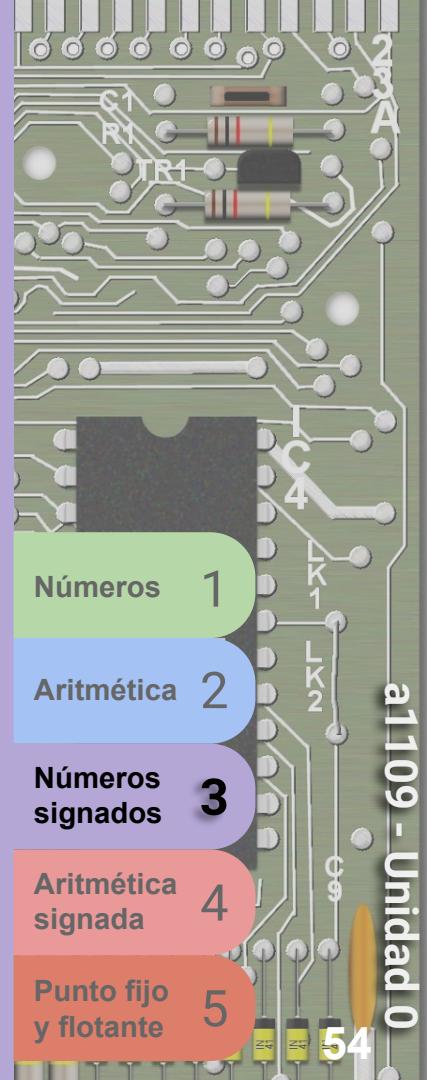
Repetimos lo mismo para las decenas, y pasamos uno para las unidades.



Complemento de un número decimal

$$\begin{array}{r} + \quad 9 \quad 9 \quad 10 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline \quad \quad \quad \quad ? \end{array}$$

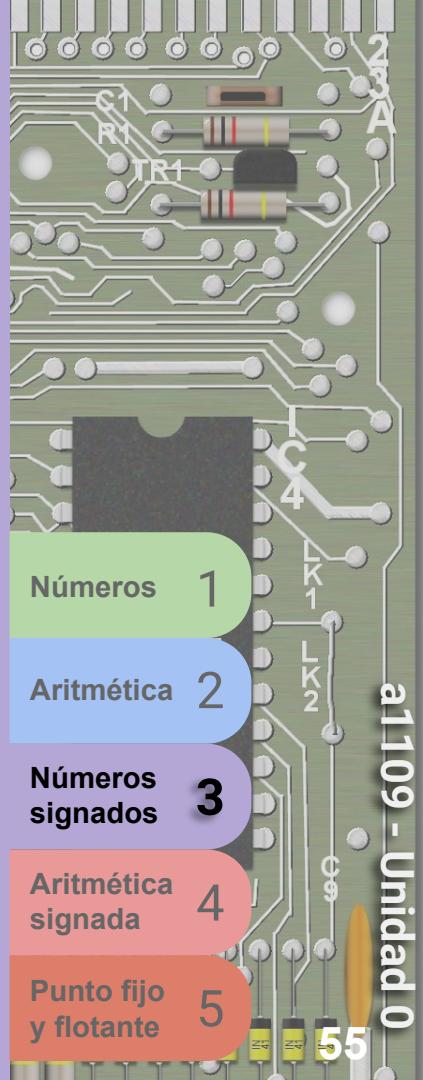
Y ahora si podemos realizar 10-4...



Complemento de un número decimal

$$\begin{array}{r} + \quad 9 \quad 9 \quad 10 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline \quad \quad \quad \quad 6 \end{array}$$

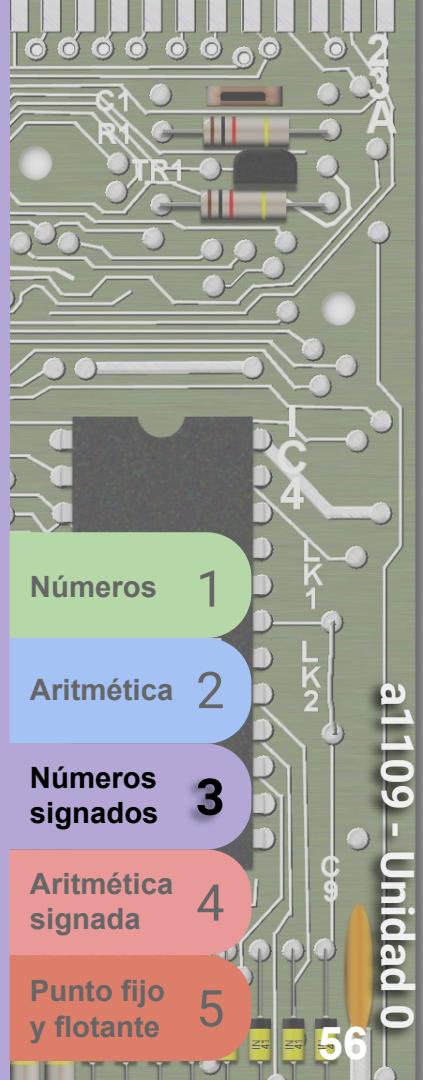
Y ahora si podemos realizar 10-4... lo cual resulta en 6



Complemento de un número decimal

$$\begin{array}{r} 0 \quad 9 \quad 9 \quad 10 \\ - \quad 0 \quad 2 \quad 3 \quad 4 \\ \hline 6 \end{array}$$

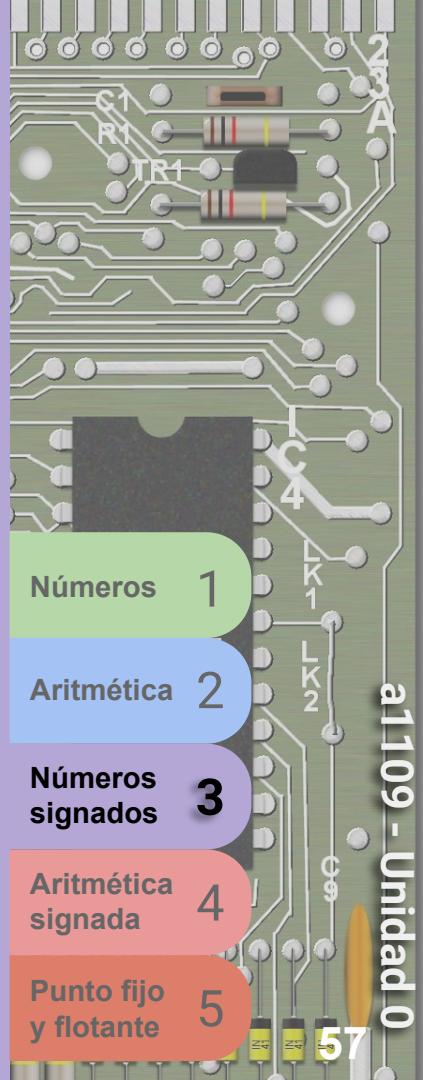
Continuamos restando.. pero notemos que todas las columnas faltantes ahora tienen siempre el dígito más alto para este sistema decimal. Esto resuelve el problema de los préstamos y la operación se puede resolver sin más pedidos.



Complemento de un número decimal

$$\begin{array}{r} & 0 & 9 & 9 & 10 \\ - & 0 & 2 & 3 & 4 \\ \hline & 0 & 7 & 6 & 6 \end{array}$$

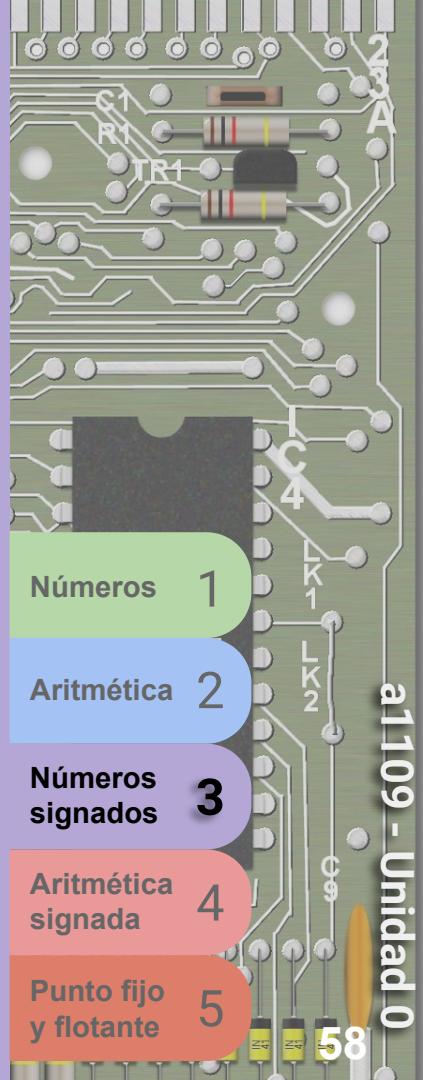
El resultado es 766, por ende el complemento a la base de 234_{10} es 766. Nótese que si sumamos el número original más su complemento a la base obtenemos nuevamente la potencia más cercana de la base. Podemos informalmente decir que obtenemos nuevamente la base. Si nos limitamos a 3 dígitos podemos decir que obtenemos 0.



Complemento de un número decimal

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

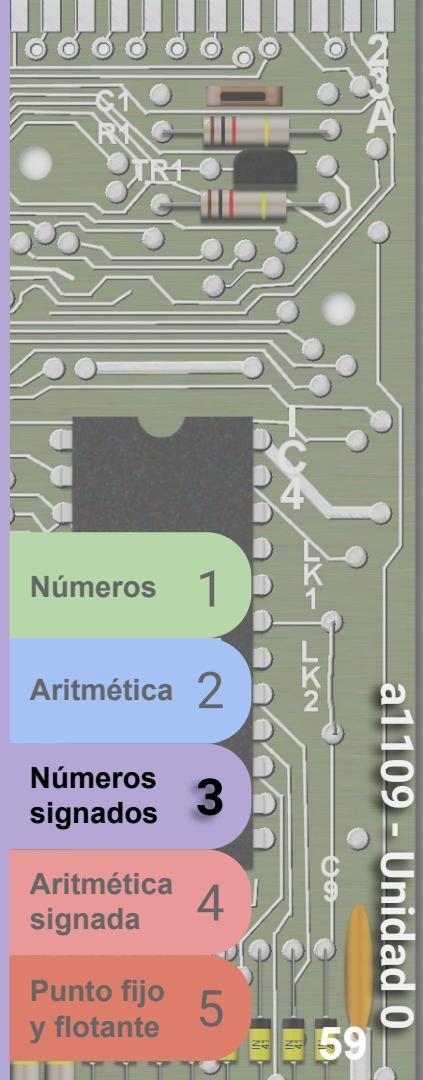
Podemos ver que para cualquier número de 3 dígitos XYZ (excepto 000) vamos a tener problemas restando en algún punto, ya que cada columna que forma la base está compuesta de ceros (excepto el dígito más significativo). Siempre va a haber que pedir préstamos en algún momento.



Complemento a la base menos uno

$$\begin{array}{r} & 9 & 9 & 9 \\ - & X & Y & Z \\ \hline \end{array}$$

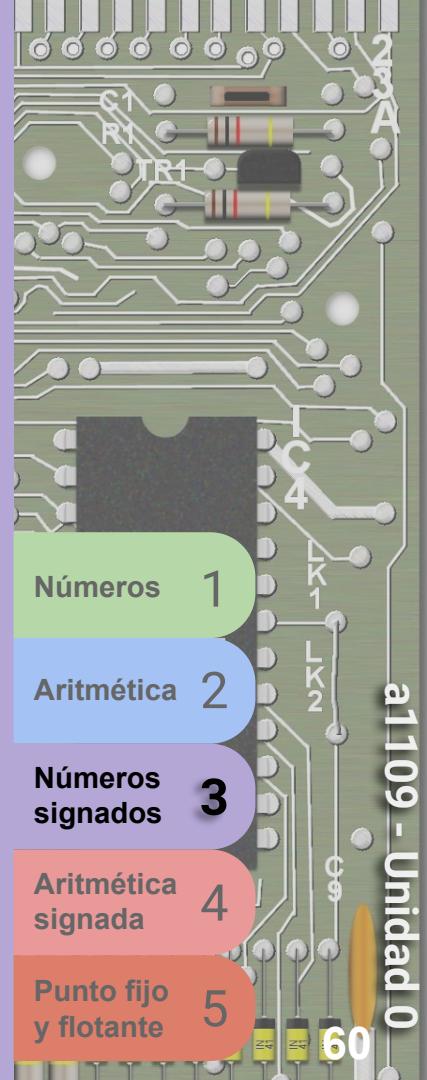
Podemos tomar como valor la base menos uno. Esto nos asegura que cada columna tendrá el dígito de mayor valor. Esto evita todo préstamo. Sin embargo el resultado obtenido no será el complemento a la base del número XYZ, sino el complemento a la base MENOS uno. Solucionamos esto sumando uno.



Complemento a la base menos uno

$$\begin{array}{r} & 9 & 9 & 9 \\ - & 2 & 3 & 4 \\ \hline \end{array}$$

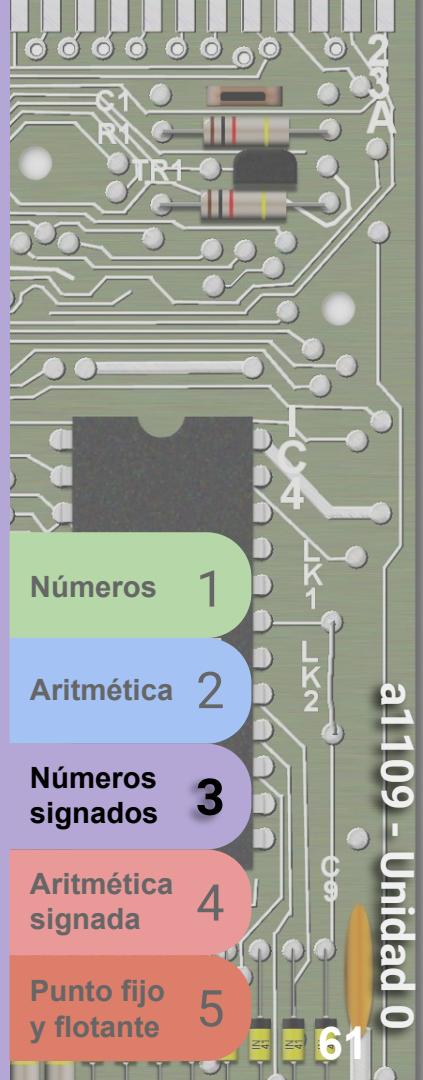
Veamos nuevamente el caso de 234, vemos que en ningún caso hay que pedir préstamos.



Complemento a la base menos uno

$$\begin{array}{r} & 9 & 9 & 9 \\ - & 2 & 3 & 4 \\ \hline & 7 & 6 & 5 \end{array}$$

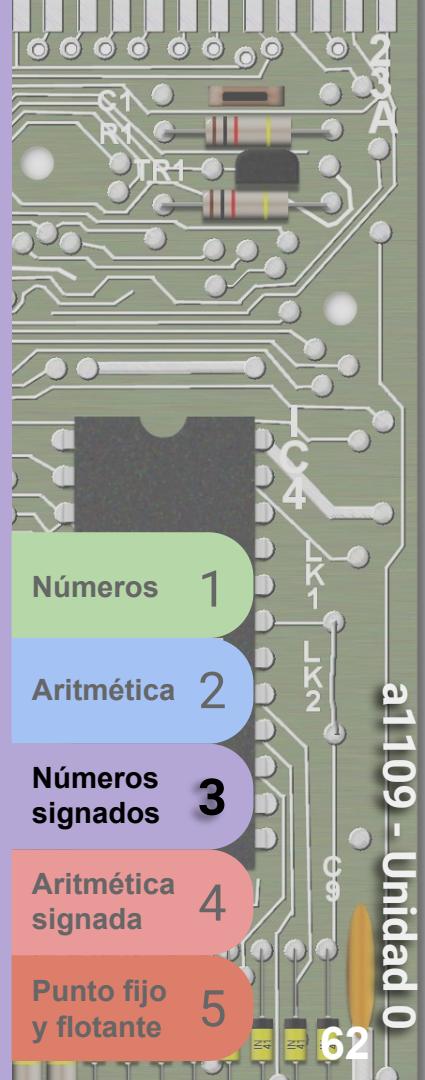
Pero el resultado obtenido no es 766 sino 765.
Cuando se calcula el complemento a la base menos uno debe sumarse el uno faltante al resultado para obtener el complemento a la base.



Complemento a la base menos uno

$$\begin{array}{r} & 9 & 9 & 9 \\ - & 2 & 3 & 4 \\ \hline & 7 & 6 & 5 \\ + & 0 & 0 & 1 \\ \hline & 7 & 6 & 6 \end{array}$$

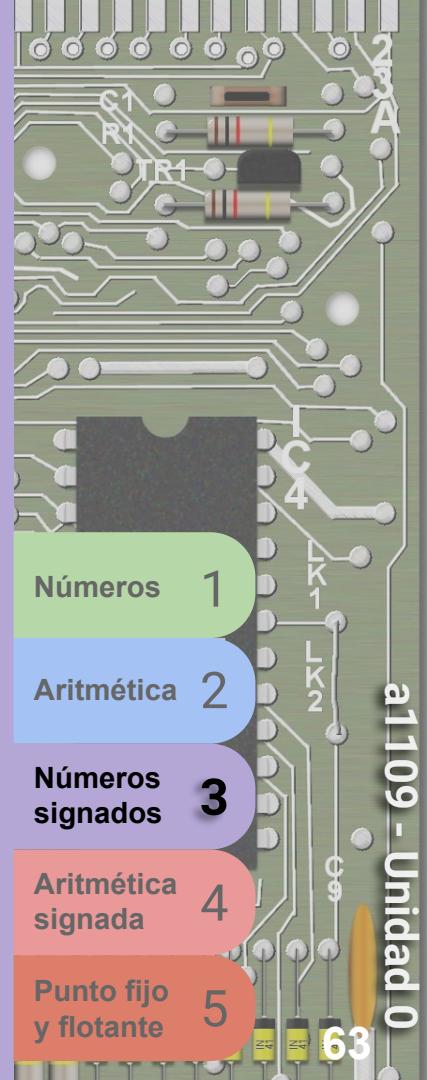
Vemos que luego de sumar 1 obtenemos el complemento a la base.



Complemento a la base en binario

El concepto de complemento a la base puede ser aplicado a cualquier base, no solo a base 10. Es especialmente útil en binario ya que podemos utilizar el complemento a la base para representar números signados o para realizar operaciones de resta.

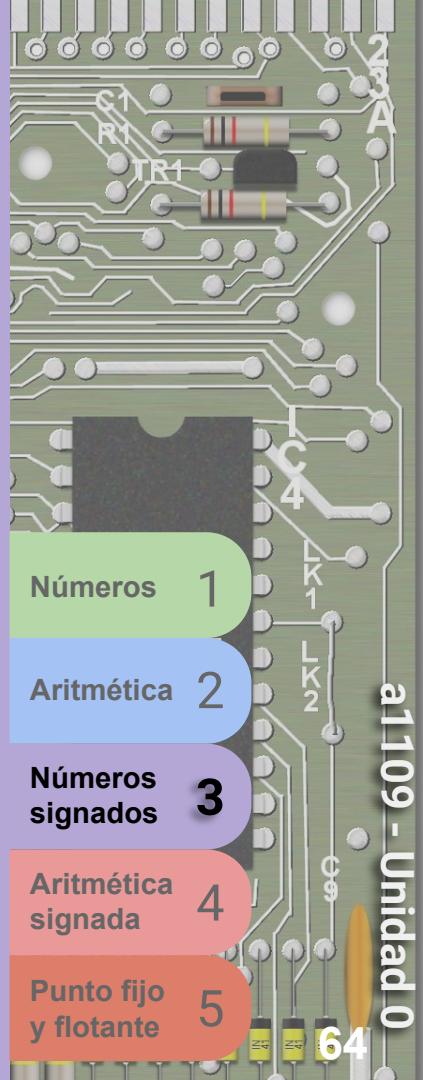
Continuando con nuestro universo de números de 3 bits , tomemos por ejemplo el número $011_2 = 3_{10}$. Al ser un número de 3 dígitos su base más cercana se calcula como $2^3 = 8_{10} = 1000_2$. Por ende sustraemos $1000_2 - 011_2$ para obtener el complemento a la base.



Complemento a la base en binario

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

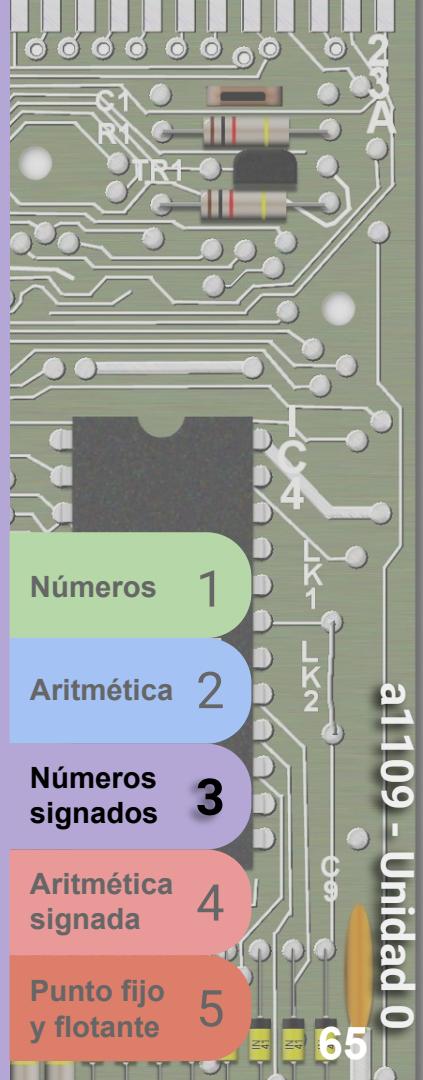
A priori podemos ver que la base para 3 dígitos en binario se escribe igual que en decimal (1000). Obviamente representan números distintos pero los dígitos utilizados son los mismos. Podemos hacer la resta pero vemos que ya tenemos problemas en la columna menos significativa.



Complemento a la base en binario

$$\begin{array}{r} + \quad 1 \quad 10 \quad 0 \quad 0 \\ - \quad 0 \quad 0 \quad 1 \quad 1 \\ \hline \quad \quad \quad \quad ? \end{array}$$

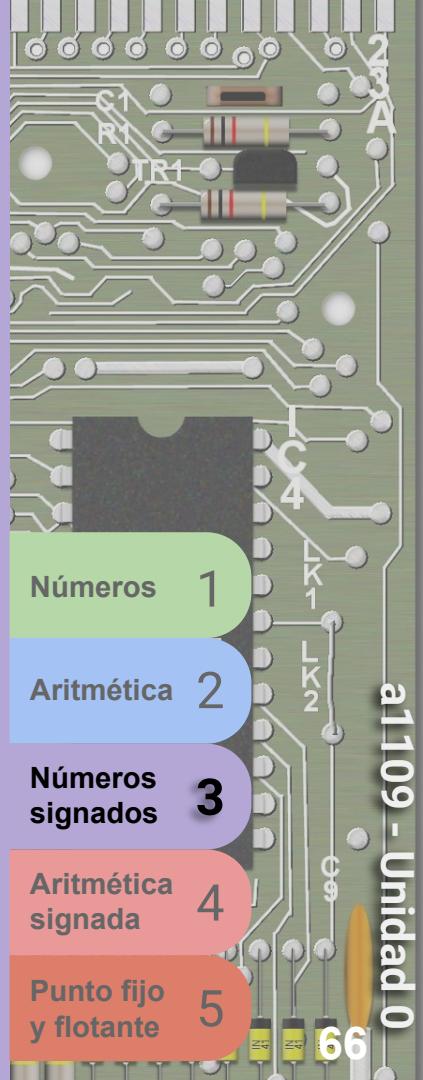
Tenemos que nuevamente llegar al 1 de la columna más significativa para pedir el préstamo. Este 1 pasa a 0 y se traslada a la columna siguiente. Vemos que tenemos 10 en esta columna que representa el 2. Pero debemos seguir prestando.



Complemento a la base en binario

$$\begin{array}{r} \textcolor{red}{1} \quad \textcolor{red}{1} \quad \textcolor{black}{10} \quad 0 \\ - \quad 0 \quad 0 \quad 1 \quad 1 \\ \hline \quad \quad \quad \quad \textcolor{red}{?} \end{array}$$

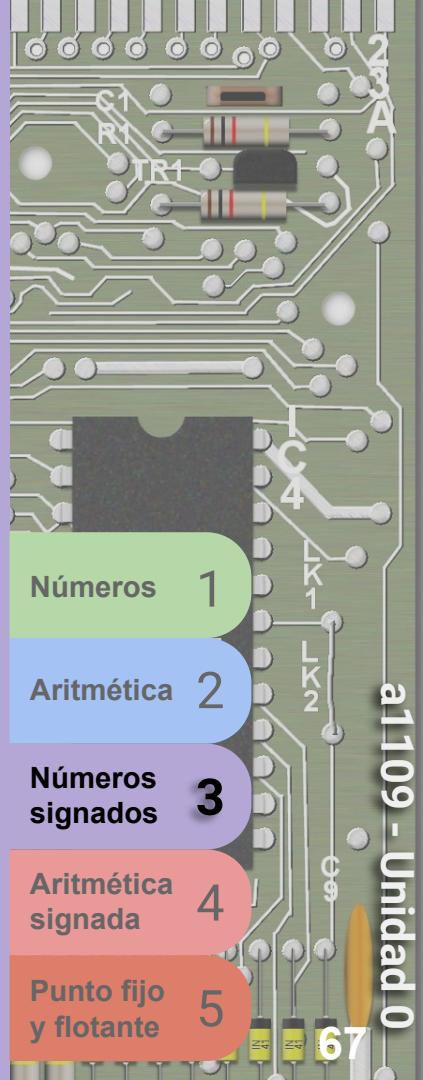
Notemos que la tercer columna (desde la derecha) que tenía 10 ahora tiene 01, o sea 1. Si lo pensamos en decimal, la columna tenía 2, y le quitamos uno, así que ahora pasó a tener 1. Ahora la segunda columna (desde la derecha) tiene 10, pero de nuevo tiene que seguir prestando...



Complemento a la base en binario

$$\begin{array}{r} \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} \\ - & 0 & 0 & 1 \\ \hline & & & \textcolor{red}{1} \end{array}$$

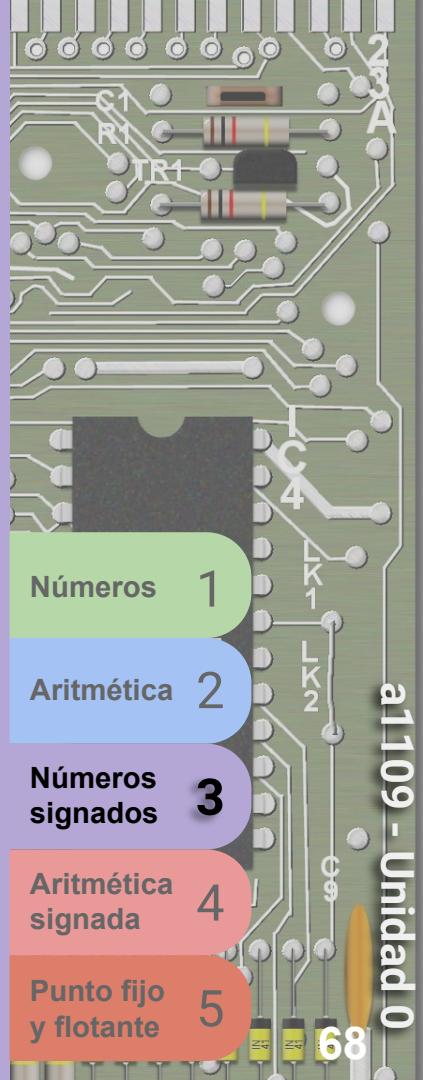
Ahora si podemos realizar la resta en la columna menos significativa. Luego el resto de la resta es trivial...



Complemento a la base en binario

$$\begin{array}{r} 0 \quad 1 \quad 1 \quad 10 \\ - \quad 0 \quad 0 \quad 1 \quad 1 \\ \hline 1 \quad 0 \quad 1 \end{array}$$

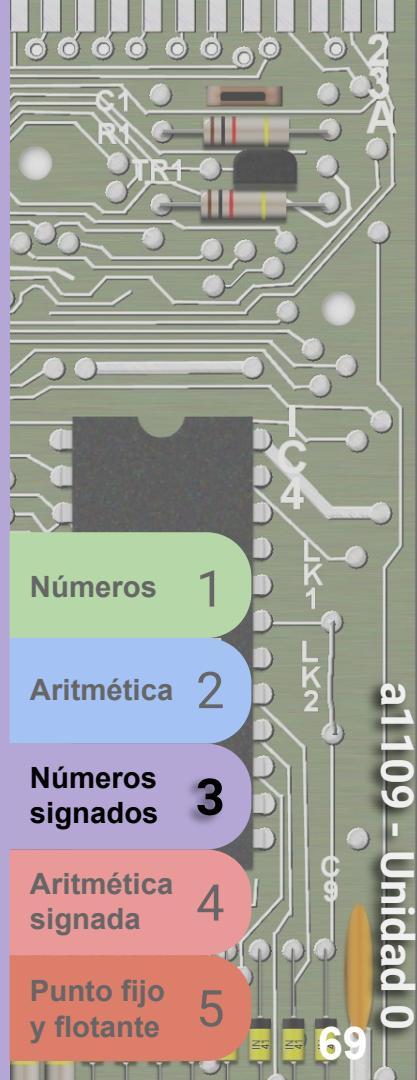
Vemos que el resultado obtenido es el número binario 101. Esto quiere decir que el complemento a la base de 011 (3) es 101 (5), lo cual tiene sentido ya que si los sumamos obtenemos 1000 (8). Pero esto es asumiendo que hablamos de números sin signo.



Complemento a la base menos uno en binario

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

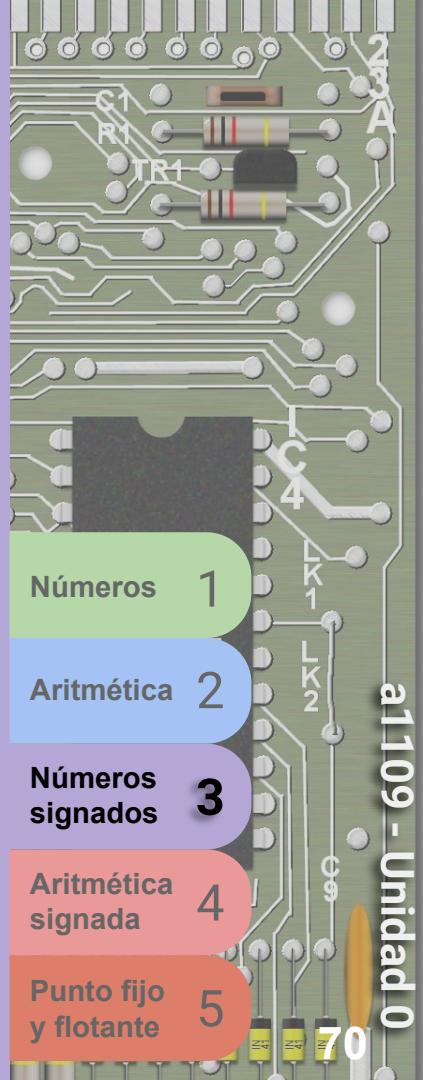
El mismo procedimiento de utilizar el complemento a la base menos uno es también válido en binario. Realizamos la resta utilizando la base menos 1 como minuendo



Complemento a la base menos uno en binario

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

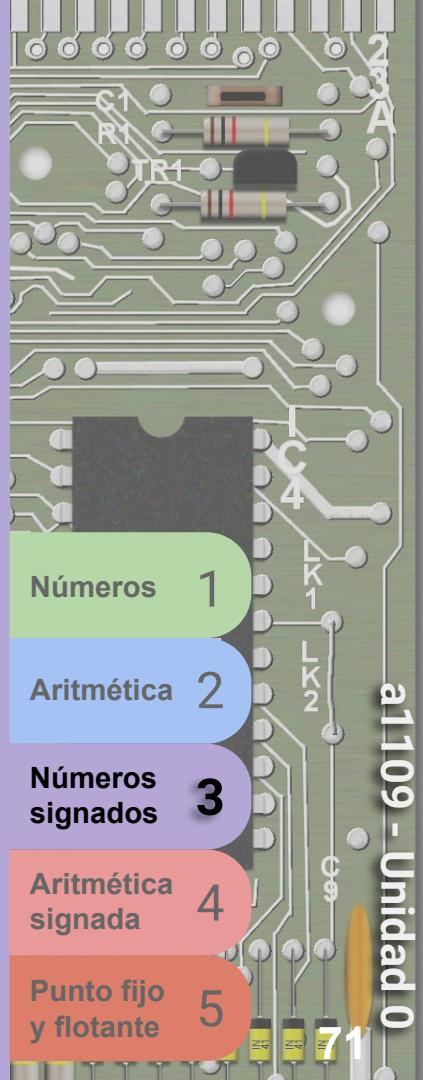
La resta ahora es trivial... y si miramos el resultado del complemento a la base menos uno, en cada columna tenemos el dígito opuesto del bit original. Esto se debe a que solo tenemos 2 dígitos. Por ende en binario es muy sencillo calcularlo.



Complemento a la base menos uno en binario

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

Volvemos a sumar 1 al complemento a la base menos uno y obtenemos el complemento a la base. Como vemos en binario esta secuencia es mucho más simple que la original.

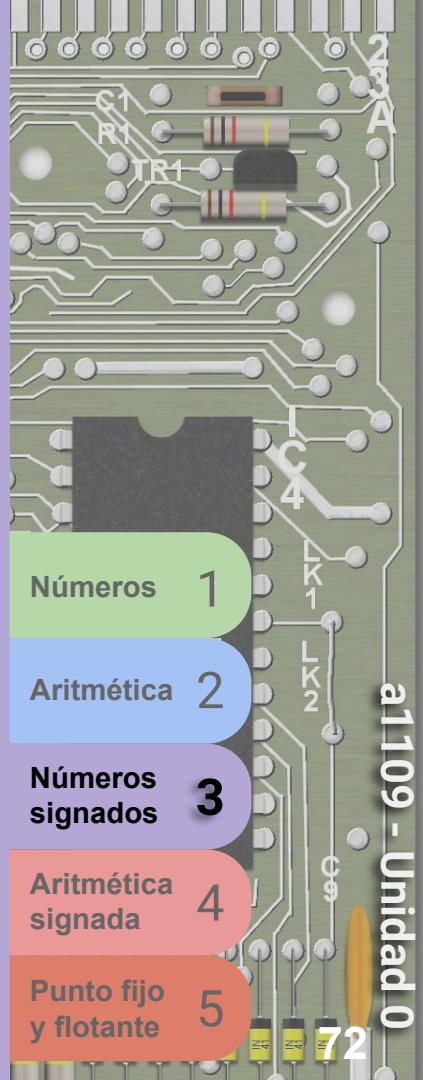


3 bits Valor

000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Volvemos a nuestro universo de 3 bits. Quitamos los valores de aquellos números cuyo bit más significativo sea 1.



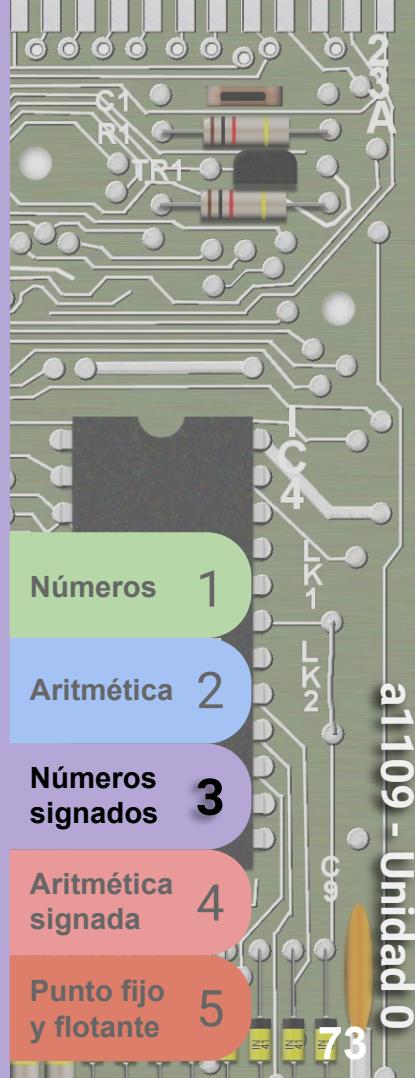
3 bits Valor

000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Binario	Decimal		
0 0 1	1		
- 0 1 0	2		

Planteamos ahora la operación 1 menos 2, o sea 001 menos 010.

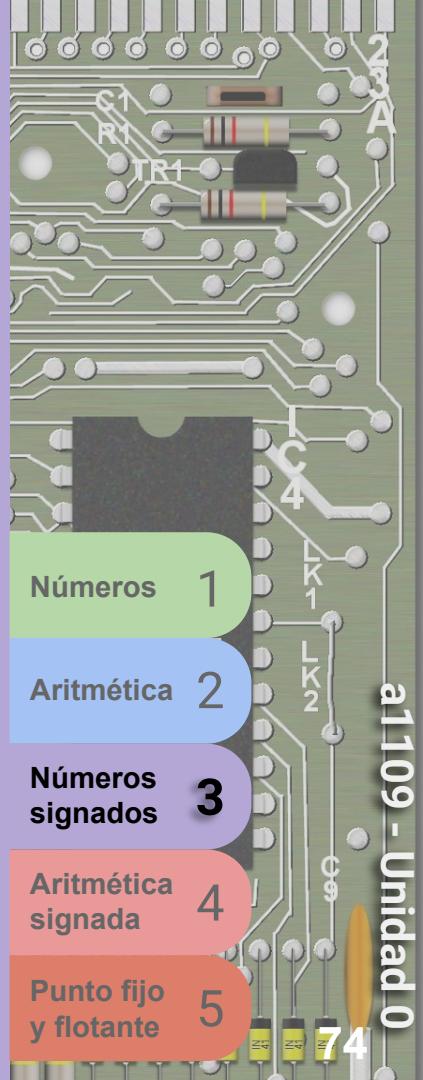


3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Binario	Decimal
0 0 1	1
- 0 1 0	2
<hr/>	
	-1

Vemos que en decimal el resultado es -1. Veamos qué pasa cuando hacemos la resta en binario.

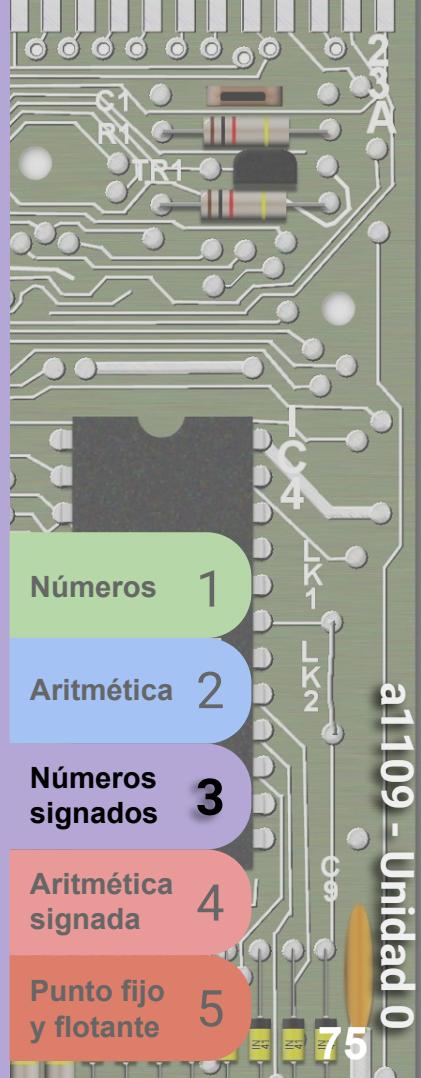


3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Binario	Decimal
1	
0 10 1 1	
- 0 1 0 2	
1 1 -1	

Notamos que en la segunda columna teníamos originalmente 0-1, y tuvimos que pedir al bit más significativo, que al ser cero dejó un préstamo (borrow) en esa columna.



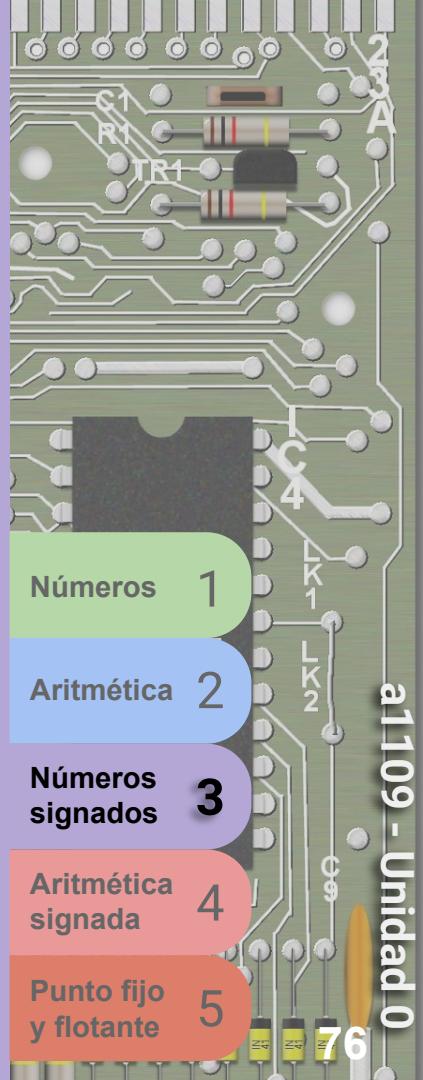
3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Binario	Decimal
0 10 1	1
- 0 1 0	2
1 1 1	-1

Este resultado (111) lo tenemos en 3 bits.

Anteriormente su valor era 7 (cuando teníamos solo números sin signo). Cambiemos el sistema ahora para que este valor (111) sea -1.

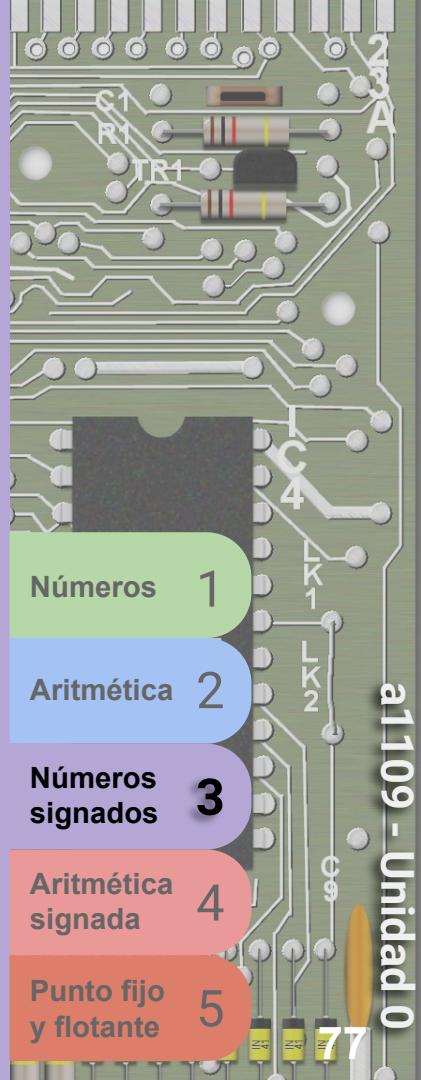


3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	
110	
111	

Universo de 3 bits signado

Binario	Decimal
1	
0 0 10 1 1	
0 0 1 0 2	
1 1 1 1 -1	

Si hubiésemos continuado (en el caso de un número de 4 bits) vemos que el borrow se sigue arrastrando hacia la columna siguiente. En el caso de 4 bits el 1111 es -1. En el caso de 5 bits será 11111 y así...

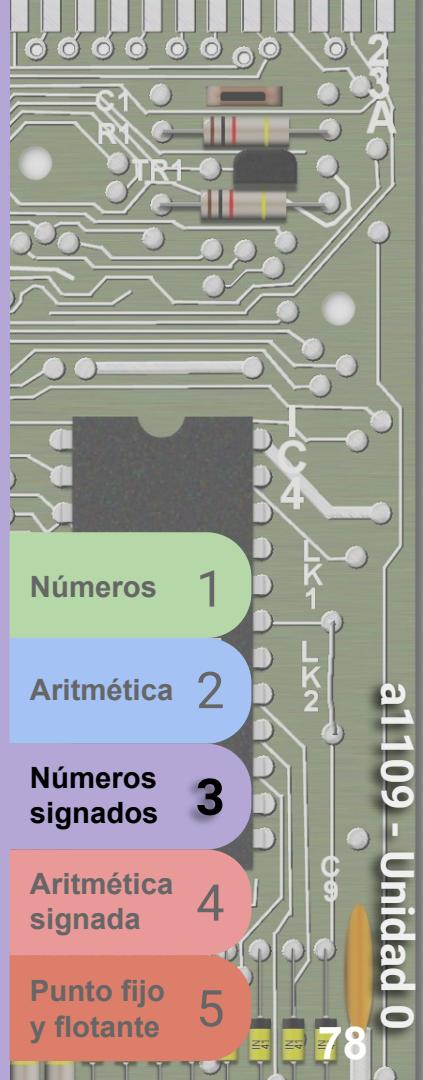


3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	
110	
111	-1

Universo de 3 bits signado

Binario	Decimal
0 10 1	1
- 0 1 0	2
1 1 1	-1

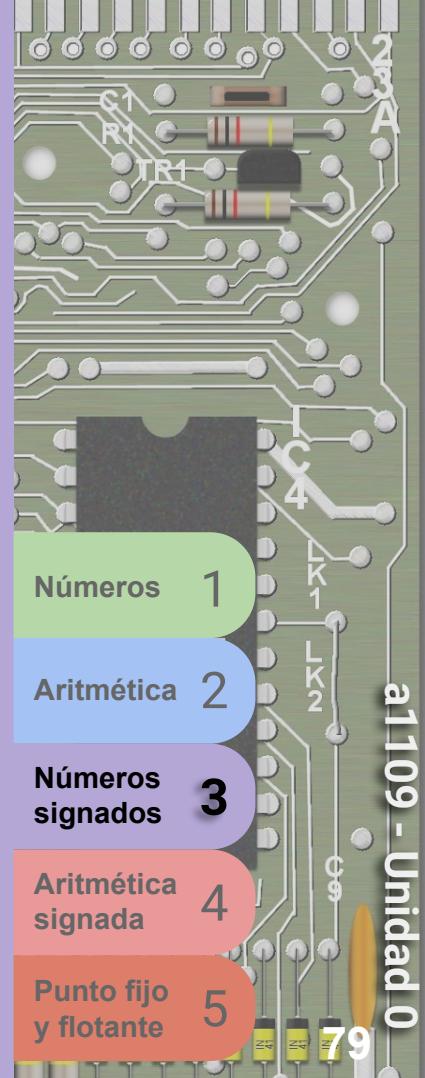
Acabamos de inventar el primer número negativo en 3 bits. Notemos que 111_2 es el complemento a la base del número 001_2 . Esta es una propiedad muy útil. Calculemos entonces el complemento del resto.



Universo de 3 bits signado

3 bits	Valor
000	0
001	1
010	2
011	3
100	
101	-3
110	-2
111	-1

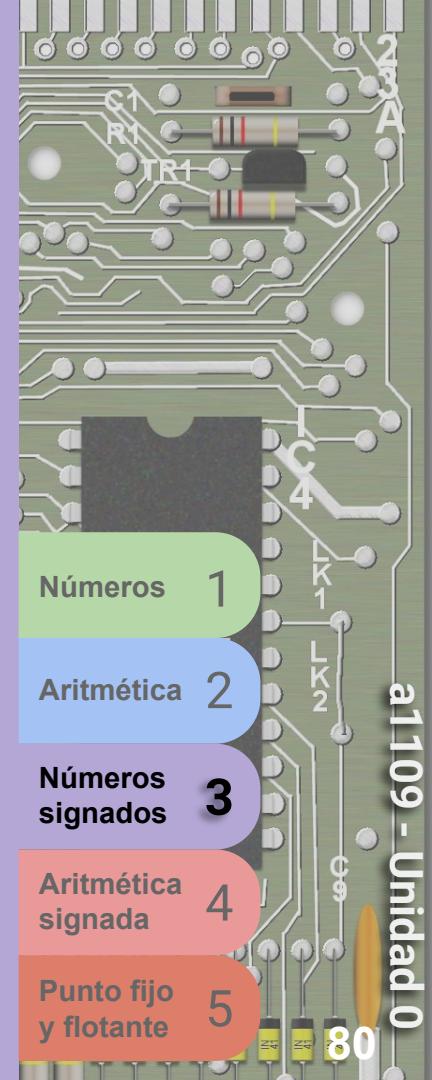
Vemos que el complemento de 010_2 es 110_2 y el de 011_2 es 101_2 . En el caso de 000, su complemento es el mismo. Nos queda el caso de 100_2 cuyo complemento también es el mismo.



Universo de 3 bits signado

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

A este caso en particular le asignamos el siguiente número negativo... o sea -4. Vemos que nuestro universo de 3 bits ahora representa números positivos (con cero) y números negativos.



Universo de 3 bits signado

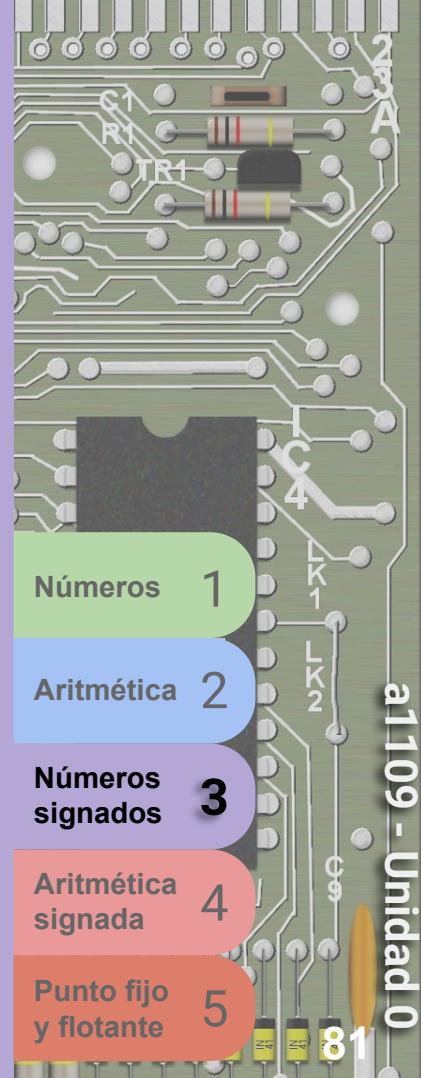
3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Hemos creado ahora un nuevo universo de números. Estos números son ahora signados. Nótese que la cantidad de elementos que se pueden representar con 3 bits se mantiene en 8 ($B^N = 2^3 = 8$). Sin embargo ya no representamos desde 0 hasta 7.

Ahora representamos desde -4 hasta 3. Esto podemos generalizarlo como:

Menor elemento: $-(B^{N-1})$, en este caso $-(2^2) = -4$

Mayor elemento: $B^{N-1} - 1$, en este caso $2^2 - 1 = 3$

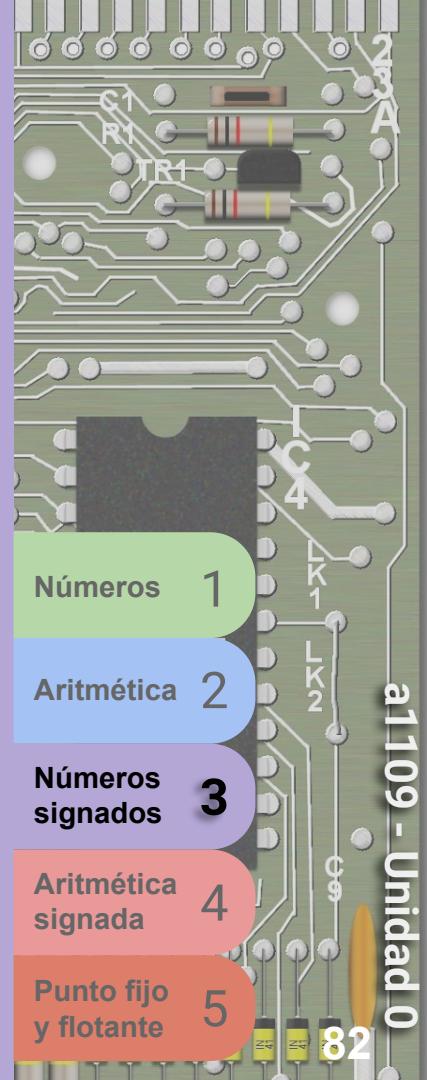


Universo de 3 bits signado

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Notemos que la cantidad de elementos negativos equivale a la cantidad de elementos positivos incluido el cero. También vemos que si contamos desde cero sumando de a uno, cuando nos pasamos del valor máximo (3) continuamos con el valor mínimo (-4).

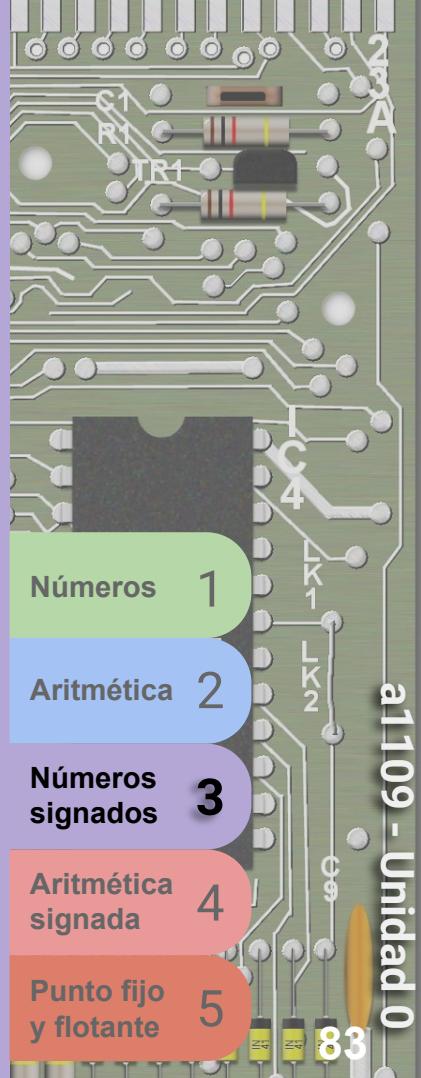
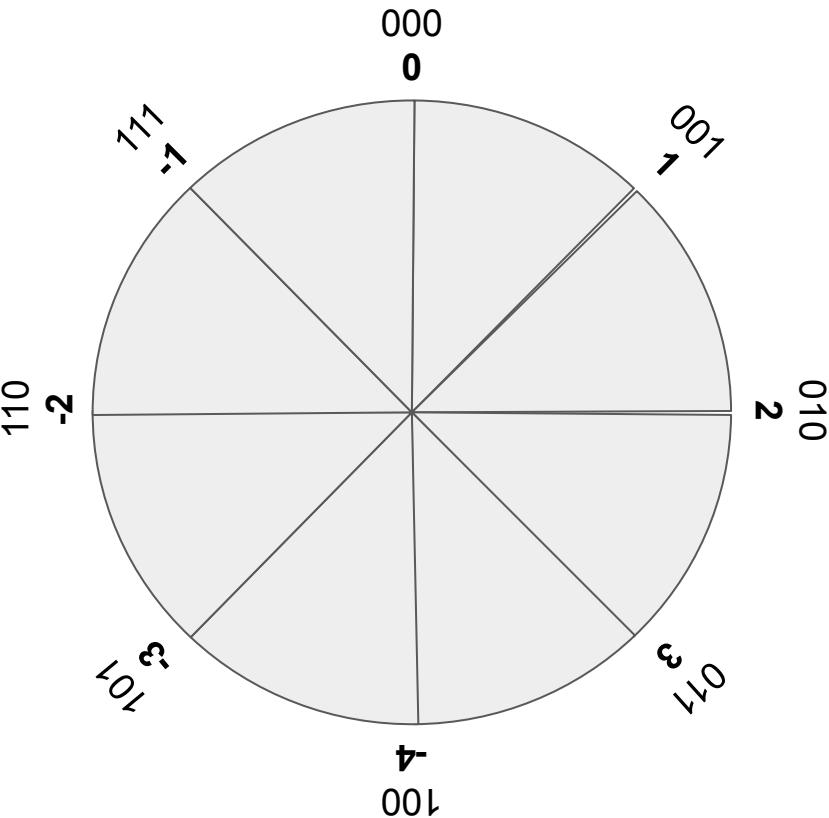
Veamos que los números negativos tienen todos sus bits más significativos en 1. A este bit lo llamamos bit de signo. Si es 0 el número es positivo (o cero), si es 1 el número es negativo.



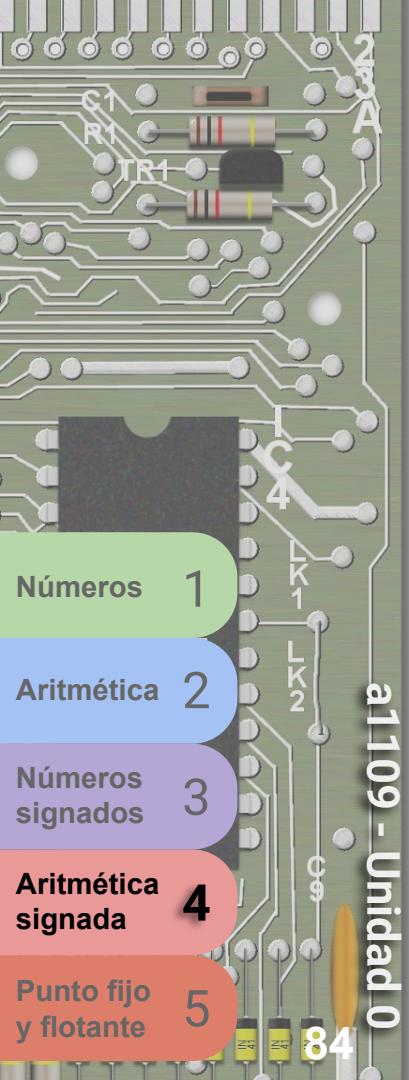
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Universo de 3 bits signado



Aritmética signada (sumas, restas y comparaciones)



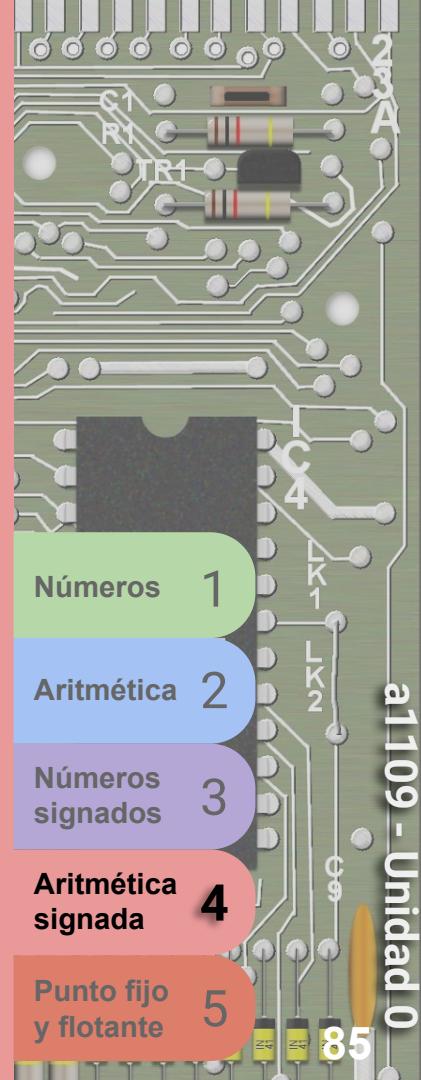
3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Si tenemos un mundo donde solo existen números de 3 bits, entonces podemos representar 2^3 elementos distintos. Esto podemos generalizarlo para todas las bases B y todas las cantidades de dígitos N como

$$B^N$$

Pero si los 3 bits son signados, el **menor** ahora es $-(B^{N-1})$, o sea $-(2^2) = -4$, y el **mayor** será $B^{N-1} - 1$, o sea $2^2 - 1 = 3$.



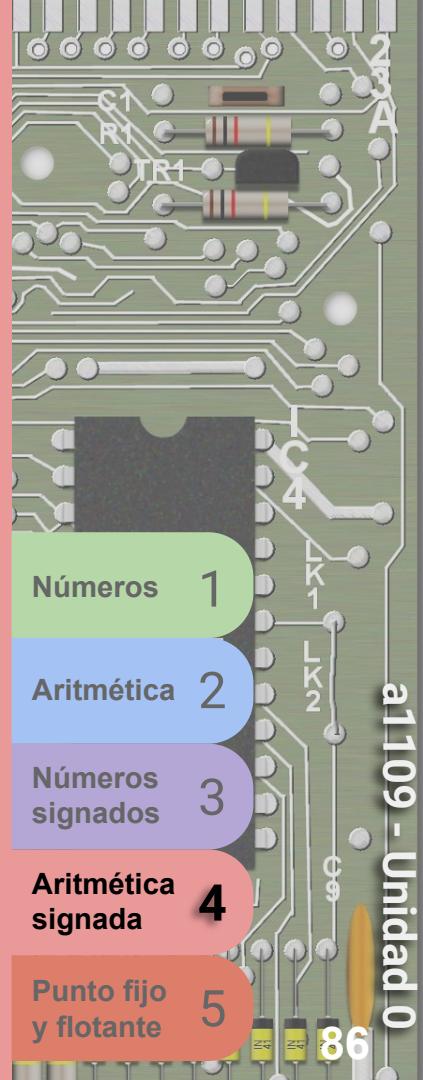
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario Decimal

$$\begin{array}{r} & 0 & 1 & 1 & | & 3 \\ + & 1 & 1 & 1 & | & -1 \\ \hline \end{array}$$



Ahora que tenemos números signados, podemos sumar números de distintos signos, por ejemplo $3 + (-1)$... que debería resultar igual que $3-1$. Pero esto lo hacemos sumando bits.

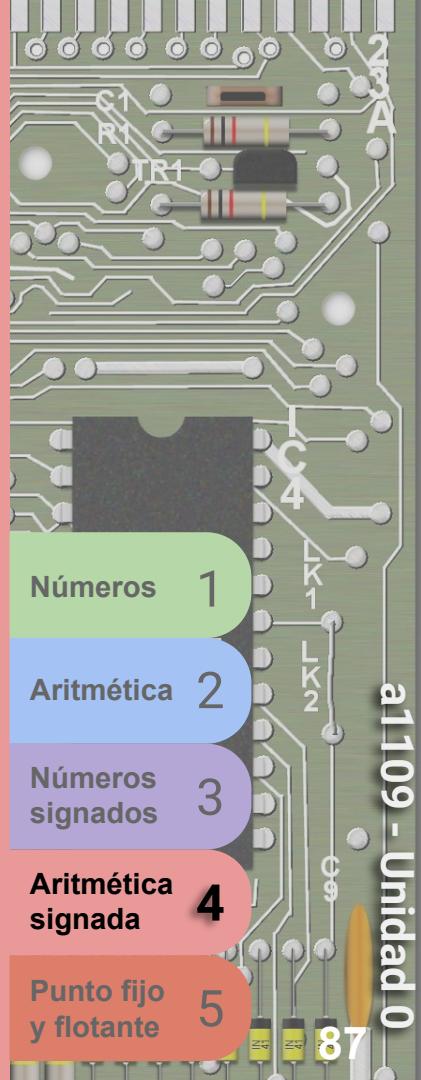
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
0 1 1 3	
+ 1 1 1 -1	
<hr/>	
	2

En decimal sabemos que $3+(-1)=3-1=2$. Para realizarlo en decimal resolvemos la regla de los signos y restamos en vez de sumar. En binario simplemente sumamos...

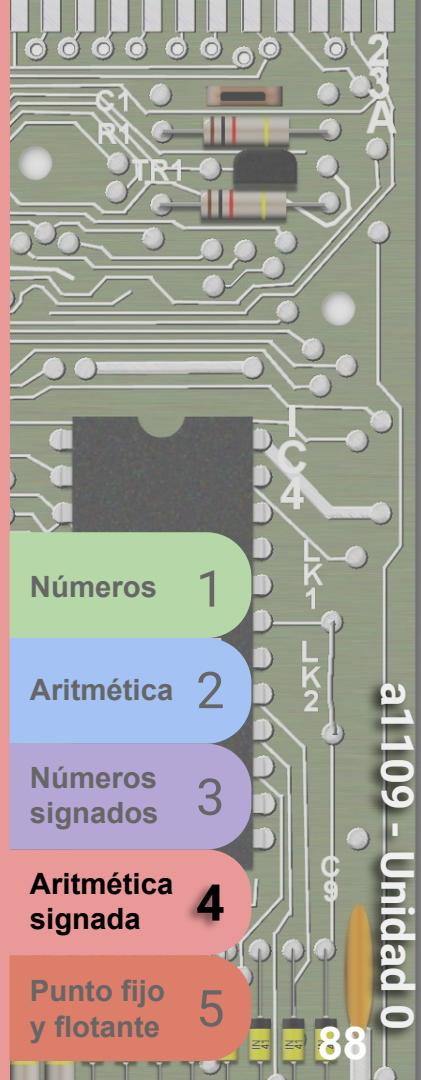


3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
1 1	
0 1 1	3
+ 1 1 1	-1
1 0 1 0 2	

Vemos que al sumar aparecen acarreos intermedios y un acarreo final. Pero notamos que el resultado efectivamente es $010_2 = 2$. Este carry final lo podemos despreciar.



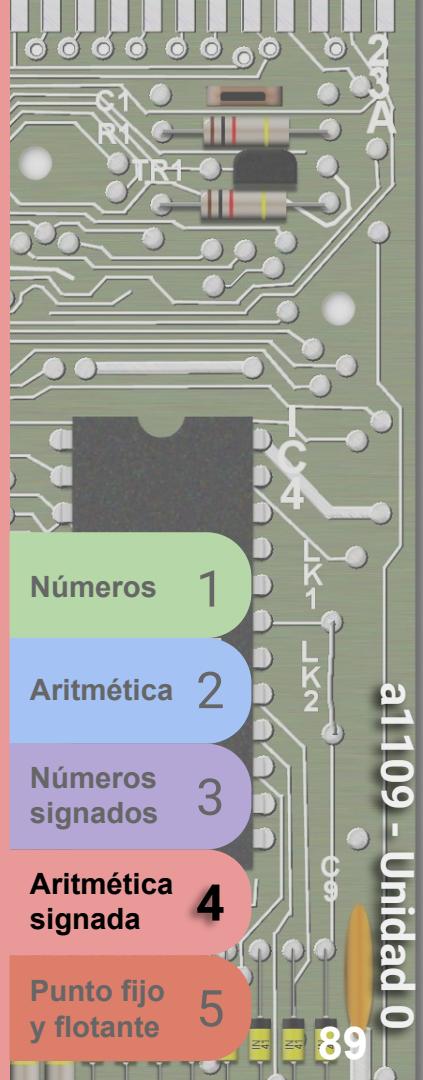
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
0 1 1 3	
+ 0 1 0 2	
<hr/>	
	5

Luego si planteamos $3+2$, podemos ver a priori que el resultado 5 no existe en este universo de 3 bits signados. Veamos que ocurre con la suma...

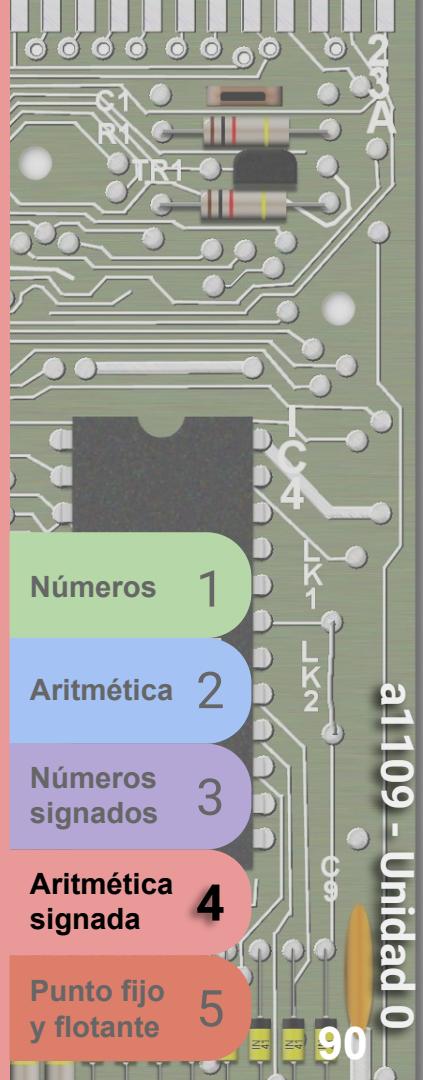


3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
1	
0 1 1	3
+ 0 1 0	2
1 0 1	5

Vemos que no hubo acarreo (salvo el interno), pero el resultado 101_2 no es 5 en este universo sino -3.
¡¡Evidentemente este no es el valor esperado!!!



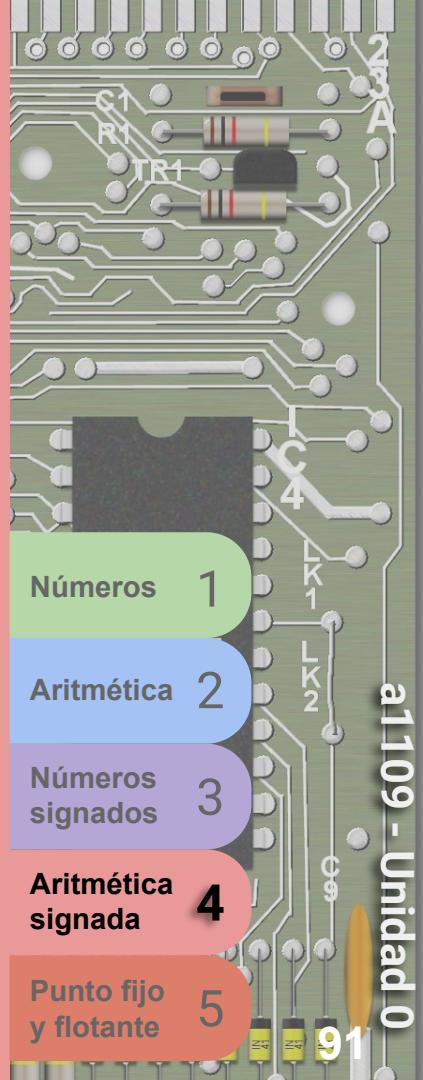
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
1	
0 1 1	3
+ 0 1 0	2
1 0 1	5

Sin embargo notemos que los bits de signo de los operandos son 0 (o sea positivos) y el resultado es un número cuyo bit de signo es 1 (o sea negativo). Pensemos que es lo que ocurre...



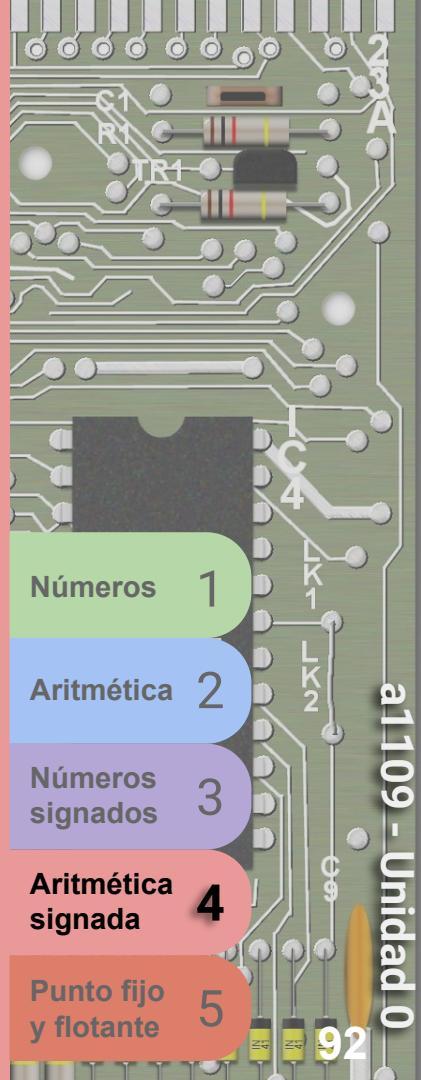
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
1	0	1	3
+	0	1	2
	1	0	5

Estamos sumando un número positivo con otro positivo.. por ende esperamos como resultado un número positivo.. ¡¡¡pero obtuvimos un número negativo!!!



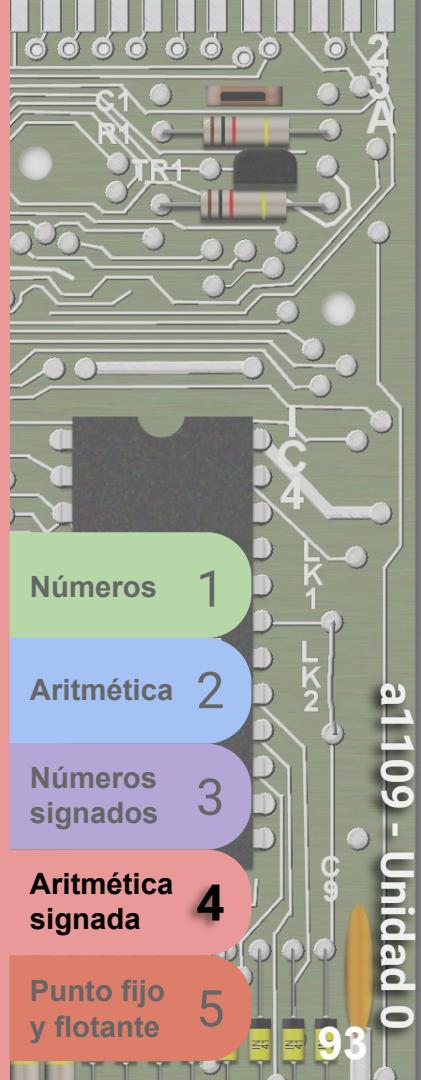
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
1			
0	1	1	3
+	0	1	2
	1	0	1
			5

Cuando esto ocurre en la suma lo llamamos desborde (**overflow**). Quiere decir que nos pasamos del límite de los números positivos (en este caso) y rebalsamos a los negativos.



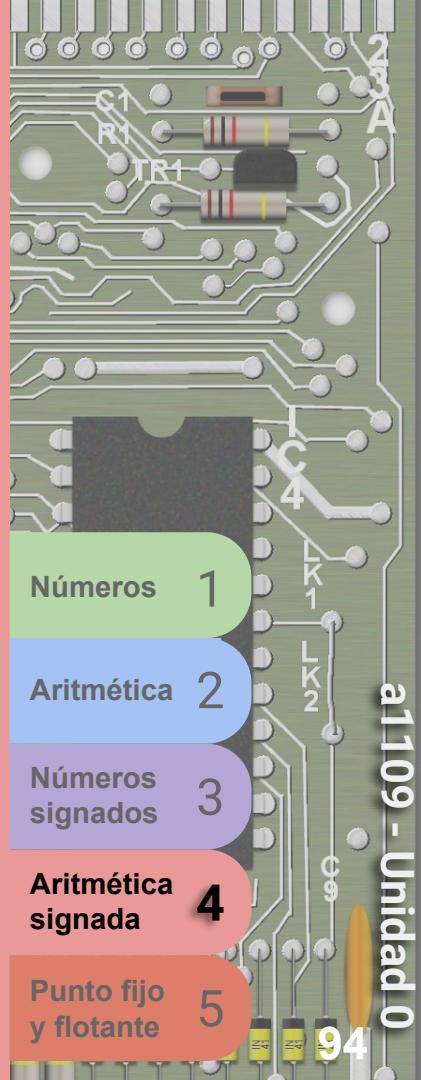
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario			Decimal	
1				
0 1 1				3
+	0 1 0			2
1 0 1			5	

Si obtenemos overflow en una suma (obviamente solo con números signados) el resultado de la misma no se puede representar con esta cantidad de bits.

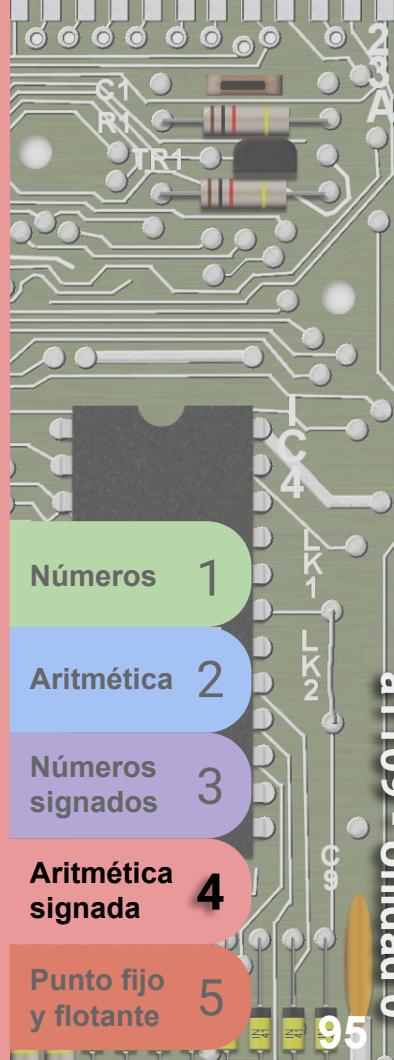
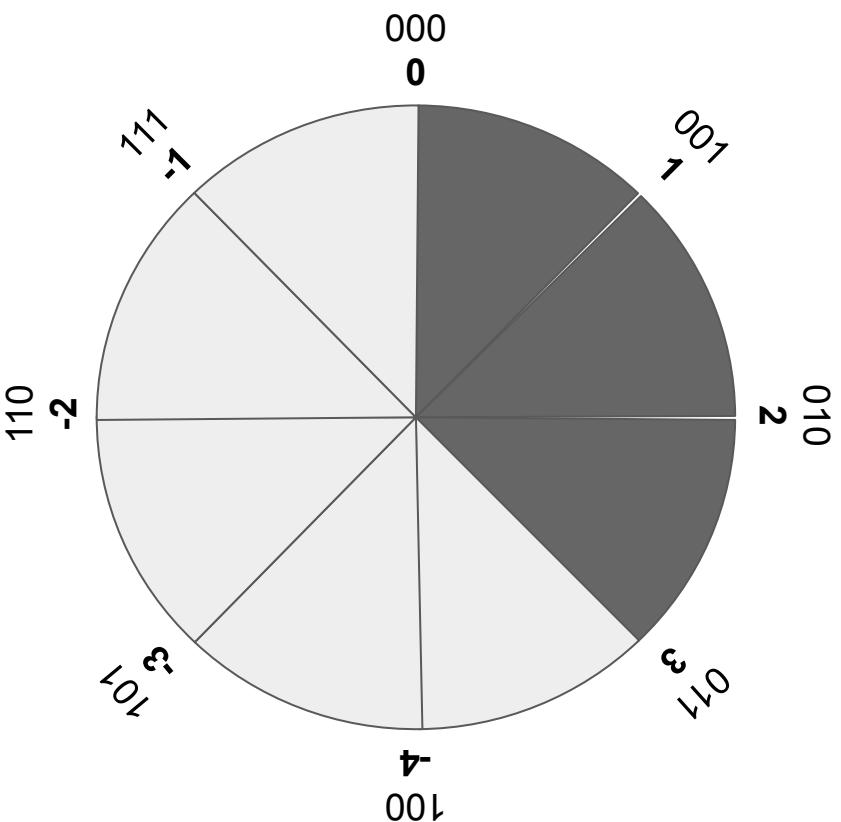


3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Tomamos 3 segmentos

Un mundo de 3 bits signados

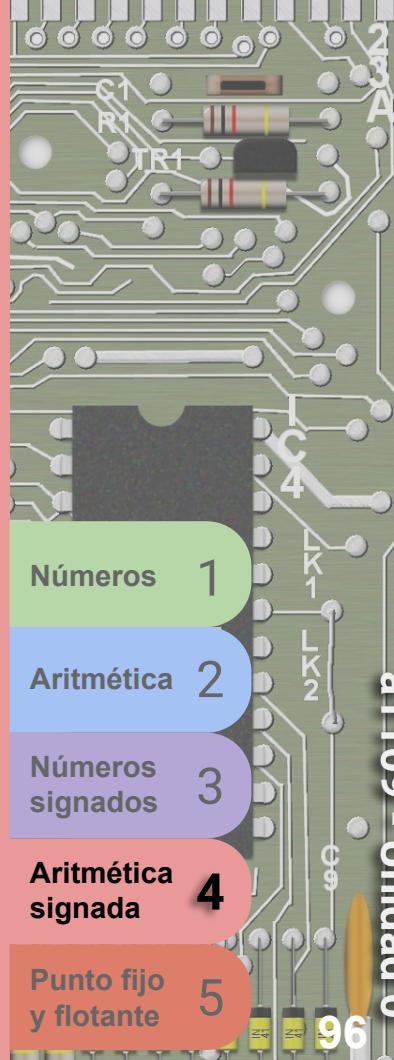
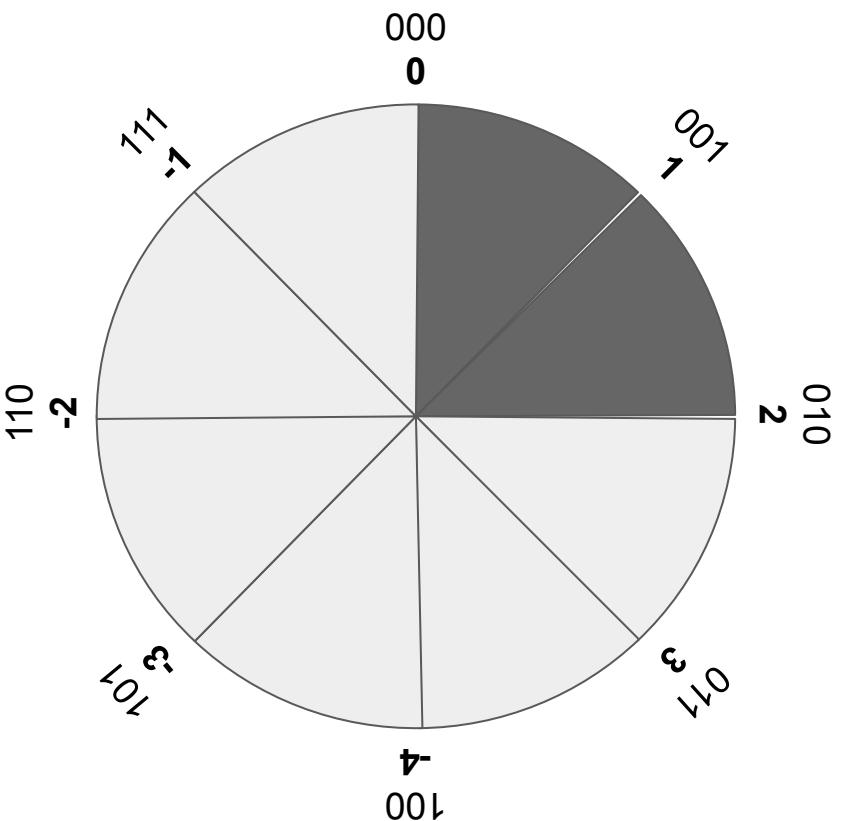


3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Tomamos 2 segmentos

Un mundo de 3 bits signados

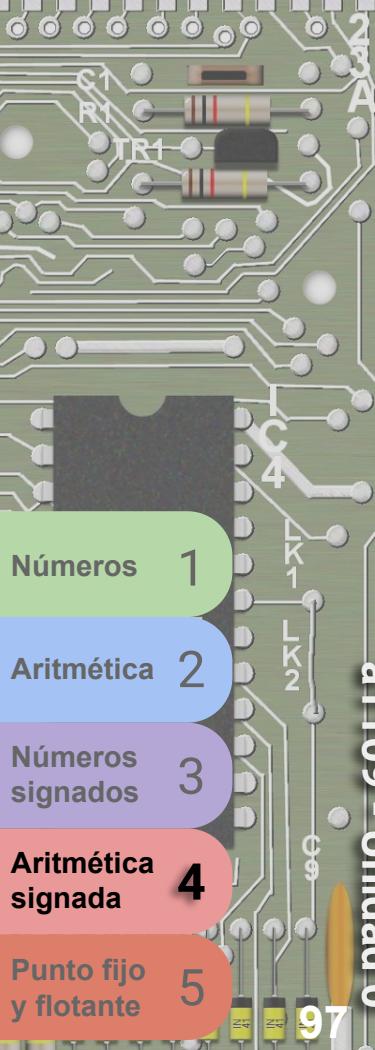
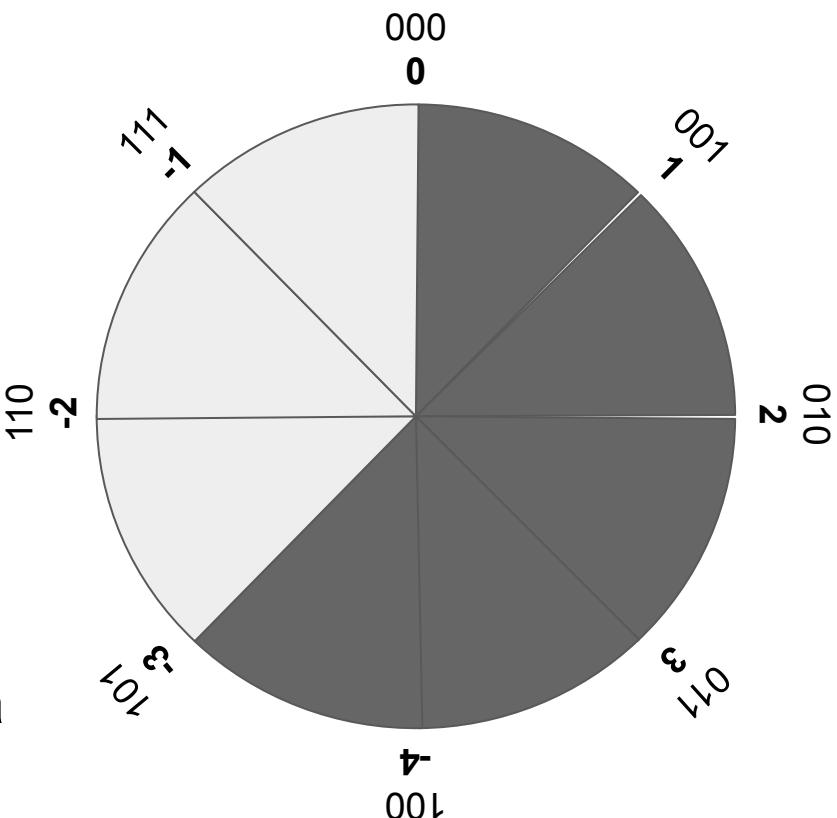


3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Vemos que la suma se desborda y pasa a los valores negativos

Un mundo de 3 bits signados



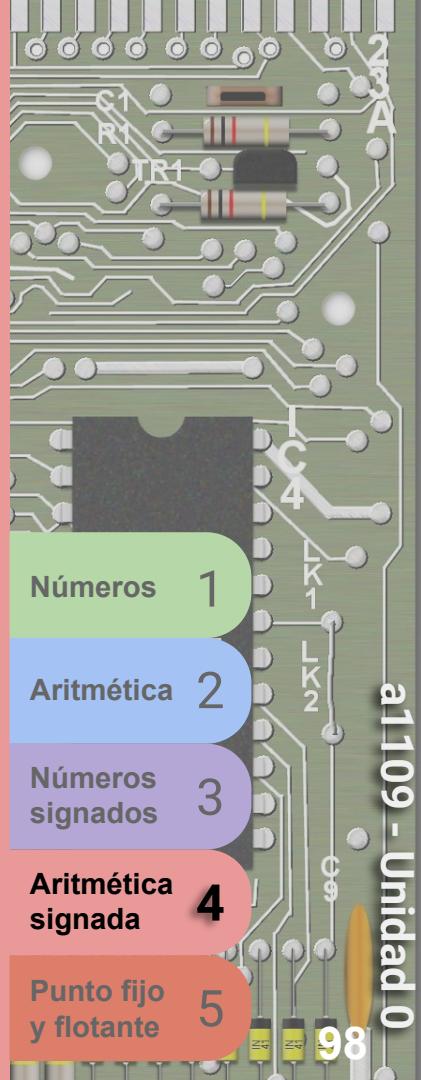
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
1 0 1	-3
+ 1 1 0	-2
—————	-5

Volvemos a plantear una suma, pero ahora de números negativos. Si sumamos $(-3) + (-2)$ esto equivale a $-3 - 2 = -5$. En decimal corregimos los signos y restamos... en binario solo sumamos



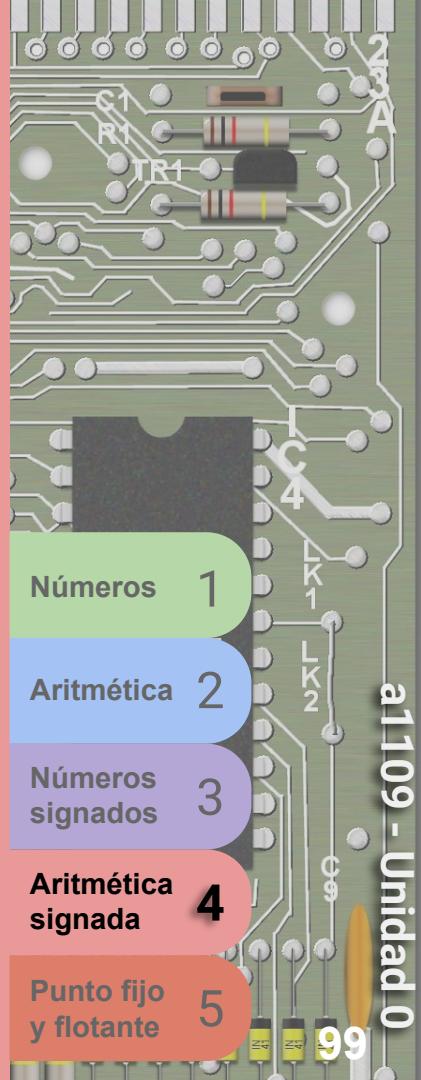
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario		Decimal	
+----- -----+			
1	0	1	-3
+	1	1	-2
1	0	1	-5

Nuevamente obtuvimos un carry, pero esto lo podemos ignorar. Lo que no podemos ignorar es que el bit de signo del resultado es 0 (positivo) mientras que estábamos sumando dos negativos.



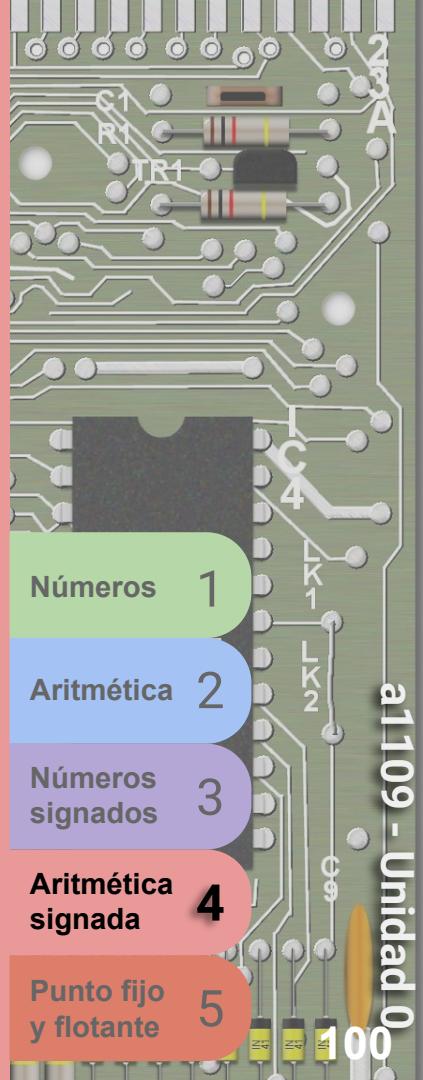
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

	Binario			Decimal
	1	0	1	-3
+	1	1	0	-2
	1	0	1	-5

Esta situación nuevamente representa un desborde (**overflow**). Este se produce en una suma de números signados donde los bits de signo de los operandos son iguales pero el resultado no.



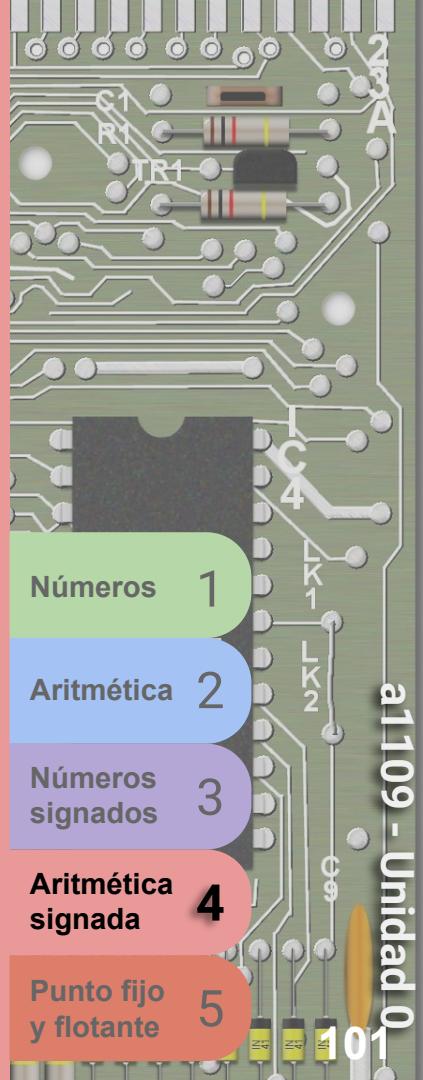
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario		Decimal	
+----- -----+			
1	0	1	-3
+	1	1	-2
1	0	1	-5

Como vemos el resultado $011_2 = 3$ en nuestro sistema en vez del -5 esperado (que no existe en nuestro sistema de 3 bits). El overflow nos indica que la operación es inválida.



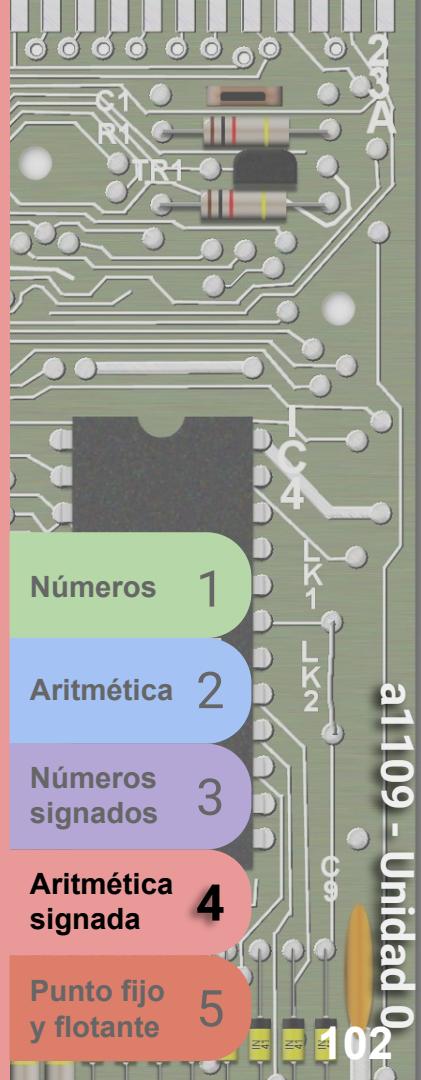
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
0 1 0	2		
-	0 1 1	3	
			-1

Ahora planteamos una resta... $2-3=-1$. Vemos que -1 es un valor existente en nuestro sistema.. así que restamos en binario.



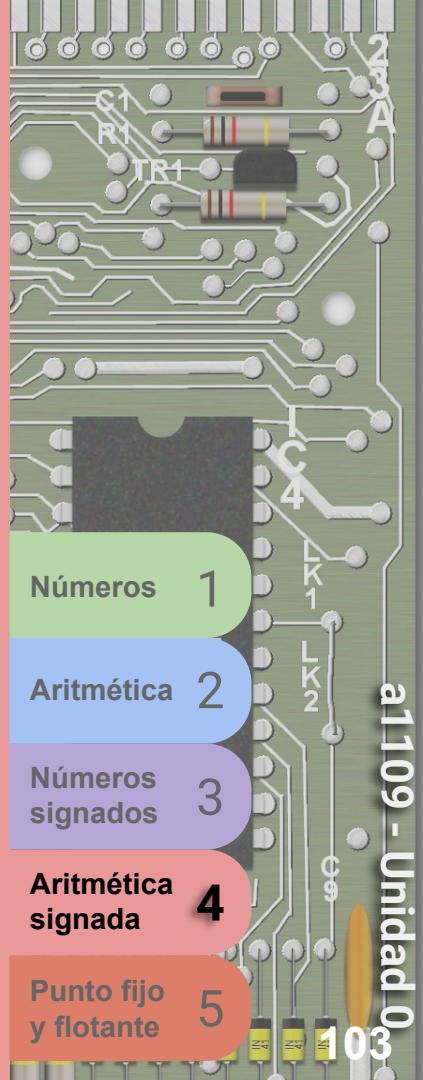
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
0 + 10 2	
- 0 1 1 3	
	1 -1

Ya en la primer columna necesitamos pedir prestado.



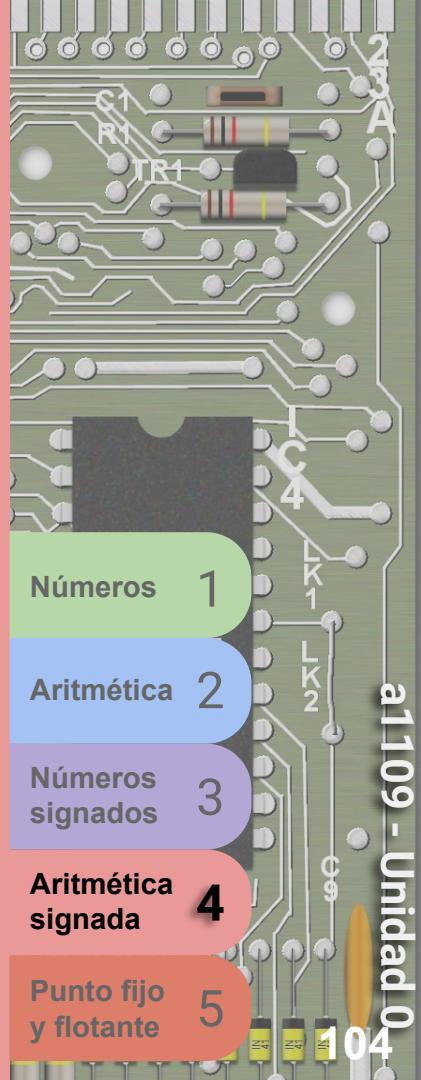
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
1	10	10	2
-	0	1	3
	1	1	-1

En la segunda columna pedimos un préstamo a la primera, que pasa a ser 1.



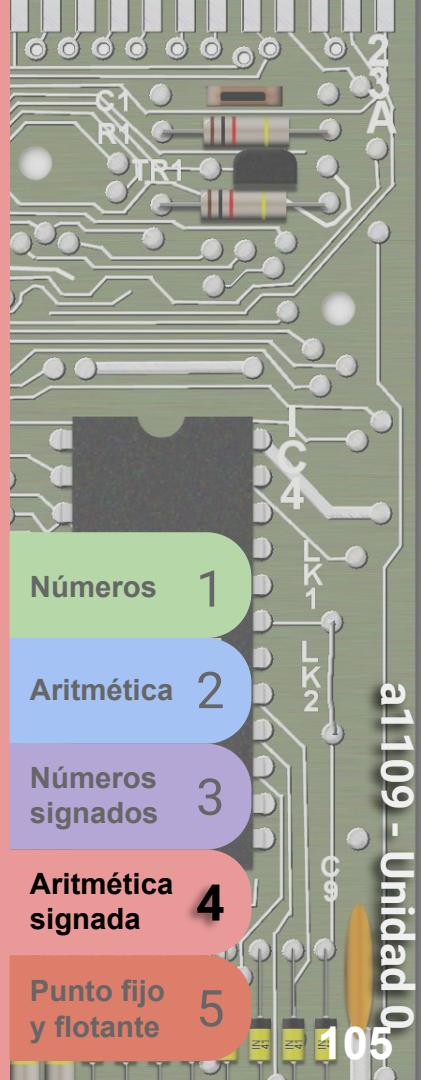
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
1	1	0	2
-	0	1	3
1	1	1	-1

Por último ahora tenemos la tercera columna ($1-0=1$) y resaltamos en rojo el prestado (borrow) de la cuarta columna.



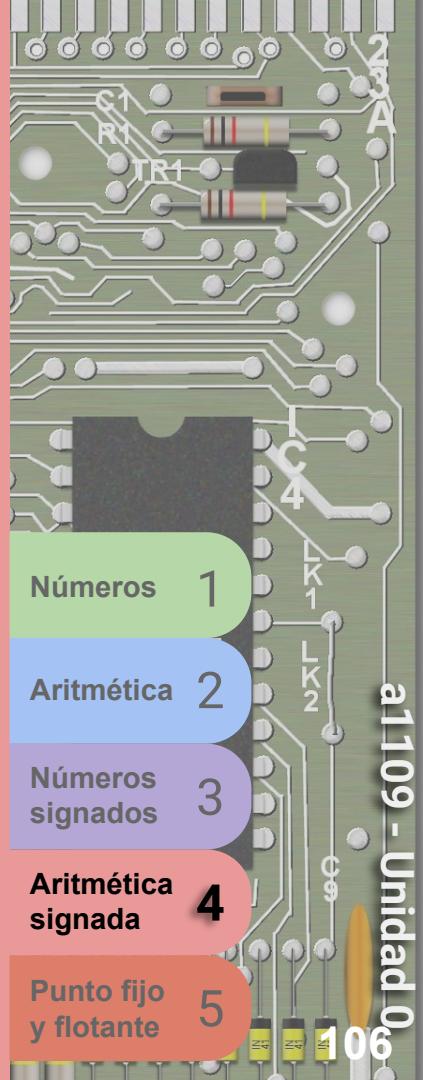
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
1 10 10	2
-	
0 1 1	3
1 1 1	-1

El resultado es efectivamente $111_2 = -1$. La operación de resta resultó en un número negativo válido, por ende despreciamos el borrow.



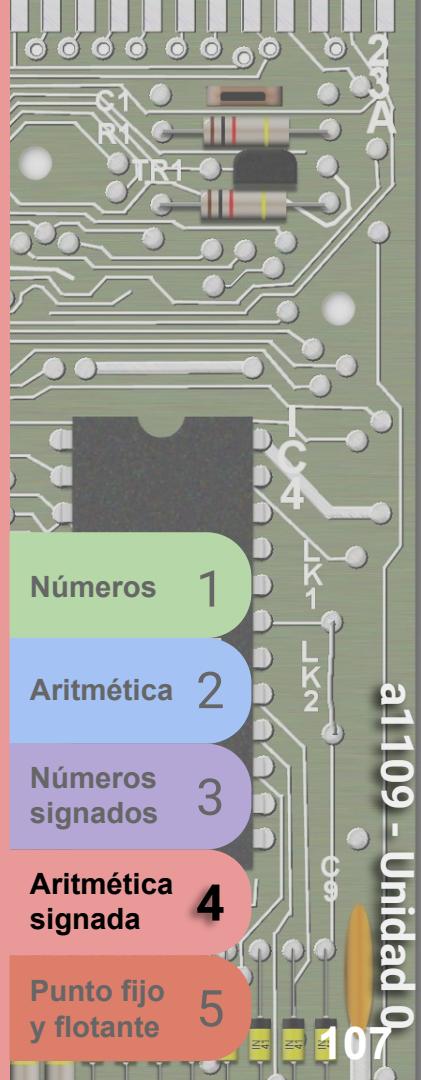
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal		
	{ }		
	1	0	1
-	0	1	0
	-3	2	-5

Volvemos a plantear una resta. En este caso $-3 - 2$ que sabemos es -5 en decimal. Vemos que -5 no existe en este sistema, así que seguro va a haber algún problema...



3 bits Valor

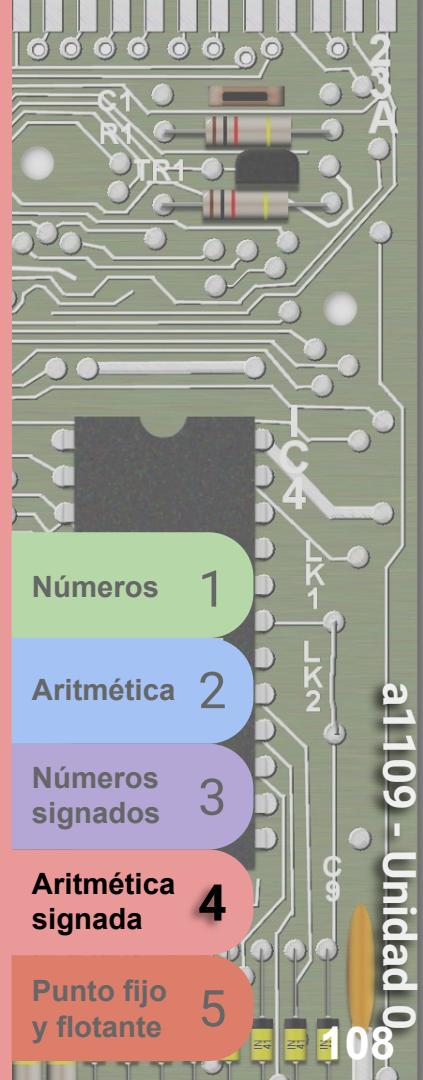
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario	Decimal
+ 1 0 1	-3
- 0 1 0	2

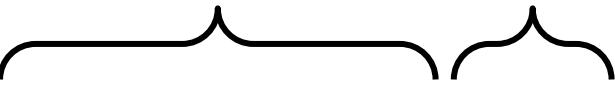
0 1 1	-5

Vemos que el resultado se resuelve sin problemas. Si bien hay un borrow de la segunda columna a la tercera, esto no trae más prestados. Pero el resultado 011 es 3 .. y no es el valor esperado!

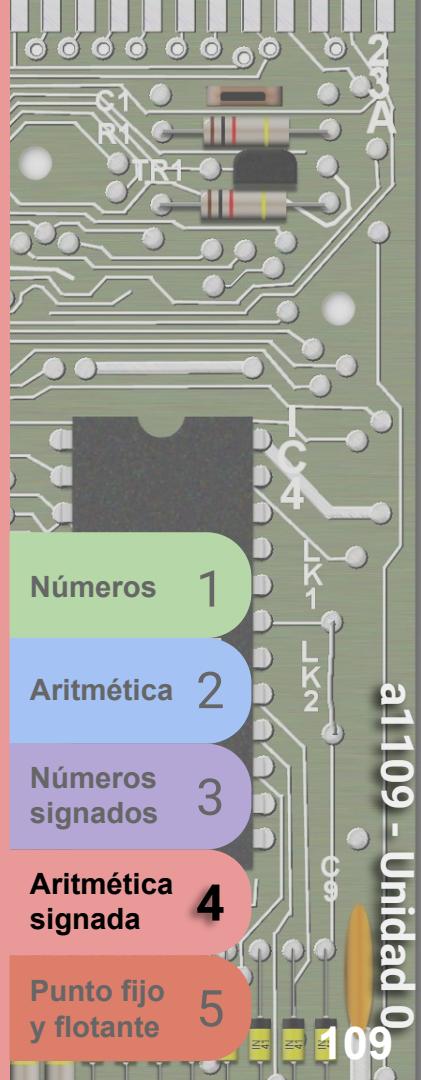


3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario		Decimal
		
+		-3
-		2
		0
		1
		1
		-5

Si realizamos una resta donde los bits de signo del minuendo y el sustraendo son distintos, y el bit de signo del resultado es opuesto al minuendo... esto es un desborde en la resta (**overflow**).



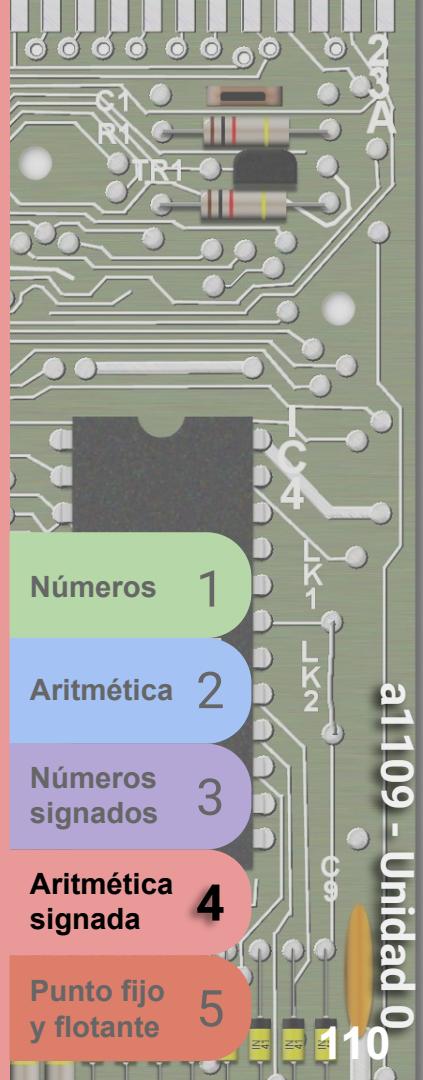
3 bits Valor

3 bits	Valor
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Un mundo de 3 bits signados

Binario		Decimal
+ 101 -3		
- 010 2		
<hr/> 0 1 1 -5		

Esto es intuitivo. Tenemos un minuendo negativo (-3) y le estamos quitando un número positivo, o sea que el resultado debería ser un número negativo. Sin embargo obtuvimos un positivo!!



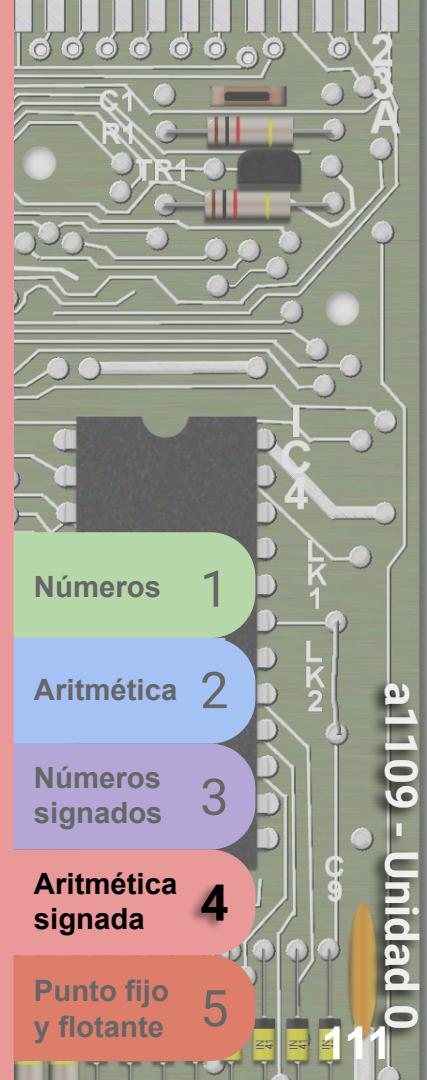
3 bits Valor

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

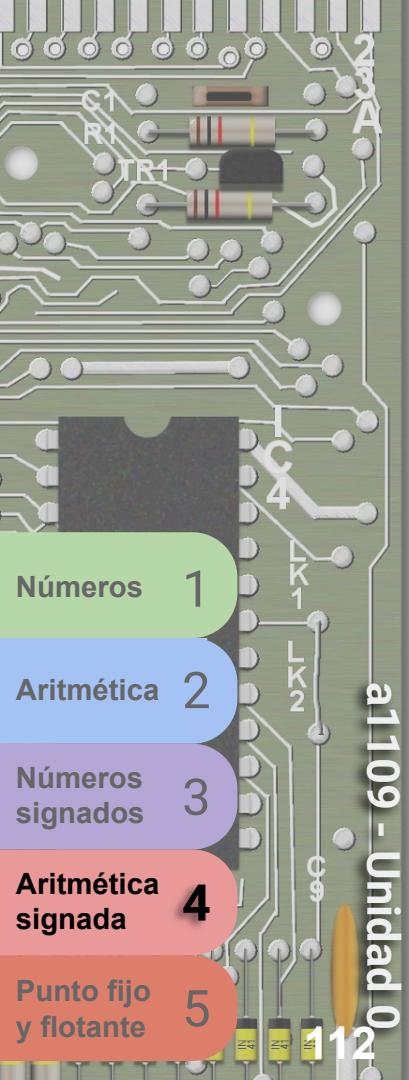
Un mundo de 3 bits signados

Binario		Decimal
_____ { }		
+ 101	-	-3
- 010		2
011		-5

Esta es otra situación de desborde, donde nos pasamos del alcance de números negativos para 3 bits. El resultado de esta operación es obviamente inválido.



Comparando números signados



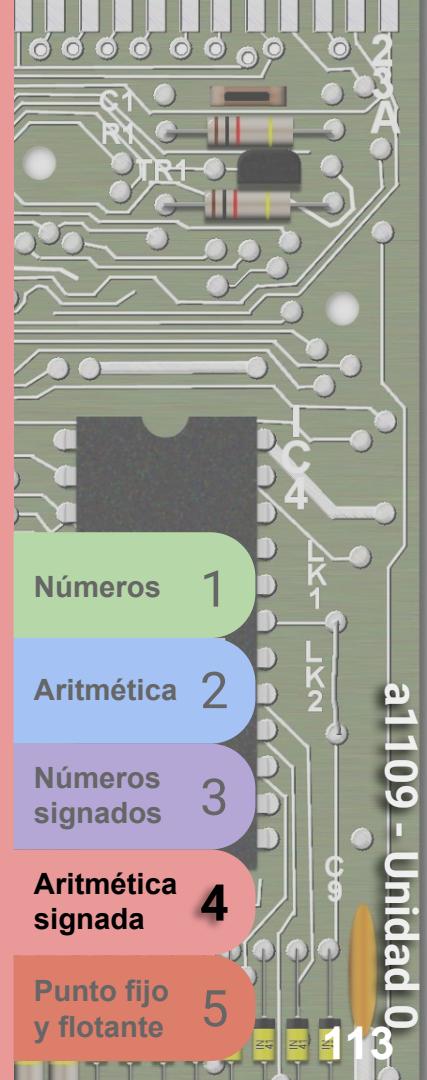
Comparamos haciendo una resta

Cuando queremos comparar dos números, por ejemplo A y B, y estos son números signados, realizamos la resta entre ambos.

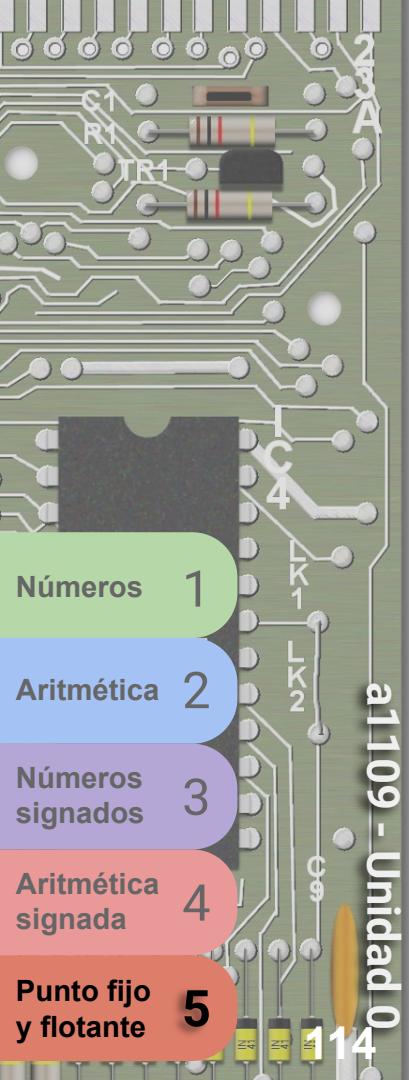
Siendo A el minuendo y B el sustraendo

- Si $A-B = 0$ entonces $A=B$
- Si $A-B$ y Negativo XOR Overflow = 1 $\rightarrow A < B$
- Si $A-B$ y Negativo XOR Overflow = 0 $\rightarrow A \geq B$

Se produce Overflow en la resta (signada obviamente) cuando el signo del minuendo y el sustraendo es distinto y el resultado tiene el signo opuesto al minuendo.



Punto fijo

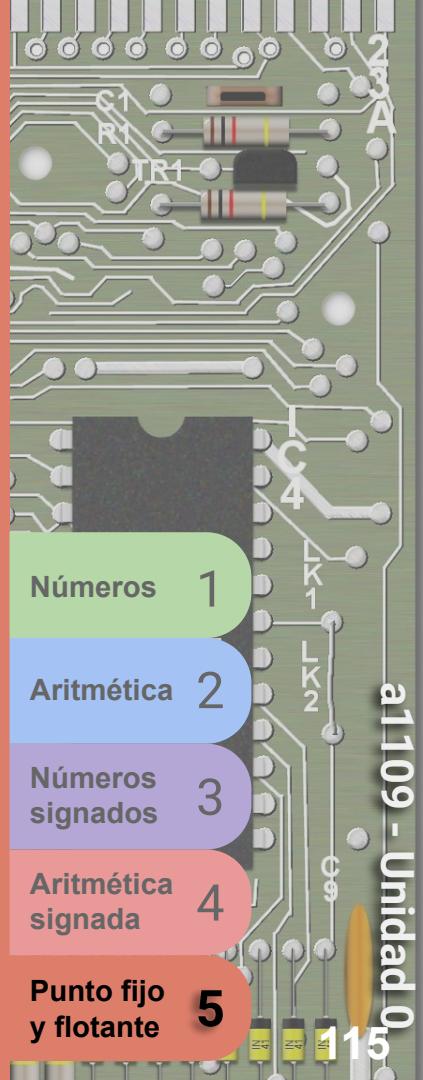


Coma implícita

$$12,345_{10} = 1 * 10^1 + 2 * 10^0 + 3 * 10^{-1} + 4 * 10^{-2} + 5 * 10^{-3}$$

$$12,345_{10} = 10 + 2 + \frac{3}{10} + \frac{4}{100} + \frac{5}{1000}$$

$$N_B = \sum_{i=-m}^n a_i * B^i$$



Conversión de base

Número	Base	Resultado	Entero
0,375	*	2	

Número	Base	Resultado	Entero
0,375	2	= 0,75	= 0
0,75	2		

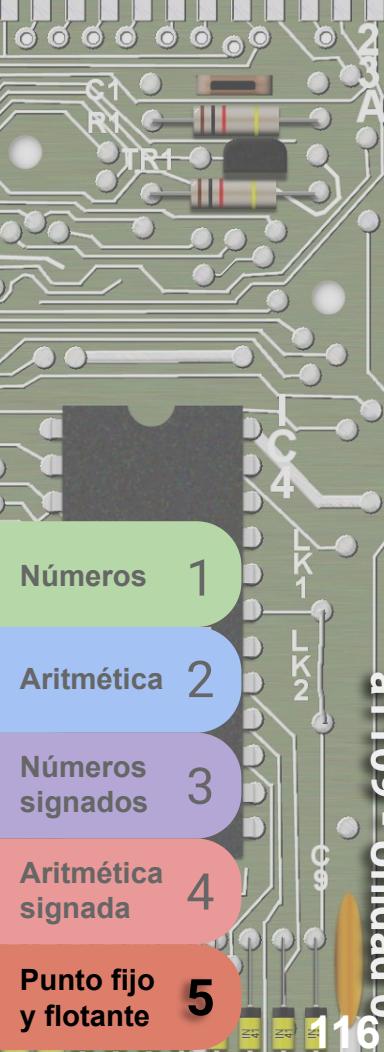
Número	Base	Resultado	Entero
0,375	2	0,75	0
0,75	2	1,5	1
0,5	2	1,0	1
0	2	0	

0,5

0,25

0,125

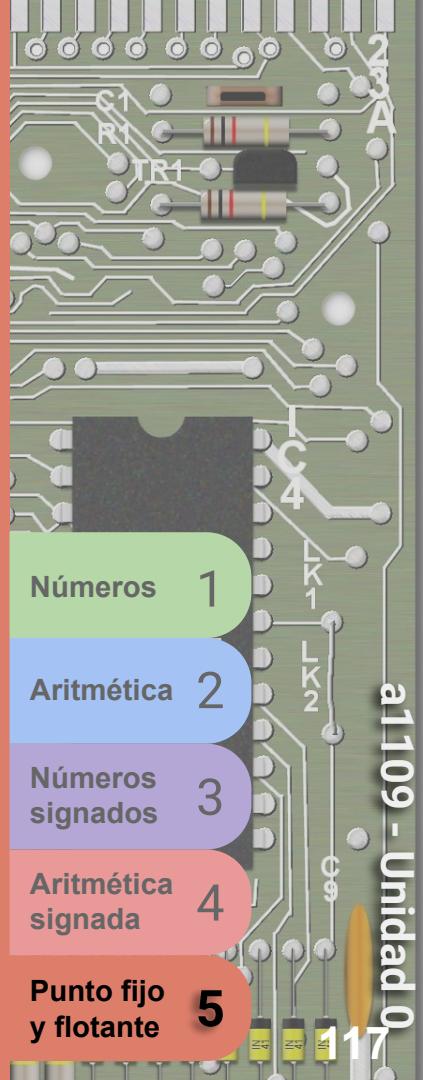
0,011
0+0,25+0,125



Conversión de base

Número	Base	Resultado	Entero
0,2	2	0,4	0

Número	Base	Resultado	Entero
0,2	2	0,4	0
0,4	2	0,8	0
0,8	2	1,6	1
0,6	2	1,2	1
0,2			



No todo es color de rosa

Número	Base	Resultado	Entero
0,2	2	0,4	0
0,4	2	0,8	0
0,8	2	1,6	1
0,6	2	1,2	1
0,2			

0,0011

$$0,125 + 0,0625 = 0,1875$$

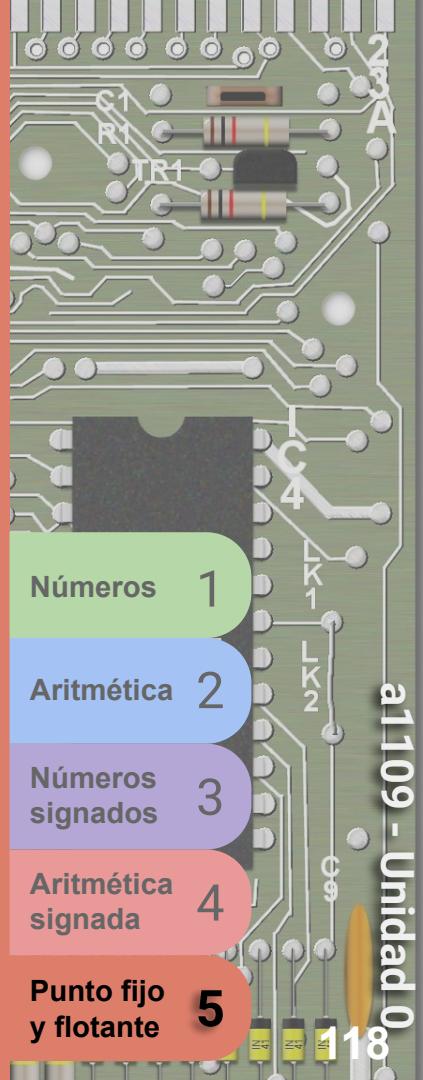
0,00110011

$$0,125 + 0,0625 + 0,0078125 + 0,00390625 = 0,19921875$$

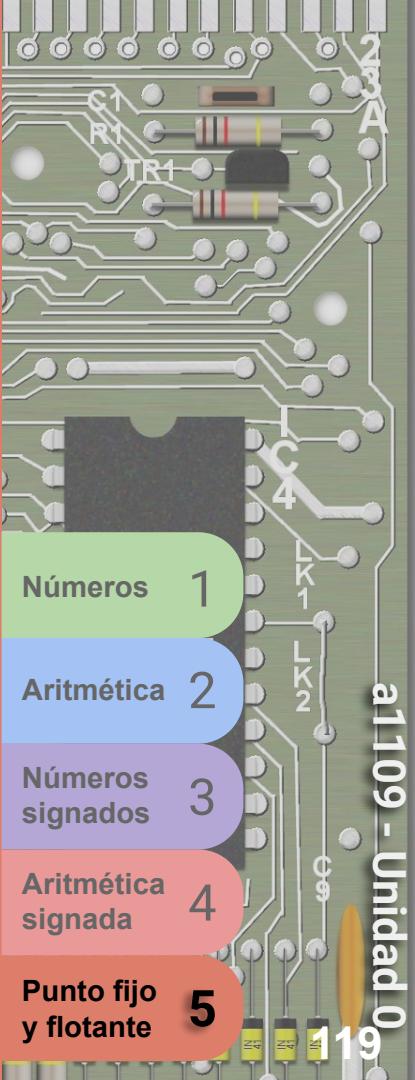
0,0011001100110011001100110011001100110011001100110011

0,1999999995343387127 ... ¿Es 0,2?

Si necesitamos representar exactamente 0,2, podemos usar punto fijo con dígitos BCD. Ver: <https://docs.python.org/3/library/decimal.html>



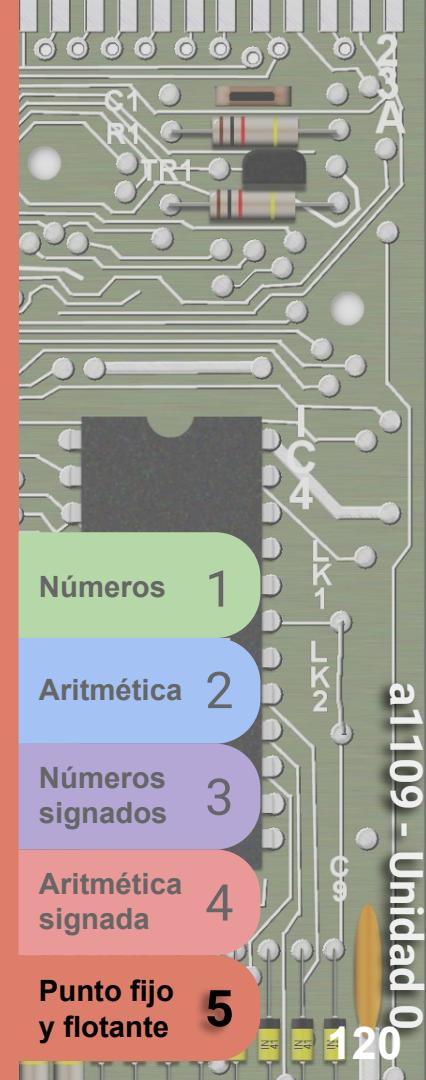
Punto Flotante



Notación científica

Imaginemos que tenemos que representar números muy grandes y números muy chicos. Con punto fijo, tenemos que elegir una cantidad de dígitos fraccionarios suficientes como para llegar a representar el número más chico y dígitos enteros suficientes como para representar la parte entera.

ej: para representar números en los millones, necesitamos al menos 7 dígitos (1000000) y si queremos representar algo en microsegundos necesitamos 6 dígitos (0,000001). Por ende con 15 dígitos (xxxxxxxx,yyyyyy) en punto fijo podríamos representar estos valores.



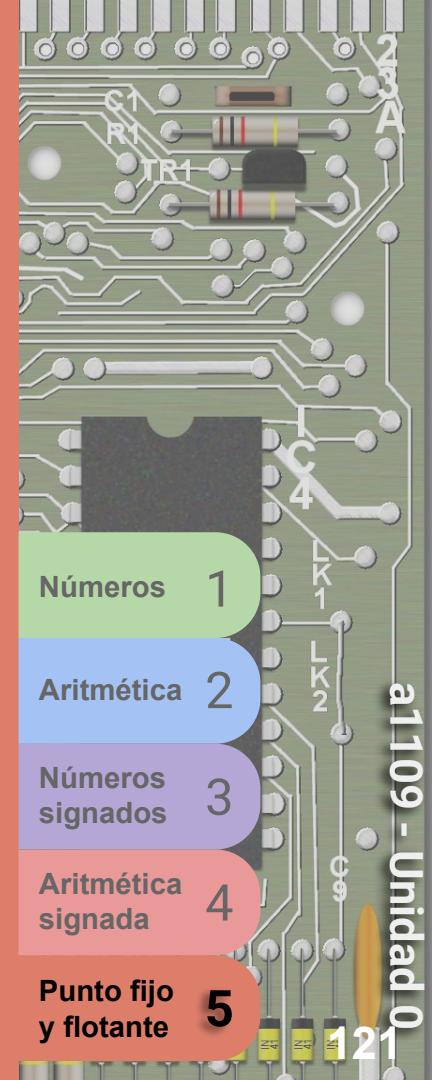
Notación científica

Dado que sería engorroso estar escribiendo muchos ceros para representar por ejemplo: 0000000,000001 podemos abreviar la notación en: 1×10^{-6} , o directamente como 1E-6.

En el caso de 10000000,000000 podemos usar 1E7.

Vemos que esto equivale a “correr la coma”, o dicho de otra forma multiplicar o dividir por la base (en este caso 10). Por ende 1E7 representa al número 1 multiplicado por la base elevada a 7, o sea multiplicar por un millón.

Esta notación es ventajosa en cuanto a la escritura pero también permite a simple vista comparar números en órdenes de magnitudes.



Punto flotante IEEE 754 simple precisión

La representación de números en punto flotante IEEE 754 no es otra cosa más que utilizar el esquema de notación científica pero adaptado al mundo binario.

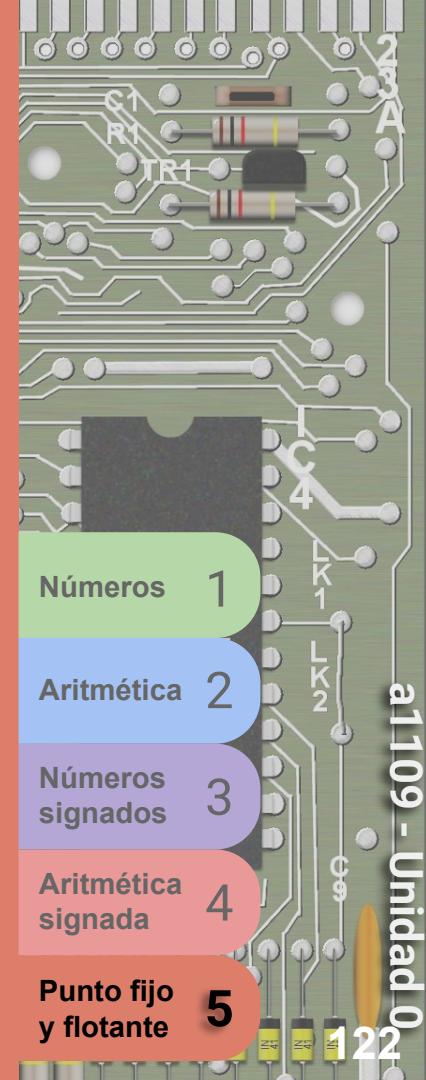
Se utiliza un espacio para representar si el número es positivo o negativo, luego una cantidad de bits para indicar el exponente (al cual se eleva la base 2) y por último una cantidad de bits para representar la mantisa.

Si bien en decimal el exponente tiene signo para indicar para qué lado se corre la coma, en IEEE754 desplazamos el exponente (representación en exceso) para cumplir el mismo objetivo.

Signo (1 bit)

Exponente (8 bits) offset 127

Mantisa (23 bits)



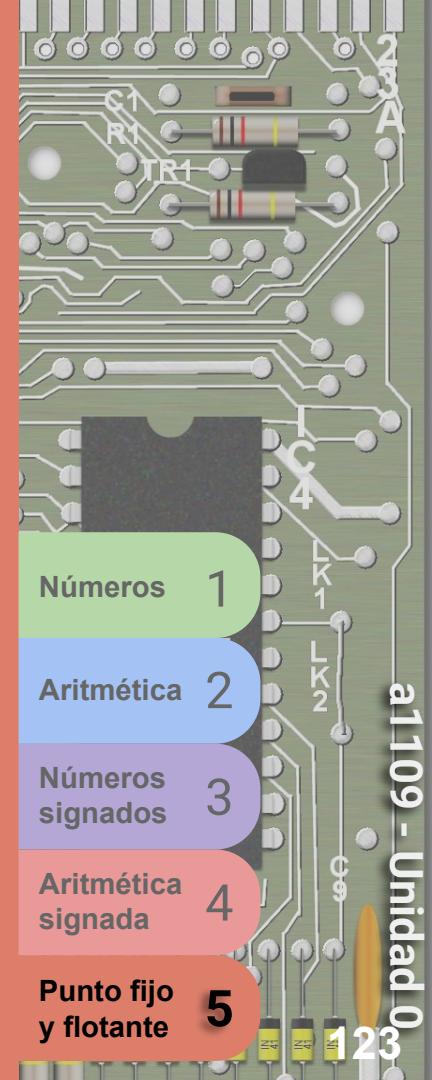
Punto flotante IEEE 754 simple precisión

Tomemos como ejemplo el número $12,75_{10}$. La parte entera (12) es en binario el 1100_2 y la parte decimal (0,75) es en binario $0,11_2$. Por ende nos queda el número binario: $1100,11_2$. Para normalizar el número corremos la coma justo antes del primer 1, por ende $1,10011$ corriendo la coma 3 veces. Este 3 indica el exponente (con base 2) pero ya que el mismo está desplazado 127 entonces $127+3=130$. El 130 en binario es 10000010_2 entonces nos queda:

0	10000010	10011000000000000000000000000000
---	----------	----------------------------------

Notemos que si bien el número normalizado es $1,10011\dots$ el primer 1 siempre estará presente, por ende podemos evitar guardarlo y asumirlo implícito. Por eso vemos que la mantisa es $1001100\dots$ en vez de $11001100\dots$ Cuando reconstruimos el número debemos recordar incorporar este 1 implícito.

Utilice este sitio para practicar : <https://www.h-schmidt.net/FloatConverter/IEEE754.html>



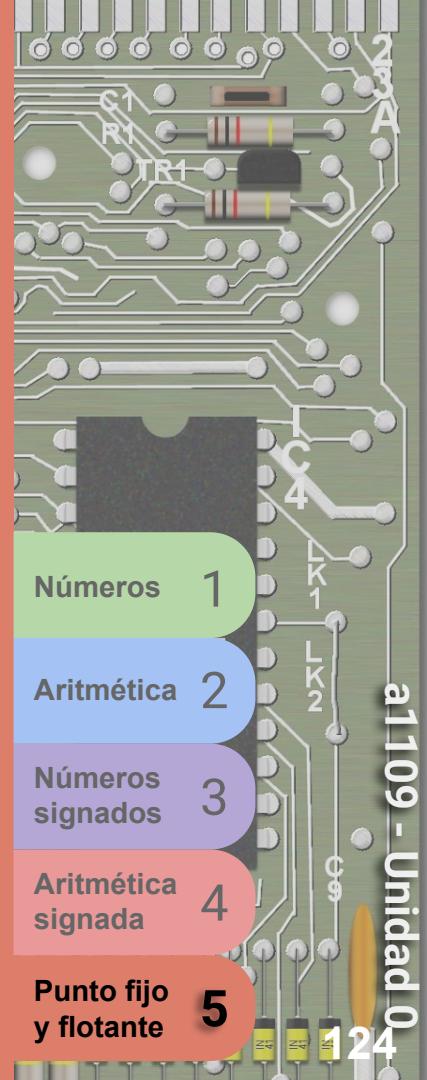
Punto flotante IEEE 754 simple precisión

Si bien para el caso anterior podemos volver a obtener el 12,75... esto no es cierto para todos los casos. El número 0,2 es periódico en binario. Esto quiere decir que si queremos representar por ejemplo el 12,2, la parte entera sigue siendo 1100_2 , pero la parte decimal es periódica $00110011001100110011\dots$

Luego el número binario seria: $1100,00110011001100110011\dots$. Si corremos la coma obtenemos $1,10000110011001100110011$. La coma se corrió nuevamente 3 posiciones por ende es el mismo exponente $10000010_2 = 130_{10}$.

0	10000010	10000110011001100110011
---	----------	-------------------------

Este número representa el $12,19999980926513671875$ en vez del 12,2.
Vemos que se acerca bastante pero no es exacto.

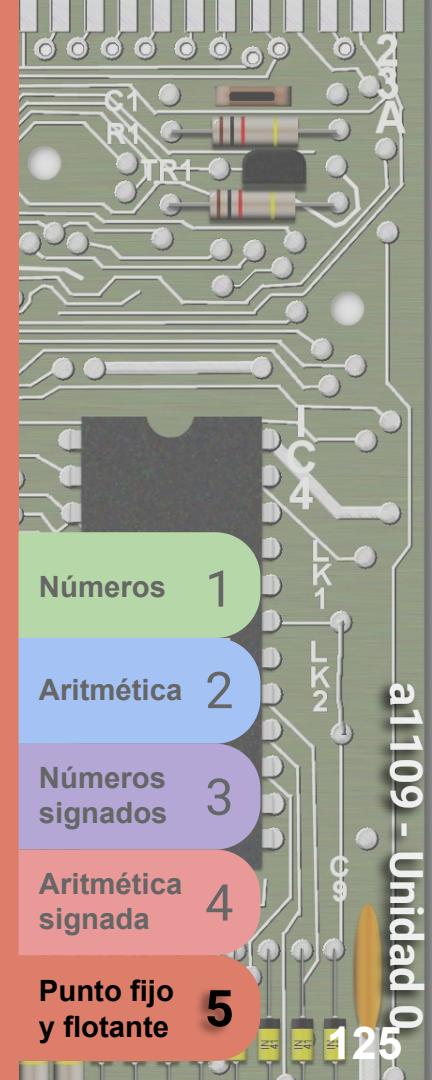


Punto flotante IEEE 754 simple precisión

Este sistema es muy útil para comparar números, ya que si tenemos dos números en punto flotante IEEE754 comparamos primero los signos, luego los exponentes y si son distintos entonces el mayor exponente representa un número más grande. Si los exponentes son iguales entonces comparamos las mantisas, siendo la mantisa más grande la que representa al número más grande.

La cantidad de bits que usamos para representar números es acotada. Esto quiere decir que algunos números tienen que truncarse para ser almacenados en IEEE754. Por ende cuando se transforma nuevamente en un número decimal puede no volver a obtenerse el número original.

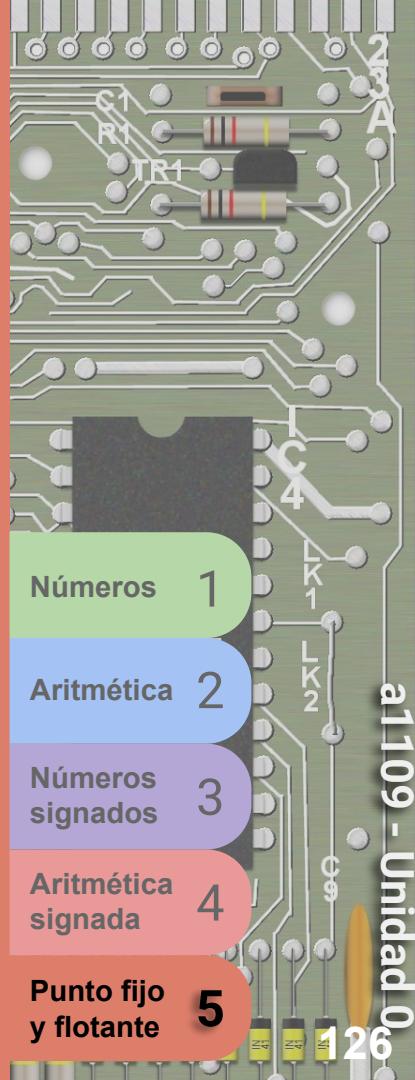
Existe un formato de doble precisión (recomendamos buscar su definición) que utiliza 64 bits. Esto brinda más bits en la mantisa con lo cual tenemos mayor precisión, pero tampoco soluciona el problema de los números periódicos.



Punto flotante IEEE 754 simple precisión

Existen casos especiales dentro de IEEE754 , por ejemplo cuando el exponente es 00000000 o 11111111. Estos casos representan casos particulares que son necesarios, como el infinito o la representación de un NaN (not a number) que suele ser el resultado de dividir por cero.

s	exp	M	Descripción
0	0111 1111	00000000000000000000000000000000	1,0
1	0111 1111	00000000000000000000000000000000	-1,0
0	0111 1100	00000000000000000000000000000000	0,125
0	0000 0000	00000000000000000000000000000000	+0
1	0000 0000	00000000000000000000000000000000	-0
0	1111 1111	00000000000000000000000000000000	+ infinito (ej: 1/0)
1	1111 1111	00000000000000000000000000000000	- infinito (ej: 1/-0)
?	1111 1111	xxxxxxxxxxxxxxxxxxxxxxxxxxxx	Not a Number (NaN)
?	0000 0000	00000000000000000000000000000001	Denormalizado 2^{-149}
?	0000 0001	00000000000000000000000000000000	2^{-126}
?	1111 1110	11111111111111111111111111111111	2^{127}



No todo es color de rosas con IEEE 754

A continuación vemos un pequeño programa en C. Se definen $a=0.2$, $B=25$ y se crea: $c=a*a*b$; y $d=b*a*a$; Sabemos que el orden de los factores no modifica el producto. Sin embargo cuando comparamos $c==d$ obtenemos que el contenido es distinto!

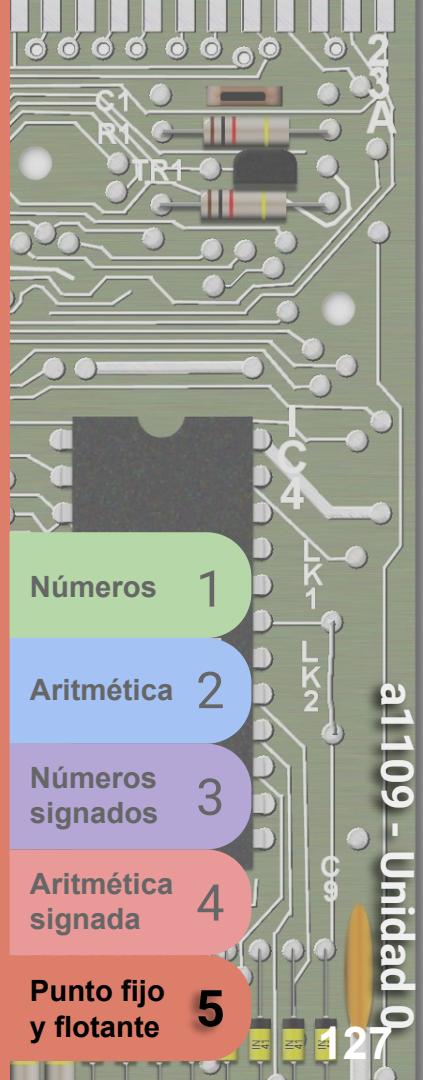
```
#include <stdio.h>
int main(){
    float a=0.2f;
    float b=25.0f;
    float c = a*a*b;
    float d = b*a*a;

    printf("a=%f \nb=%f \nc=%f \nd=%f \n",a,b,c,d);

    if (c == d) {
        printf("Iguales\n");
    } else {
        printf("Distintos\n");
    }
    return 0;
}
```

a=0.200000
b=25.000000
c=1.000000
d=1.000000
Como puede ser!
Distintos

Es evidente que ocurre algo raro. Evidentemente C o D no tienen exactamente el valor 1. A continuación accedemos a la memoria para leer qué valores hay almacenados en las posiciones de memoria A,B,C y D.



No todo es color de rosas con IEEE 754

Si accedemos a la memoria y vemos que valores tienen A, B, C y D vemos que C y D difieren en su último bit. Agregamos A' como ejemplo de A con su último bit cambiado. Vemos que A no es exactamente 0,2 sino que se pasa un poco, y A' (cambiando el último bit) no llega a ser 0,2 tampoco. Debido a que hay que truncar bits, entonces el orden de los factores PUEDE modificar el resultado. Es por esto que cuando trabajamos con floats tenemos que ser conscientes de esto a la hora de comparar.

$$A = 0\ 0111\ 1100\ 10011001100110011001101 = 0,20000000298023223876953125$$

$$A' = 0\ 0111\ 1100\ 10011001100110011001100 = 0,199999988079071044921875$$

$$B = 0\ 1000\ 0011\ 10010000000000000000000000 = 25$$

$$C = 00111111000000000000000000000000000000000001 = 1,00000011920928955078125$$

$$D = 00111111000000000000000000000000000000000000 = 1$$

$$C \neq D$$

Para comparar floats tomamos el valor absoluto de la resta entre los valores a comparar y verificamos si es menor a un cierto epsilon (valor muy pequeño totalmente arbitrario).

