1. Introducción a los sistemas de numeración

1.1 El sistema de numeración decimal

El sistema de numeración decimal [Wik14i] es un sistema de numeración que usa diez símbolos: '0', '1', '2', '3', '4', '5', '6', '7', '8' y '9'. Es el sistema de numeración más usado en todo el mundo. Según los antropólogos, el origen del sistema decimal está en los diez dedos que tienen los seres humanos en las manos que siempre han servido como base para contar. El sistema decimal es un sistema de numeración de base 10 puesto que son diez los símbolos diferentes que se usan para expresar cantidades en este sistema.

Definición 1.1.1 — Base. Es el número diferente de símbolos que se pueden usar en un sistema de numeración. En este libro usaremos la letra b para expresar la base.

Supongamos que queremos expresar cantidades usando 4 dígitos mediante el sistema decimal. En el resto del libro usaremos la letra N para expresar el número de dígitos a usar, por lo tanto en este caso $N = 4$. La primera cantidad a expresar es el *cero*, para ello usaremos el primero de los símbolos '0' para rellenar los cuatro dígitos, por lo tanto: *cero* $\rightarrow 0000_{10}$ (donde $\rightarrow$ expresa "se representa como"). El subíndice 10 se usa para indicar que la cantidad en cuestión está expresada en un sistema en base 10 (es decir usando el sistema decimal). ¿Qué debemos hacer para expresar la cantidad *uno*? La solución a esta pregunta es usar el siguiente símbolo del sistema decimal '1' en la posición menos significativa (la de más a la derecha): *uno* $\rightarrow 0001_{10}$. Para la siguiente cantidad, *dos*, utilizamos la misma estrategía, es decir usar el siguiente símbolo en la posición menos significativa: *dos* $\rightarrow 0002_{10}$.

El proceso continuaría igual para expresar las cantidades *tres*, *cuatro*, *cinco*, *seis*, *siete*, *ocho* y *nueve*. Pero, ¿qué ocurre con la cantidad que sucede a la cantidad *nueve* (es decir la cantidad *diez*)?. No podemos usar la estrategía anterior, el uso de un símbolo nuevo, puesto que los diez símbolos del sistema decimal ya han sido usados. La solución es reutilizar los símbolos existenes. Para ello, se usa el siguiente símbolo, al existente, en el dígito a la izquierda del menos significativo (es decir, se pasa del '0' al '1') y se vuelve a usar el primer símbolo en el dígito menos significativo (es decir, el símbolo '0').

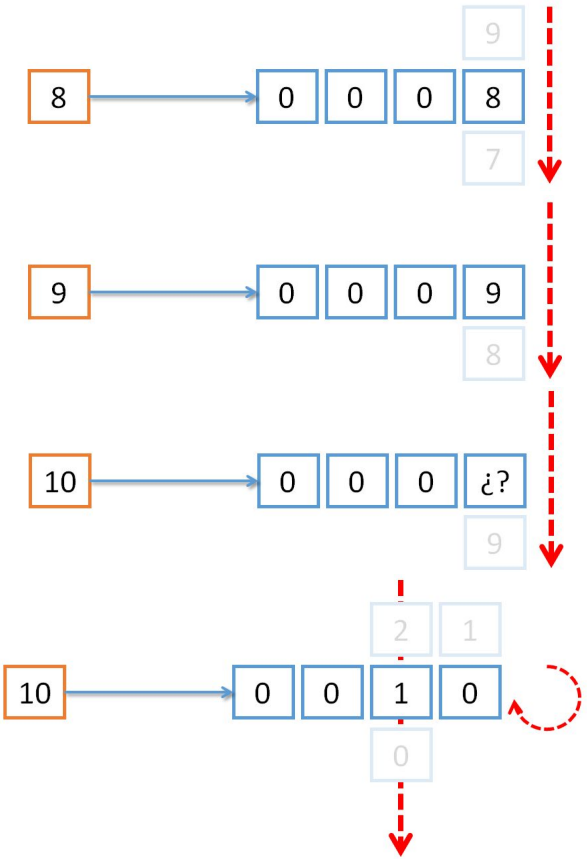


Figura 1.1: Ejemplo de como se forma en decimal las cantidades *ocho* y *nueve*. Para la cantidad *diez* no existe un nuevo símbolo que podamos usar, por lo tanto reutilizamos los símbolos existentes en el dígito más a la derecha y usamos el siguiente en el de su izquierda.

Por lo tanto, *diez* $\rightarrow 0010_{10}$. La Figura 1.1 muestra de forma esquemática este proceso.

Continuando con esta estrategia, llegaríamos a la cantidad *nueve mil novecientos noventa y nueve* como sigue: *nueve mil novecientos noventa y nueve* $\rightarrow 9999_{10}$. ¿Cómo expresamos la siguiente cantidad (es decir, *diez mil*)?. De acuerdo a la estrategia comentada, es evidente que se debería expresar de la siguiente forma: *diez mil* $\rightarrow 10\,000_{10}$. Sin embargo, solo tenemos 4 dígitos ($N = 4$) y por lo tanto, no podemos representar cantidades superiores a *nueve mil novecientos noventa y nueve* pues necesitan más de 4 dígitos. Por lo tanto, con $N = 4$ la cantidad mayor que se puede expresar es $10^4 - 1 = 9999$.

El número máximo de cantidades que se puede expresar en el sistema decimal con N dígitos es: 10^N. Las cantidades estarán en el rango $[0, 10^N - 1]$. El rango anterior se puede escribir también como $[0, 10^N)$.

Definición 1.1.2 — Rango. En lenguaje matemático, un rango es una forma de expresar el conjunto de todas las cantidades posibles comprendidas entre dos dadas. Para expresar un rango podemos usar los símbolos '[' y ']' o los símbolos '(' y ')'. En el primero de los casos, los números que definen el rango forman parte del mismo. En el segundo caso, estos números no forman parte del rango. Por ejemplo, $[1,4]$ es

Tabla 1.1: Valores decimales de los símbolos del sistema decimal

Símbolo	Valor decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

el conjunto formado por las cantidades 1, 2, 3 y 4. Sin embargo, $(1,4)$ es el conjunto formado por las cantidades 2 y 3, puesto que las cantidades 1 y 4 no forman parte del conjunto. También es posible definir un rango como $[1,4)$. En este último caso, el conjunto de cantidades será: 1, 2 y 3.

1.2 Sistemas de numeración posicionales

El sistema de numeración decimal es un sistema posicional de base 10. En un sistema de numeración posicional [Wik14f] cada dígito posee un valor que depende de:

1. Su posición relativa, la cual está determinada por la base.
2. Valor decimal del símbolo.

La Tabla 1.1 muestra los valores decimales de cada uno de los 10 dígitos del sistema decimal.

A pesar de que los sistemas de numeración posicionales se usan en la práctica totalidad de culturas actuales (casi todas usan el sistema decimal), no siempre ha sido así. En antiguas culturas, como la de Mesopotamia, el Antiguo Egipto, la Antigua Grecia o Roma, no utilizaban la notación posicional, lo que hacía sumamente complejo el cálculo, y dificultaba el desarrollo del álgebra. Por ejemplo, el sistema de numeración romano fue utilizado durante varios siglos a pesar de no ser tan útil como el sistema decimal usado en la actualidad.

1.2.1 Conversión de cualquier base a decimal

Sea b la base de un sistema de numeración posicional y N el número de dígitos a usar. Entonces, sea cual sea la base b en la que está expresado un número x_b (el subíndice indica la base) es posible expresar ese número en el sistema decimal x_{10} mediante la siguiente fórmula:

$$x_{10} = \sum_{p=N-1}^{0} x_b[p] * b^p \tag{1.1}$$

donde $x_b[p]$ es el valor decimal asociado al símbolo situado en la posición p-ésima del número x_b. Por ejemplo, si $x_b = 754_{10}$ ($b = 10$ y $N = 3$), entonces $x_b[2] = 7$, $x_b[1] = 5$ y $x_b[0] = 4$. Para obtener el valor decimal de cada uno de los símbolos es necesario consultar una tabla de equivalencias tal como la mostrada en la Tabla 1.1 para el sistema decimal. Este número se puede expresar como sigue: $754_{10} = 7*10^2+5*10^1+4*10^0 = 7*100+5*10+4*1 = 700+50+4$. La Fórmula 1.1 solo es aplicable a números enteros positivos.

Para números reales, se debe usar la Fórmula 1.2 , donde N^+ y N^- es el número de dígitos de la parte entera y de la parte fraccionaria, respectivamente.

$$x_{10} = \sum_{p=N^+-1}^{0} x_b[p] * b^p + \sum_{q=-1}^{-N^-} x_b[q] * b^q \tag{1.2}$$

Definición 1.2.1 — Parte entera y parte fraccionaria de un número real. Todo número real x puede escribirse en la forma $y+z$ donde y es un entero (la parte entera de x) y z es un número real no negativo menor que 1, denominado la parte fraccionaria o parte fraccional de x que es el resultado de restarle al número real la parte entera. Por ejemplo, el número $x = 5{,}24_{10}$ tiene parte entera $y = 5_{10}$ y parte fraccionaria $z = 0{,}24_{10}$.

■ **Ejemplo 1.1** El número $305{,}45_{10}$ se expresa como sigue ($N^+ = 3$ y $N^- = 2$):

$$\begin{aligned} 305{,}45_{10} &= 3*10^2+0*10^1+5*10^0+4*10^{-1}+5*10^{-2} \\ &= 3*100+\ 0*10+\ \ 5*1+\ \ 4*0{,}1+\ 5*0{,}01 \\ &= \ \ 300+\ \ \ \ 0+\ \ \ \ 5+\ \ \ 0{,}4+\ \ 0{,}05 \end{aligned}$$

■

■ **Ejemplo 1.2** El número $3876{,}213_{10}$ se descompone usando la Fórmula 1.2 como sigue:

$$\begin{aligned} 3876{,}213_{10} &= \ 3*10^3+8*10^2+7*10^1+6*10^0+2*10^{-1}+1*10^{-2}+\ 3*10^{-3} \\ &= 3*1000+8*100+\ 7*10+\ \ 6*1+\ \ 2*0{,}1+\ 1*0{,}01+3*0{,}001 \\ &= \ \ 3000+\ \ \ 800+\ \ \ 70+\ \ \ \ 6+\ \ \ \ 0{,}2+\ \ \ 0{,}01+\ \ \ 0{,}003 \end{aligned}$$

■

Ejercicio 1.1 Aplica la Fórmula 1.2 al número $567{,}087_{10}$ tal como se ha realizado en el ejemplo 1.2 ■

Ejercicio 1.2 Aplica la Fórmula 1.2 al número $0{,}0807_{10}$ tal como se ha realizado en el ejemplo 1.2 ■

Ejercicio 1.3 Aplica la Fórmula 1.2 al número $500{,}002_{10}$ tal como se ha realizado en el ejemplo 1.2 ■

Tabla 1.2: Valores decimales de los símbolos del sistema *Tri*

Símbolo	Valor decimal
α	0
β	1
γ	2

1.2.2 Rango de cantidades expresables

Sea cual sea la base, es posible saber el rango de cantidades que se pueden expresar usando N dígitos usando la siguiente fórmula:

$$\Delta = [0, b^N - 1] \tag{1.3}$$

donde Δ es el conjunto de todas las cantidades que se pueden expresar dada una base b y un número de dígitos N.

▪ **Ejemplo 1.3** Calcula la cantidad más grande que se puede expresar en un sistema posicional de base 3 con 5 dígitos. En este caso, $b = 3$ y $N = 5$, por lo tanto aplicando la Fórmula 1.3, la cantidad más grande es $b^N - 1 = 3^5 - 1 = 242$, dicho de otra forma, el rango es $[0, 242]$. ▪

Ejercicio 1.4 Calcula la cantidad más grande que se puede expresar en un sistema posicional de base 7 ($b = 7$) con 4 dígitos ($N = 4$). ▪

Ejercicio 1.5 Calcula la cantidad más grande que se puede expresar en un sistema posicional de base 2 ($b = 2$) con 5 dígitos ($N = 5$). ▪

Como se puede comprobar, cuanto más pequeña es la base, más dígitos son necesarios para expresar la misma cantidad. Por ejemplo, para expresar la cantidad *setenta*, son necesarios 2 dígitos si $b = 10$, 3 si $b = 8$ y 7 si $b = 2$.

1.2.3 *Tri*: Un sistema de numeración inventado de base 3

En el aparatado anterior hemos visto como se puede aplicar las fórmulas 1.1 y 1.2 a números expresados en base 10, pero dichas fórmulas se pueden aplicar a números expresados en cualquier base. Supongamos la existencia de un sistema en base 3 ($b = 3$) con tres símbolos: 'α', 'β' y 'γ', y que tal como muestra la Tabla 1.2 se corresponden con los valores decimales 0, 1 y 2, respectivamente.

A este sistema le llamaremos *Tri* a lo largo de este libro y es una invención que nos servirá para entender mejor como funcionan los sistemas de numeración posicionales.

Para formar los números en este sistema tenemos que seguir la misma estrategía que usamos en el apartado 1.1 pero adaptándolo al número de símbolos que podemos usar en este nuevo sistema de numeración. Así, las tres primeras cantidades, *cero*, *uno* y *dos*, se expresarán como sigue (con $N = 4$): *cero* $\rightarrow \alpha\alpha\alpha\alpha_3$, *uno* $\rightarrow \alpha\alpha\alpha\beta_3$ y *dos* $\rightarrow \alpha\alpha\alpha\gamma_3$. Para la siguiente cantidad (*tres*) ocurrirá lo mismo que pasaba en el sistema decimal con la cantidad *diez*, no podemos usar un nuevo símbolo pues en este sistema de base

3 solo tenemos 3 símbolos. Por lo tanto, tenemos que reutilizar los existentes, y de esta forma la cantidad *tres* se expresará como sigue: $tres \rightarrow \alpha\alpha\beta\alpha_3$. De forma similar, las siguientes cantidades serán expresadas de la siguiente forma: $cuatro \rightarrow \alpha\alpha\beta\beta_3$, $cinco \rightarrow \alpha\alpha\beta\gamma_3$, $seis \rightarrow \alpha\alpha\gamma\alpha_3$ y así sucesivamente. La última cantidad que podemos expresar con el sistema *Tri* (b=3) y con 4 dígitos ($N = 4$) se obtiene aplicando la Fórmula 1.3: $b^N - 1 = 3^4 - 1 = 80$.

Para obtener el equivalente de un número expresado en el sistema Tri en el sistema decimal aplicaremos las fórmulas 1.1 o 1.2 según el número sea un entero o un real, respectivamente. Por ejemplo, para obtener el equivalente en decimal del número $\alpha\beta\gamma_3$, aplicaremos la Fórmula 1.1 tal como muestra el ejemplo 1.4.

■ **Ejemplo 1.4** Calcula el equivalente en base 10 del número $\alpha\beta\gamma_3$ expresado en el sistema de numeración *Tri*.

$$\begin{aligned}\alpha\beta\gamma_3 &= 0*3^2+1*3^1+2*3^0\\ &= 0*9+\ 1*3+\ 2*1\\ &= \quad 0+\quad 3+\quad 2\\ &= \ 5_{10}\end{aligned}$$

■

Recordar que tal como muestra la Tabla 1.2, los valores decimales de los símbolos 'α', 'β' y 'γ' son 0, 1 y 2, respectivamente.

Ejercicio 1.6 Aplica la Fórmula 1.1 al número $\beta\alpha\beta\gamma\alpha_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.4 ■

Ejercicio 1.7 Aplica la Fórmula 1.1 al número $\gamma\beta\beta\alpha_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.4 ■

Además de representar números enteros, el sistema *Tri* también puede ser usado para representar números reales. Para obtener elequivalente en decimal debemos aplicar la Fórmula 1.2, tal como se muestra en el ejemplo 1.5.

■ **Ejemplo 1.5** Calcula el equivalente en base 10 del número $\beta\beta\gamma,\alpha\beta_3$ expresado en el sistema de numeración *Tri*.

$$\begin{aligned}\beta\beta\gamma,\alpha\beta_3 &= 1*3^2+1*3^1+2*3^0+\ 0*3^{-1}+\ 1*3^{-2}\\ &= \ 1*9+\ 1*3+\ 2*1+0*(1/3)+1*(1/9)\\ &= \quad 9+\quad 3+\quad 2+\quad 0+\quad 1/9\\ &= 14.\widehat{1}_{10}\end{aligned}$$

■

El símbolo '$\widehat{\ }$' se usa para expresar periodo. En este ejemplo, el número resultado es $14{,}111111\ldots$ debido a que $1/9$ tiene un número infinito de cifras decimales. En lenguaje matemático se usa el periodo para representar cifras decimales que se repiten

indefinidamente. Por lo tanto, el número $14{,}111111\ldots$ se escribe $14.\widehat{1}$ para indicar que el 1 se repite indefinidamente.

Ejercicio 1.8 Aplica la Fórmula 1.2 al número $\beta\alpha\beta\gamma\alpha,\gamma\gamma_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.5 ■

Ejercicio 1.9 Aplica la Fórmula 1.2 al número $\gamma\beta\beta\alpha,\beta\alpha\beta_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.5 ■

1.3 Bits y bytes

Como hemos comentado anteriormente, los ordenadores codifican la información usando el sistema binario el cual es un sistema de numeración que usa solamente los símbolos '0' y '1'. A la hora de almacenar información, un ordenador puede almacenar en un elemento de información un '1' o un '0' usando para ello una magnitud física medible y que presente dos estados bien diferenciados, como por ejemplo el nivel de voltaje, el magnetismo o la capacitancia, entre otros tipos de magnitudes. Al elemento de información capaz de almacenar un único dígito en binario se le denomina bit [Wik14a]. Un conjunto de 8 bits se denomina byte [Wik14b]. En informática se usa el byte como unidad para medir la cantidad de información que es capaz de almacenar un dispositivo. Al igual que con otro tipo de unidades, se pueden utilizar los prefijos del Sistema Internacional de Unidades (SI), kilo (K), mega (M), giga (G) y tera (T) para representar 10^3, 10^6, 10^9 y 10^{12} bytes, respectivamente.

Existe cierta confusión con el uso de estos prefijos, puesto que en ocasiones a 2^{10} bytes se le denomina kilobyte, aunque en realidad $2^{10} = 1024$ y no 1000. De forma similar $2^{20} \neq 10^6$ (mega), $2^{30} \neq 10^9$ (giga) y $2^{40} \neq 10^{12}$ (tera). Para evitar confusiones, es preferible usar los prefijos binarios kibi (Ki), mebi (Mi), gibi (Gi) y tebi (Ti) que usan potencias de 2. En este libro se usarán los prefijos KB, MB, GB, TB para representar 10^3, 10^6, 10^9 y 10^{12} bytes; y KiB, MiB, GiB, TiB para representar 2^{10}, 2^{20}, 2^{30} y 2^{40} bytes. De esta forma seguiremos las recomendaciones de la octava edición del SI publicada en el año 2006 que precisa que los prefijos del SI se deben utilizar estrictamente para referirse a potencias de 10, y que recomienda el uso de los prefijos binarios en el campo de la tecnología de la información para evitar el uso incorrecto de los prefijos del SI. Los prefijos binarios están incluidos en el standard ISO/IEC 80000-13. Sin embargo, es importante tener en cuenta que, a pesar de las recomendaciones del SI, en multitud de ocasiones todavía podemos encontrar libros, páginas web, manuales de dispositivos de almacenamiento y otros documentos que utilizan el término kilobyte como 2^{10} bytes cuando, como hemos comentado anteriormente, es erróneo. Las tablas 1.3 y 1.4 muestran los principales prefijos decimales y binarios, respectivamente.

Hemos comentado anteriormente que para medir la cantidad de información que el dispositivo de almacenamiento es capaz de almacenar se usa el byte y sus múltiplos. Así, cuando compramos un disco duro, la cantidad de información que puede almacenar puede ser por ejemplo, $500GB$, es decir 500 gigabytes que es igual a $500 * 10^9$ bytes. Puesto que un byte son 8 bits, un disco duro de $500GB$ es capaz de almacenar $500 * 10^9 * 8$

Tabla 1.3: Principales prefijos decimales

Nombre	Prefijo	Número bytes
kilobyte	KB	$10^3 = 1000$
Megabyte	MB	$10^6 = 1\,000\,000$
Gigabyte	GB	$10^9 = 1\,000\,000\,000$
Terabyte	TB	$10^{12} = 1000\,000\,000\,000$

Tabla 1.4: Principales prefijos binarios

Nombre	Prefijo	Número bytes
kibibyte	KiB	$2^{10} = 1024$
Mebibyte	MiB	$2^{20} = 1\,048\,576$
Gibibyte	GiB	$2^{30} = 1\,073\,741\,824$
Tebibyte	TiB	$2^{40} = 1\,099\,511\,627\,776$

bits. Dicho de otro modo, este disco duro tiene $500 * 10^9 * 8$ celdas donde se puede escribir un 1 o un 0. Ocurre en ocasiones que el fabricante del disco duro interpreta correctamente las unidades, es decir, para el fabricante $3TB$ son $3 * 10^{12}$ bytes, sin embargo, el sistema operativo lo interpreta incorrectamente, puesto que comete el error de creer que $3TB$ son $3 * 2^{40}$ bytes. Por lo tanto, cuando le solicitamos al sistema operativo que nos informe sobre cuantos bytes caben en el disco duro nos informa que hay sobre $2{,}7TB$ cuando en realidad debería decir que hay sobre $2{,}7TiB$. Este número se obtiene facilmente aplicando una regla de tres para saber cuandos TiB son $3TB$: $3 * (10^{12}/2^{40}) \approx 2{,}7$. De esta forma, da la impresión de que el fabricante nos ha "engañado", cuando en realidad es el sistema operativo el que comete el error, puesto que el disco duro tiene los bytes que el fabricante dice que tiene.

Para medir la cantidad de información que se transmite por unidad de tiempo (por ejemplo en una red WiFi), se usa el bit (y no el byte) y sus múltiplos. Así, podemos tener en casa una red WiFi capaz de transmitir la información a $500Mib/s$, es decir $500 * 2^{20}$ bits por segundo. Para diferenciar entre bytes y bits, se suele usar una *B* mayúscula para byte y una *b* minúscula para bit.

Ejercicio 1.10 ¿Cuántos bytes son $45KB$? ■

Ejercicio 1.11 ¿Cuántos bits son $2KiB$? ■

Ejercicio 1.12 ¿Cuántos bits son $3MiB$? ■

Ejercicio 1.13 Si una red es capaz de transmitir a 3Mib por segundo ¿Cuántos bits transmitirá en un minuto? ■

Ejercicio 1.14 Si una red es capaz de transmitir a 2Kib por segundo ¿Cuántos bytes transmitirá en un minuto? ■

1.4 Solución a los ejercicios propuestos

Solución al ejercicio 1.1 Aplica la Fórmula 1.2 al número $567{,}087_{10}$ tal como se ha realizado en el ejemplo 1.2:

$$\begin{aligned} 567{,}087_{10} &= 5*10^2+6*10^1+7*10^0+0*10^{-1}+8*10^{-2}+7*10^{-3} \\ &= 5*100+6*10+7*1+0*0{,}1+8*0{,}01+7*0{,}001 \\ &= 500+60+7+0+0{,}08+0{,}007 \end{aligned}$$

■

Solución al ejercicio 1.2 Aplica la Fórmula 1.2 al número $0{,}0807_{10}$ tal como se ha realizado en el ejemplo 1.2:

$$\begin{aligned} 0{,}0807_{10} &= 0*10^0+0*10^{-1}+8*10^{-2}+0*10^{-3}+7*10^{-4} \\ &= 0*1+0*0{,}1+8*0{,}01+0*0{,}001+7*0{,}0001 \\ &= 0+0{,}0+0{,}08+0+0{,}0007 \end{aligned}$$

■

Solución al ejercicio 1.3 Aplica la Fórmula 1.2 al número $500{,}002_{10}$ tal como se ha realizado en el ejemplo 1.2:

$$\begin{aligned} 500{,}002_{10} &= 5*10^2+0*10^1+0*10^0+0*10^{-1}+0*10^{-2}+2*10^{-3} \\ &= 5*100+0*10+0*1+0*0{,}1+0*0{,}01+2*0{,}001 \\ &= 500+0+0+0+0+0{,}002 \end{aligned}$$

■

Solución al ejercicio 1.4 Calcula la cantidad más grande que se puede expresar en un sistema posicional de base 7 ($b = 7$) con 4 dígitos ($N = 4$).

La cantidad más grande es $b^N - 1 = 7^4 - 1 = 2400_{10}$. ■

Solución al ejercicio 1.5 Calcula la cantidad más grande que se puede expresar en un sistema posicional de base 2 ($b = 2$) con 5 dígitos ($N = 5$).

La cantidad más grande es $b^N - 1 = 2^5 - 1 = 31_{10}$ ■

Solución al ejercicio 1.6 Aplica la Fórmula 1.1 al número $\beta\alpha\beta\gamma\alpha_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha

realizado en el ejemplo 1.4:

$$\begin{aligned}\beta\alpha\beta\gamma\alpha_3 &= 1*3^4+0*3^3+1*3^2+2*3^1+0*3^0\\ &= 1*81+0*27+\ 1*9+\ 2*3+\ 0*1\\ &= \ \ 81+\ \ \ 0+\ \ \ 9+\ \ \ 6+\ \ \ 0\\ &= 96_{10}\end{aligned}$$

■

Solución al ejercicio 1.7 Aplica la Fórmula 1.1 al número $\gamma\beta\beta\alpha_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.4:

$$\begin{aligned}\gamma\beta\beta\alpha_3 &= 2*3^3+1*3^2+1*3^1+0*3^0\\ &= 2*27+\ 1*9+\ 1*3+\ 0*1\\ &= \ \ 54+\ \ \ 9+\ \ \ 3+\ \ \ 0\\ &= 66_{10}\end{aligned}$$

■

Solución al ejercicio 1.8 Aplica la Fórmula 1.2 al número $\beta\alpha\beta\gamma\alpha,\gamma\gamma_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.5:

$$\begin{aligned}\beta\alpha\beta\gamma\alpha,\gamma\gamma_3 &= 1*3^4+0*3^3+1*3^2+2*3^1+0*3^0+\ 2*3^{-1}+\ 2*3^{-2}\\ &= 1*81+0*27+\ 1*9+\ 2*3+\ 0*1+2*(1/3)+2*(1/9)\\ &= \ \ 81+\ \ \ 0+\ \ \ 9+\ \ \ 6+\ \ \ 0+\ \ \ 2/3+\ \ \ 2/9\\ &= 96.\widehat{8}_{10}\end{aligned}$$

■

Solución al ejercicio 1.9 Aplica la Fórmula 1.1 al número $\gamma\beta\beta\alpha,\beta\alpha\beta_3$ expresado en el sistema de numeración *Tri* para obtener su equivalente en base 10, tal como se ha realizado en el ejemplo 1.5:

$$\begin{aligned}\gamma\beta\beta\alpha,\beta\alpha\beta_3 &= \ \ 2*3^3+1*3^2+1*3^1+0*3^0+\ 1*3^{-1}+\ 0*3^{-2}+\ \ 1*3^{-3}\\ &= \ \ 2*27+\ 1*9+\ 1*3+\ 0*1+1*(1/3)+0*(1/9)+1*(1/27)\\ &= \ \ \ \ 54+\ \ \ 9+\ \ \ 3+\ \ \ 0+\ \ \ 1/3+\ \ \ \ 0+\ \ \ 1/27\\ &= 66.\widehat{370}_{10}\end{aligned}$$

■

Solución al ejercicio 1.10 ¿Cuántos bytes son $45KB$?

$45KB$ son $45 * 10^3$ bytes. Recordar que se ha usado el prefijo decimal KB y no el binario KiB. ■

Solución al ejercicio 1.11 ¿Cuántos bits son $2KiB$?

$2KiB$ son $2 * 2^{10}$ bytes. Para obtener los bits hay que multiplicar por 8. Por lo tanto, $2KiB$ son $2 * 2^{10} * 8$ bits. En este ejercicio (a diferencia del anterior) se ha usado el prefijo binario KiB. ■

Solución al ejercicio 1.12 ¿Cuántos bits son $3MiB$?

$3MiB$ son $3 * 2^{20}$ bytes. Pero nos preguntan cuantos bits, por lo tanto $3MiB$ son $3 * 2^{20} * 8$ bits. ■

Solución al ejercicio 1.13 Si una red es capaz de transmitir a 3Mib por segundo ¿Cuántos bits transmitirá en un minuto?

$3Mib/s$ son $3 * 2^{20}$ bits por segundo. Como un minuto tiene 60 segundos, la respuesta correcta es $3 * 2^{20} * 60$ bits. ■

Solución al ejercicio 1.14 Si una red es capaz de transmitir a 2Kib por segundo ¿Cuántos bytes transmitirá en un minuto?

$2Kib/s$ son $2 * 2^{10}$ bits por segundo. En un minuto serán $2 * 2^{10} * 60$ bits. Pero el ejercicio pregunta cuantos bytes, por lo tanto habrá que dividir por 8. Entonces, la respuesta es $2 * 2^{10} * 60/8 = 15 * 2^{10}$ bytes. ■

2. El sistema de numeración binario

2.1 El sistema de numeración binario

El sistema de numeración binario es un sistema de numeración posicional de base 2 ($b = 2$) que se usa principalmente en el campo de las tecnologías de la información. En concreto, los ordenadores actuales usan el sistema binario para codificar la información ya sean números enteros o números reales (tanto positivos como negativos), o caracteres alfanuméricos.

El sistema binario tiene únicamente dos símbolos el '0' y el '1'. La Tabla 2.1 muestra los valores decimales de los dos símbolos del sistema binario. Para formar números usando el sistema binario, hemos de proceder tal como se ha mostrado en el capítulo anterior con el sistema decimal y el sistema inventado *Tri* de base 3 (ver apartados 1.1 y 1.2). Con 4 dígitos ($N = 4$), las dos primeras cantidades se expresarán como sigue: *cero* $\rightarrow 0000_2$ y *uno* $\rightarrow 0001_2$. Para la siguiente cantidad (*dos*), hemos de proceder de igual forma que con la cantidad *diez* en el sistema decimal o con la cantidad *tres* en el sistema *Tri*, es decir, hemos de reutilizar los símbolos que tenemos. De esta forma, las cantidades: *dos*, *tres*, *cuatro* y *cinco* se expresarán como sigue: *dos* $\rightarrow 0010_2$, *tres* $\rightarrow 0011_2$, *cuatro* $\rightarrow 0100_2$ y *cinco* $\rightarrow 0101_2$. Con el resto de cantidades se debe proceder siguiendo la misma estrategia. La figura 2.1 muestra este proceso.

El rango de valores que se puede expresar en binario se calcula aplicando la Fórmula 1.3. Por lo tanto, con $N = 4$, el rango de valores que se pueden expresar en binario es: $[0, 2^N - 1] = [0, 15]$. Como se puede comprobar, son necesarios muchos dígitos para expresar cantidades relativamente pequeñas en comparación al sistema decimal.

Dadas las características del sistema de numeración binario es muy sencillo saber si un número es par o impar, puesto que todos los números pares acaban en 0 y los impares en 1.

2.2 De binario a decimal

Como en cualquier sistema de numeración posicional, para pasar de binario a decimal no hay más que aplicar la Fórmula 1.1.

Tabla 2.1: Valores decimales de los símbolos del sistema binario

Símbolo	Valor decimal
0	0
1	1

Tabla 2.2: Principales potencias de 2

2^0	1_{10}	2^5	32_{10}	2^{10}	1024_{10}	2^{15}	$32\,768_{10}$
2^1	2_{10}	2^6	64_{10}	2^{11}	2048_{10}	2^{16}	$65\,536_{10}$
2^2	4_{10}	2^7	128_{10}	2^{12}	4096_{10}	2^{17}	$131\,072_{10}$
2^3	8_{10}	2^8	256_{10}	2^{13}	8192_{10}	2^{18}	$262\,144_{10}$
2^4	16_{10}	2^9	512_{10}	2^{14}	$16\,384_{10}$	2^{19}	$524\,288_{10}$

■ **Ejemplo 2.1** Calcula el equivalente en decimal del número 000101_2 expresado en binario.

$$\begin{aligned} 000101_2 &= 0*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0 \\ &= 0+ 0+ 0+ 2^2+ 0+ 2^0 \\ &= 0+ 0+ 0+ 4+ 0+ 1 \\ &= 5_{10} \end{aligned}$$

■

■ **Ejemplo 2.2** Calcula el equivalente en decimal del número 110_2 expresado en binario.

$$\begin{aligned} 110_2 &= 1*2^2+1*2^1+0*2^0 \\ &= 2^2+ 2^1+ 0 \\ &= 4+ 2+ 0 \\ &= 6_{10} \end{aligned}$$

■

Para la resolución de los siguientes ejercicios puede ser de utilidad consultar la Tabla 2.2 que muestra las veinte primeras potencias de dos.

Ejercicio 2.1 Calcula el equivalente en decimal del número 101011_2 expresado en binario. ■

Ejercicio 2.2 Calcula el equivalente en decimal del número 110111_2 expresado en binario. ■

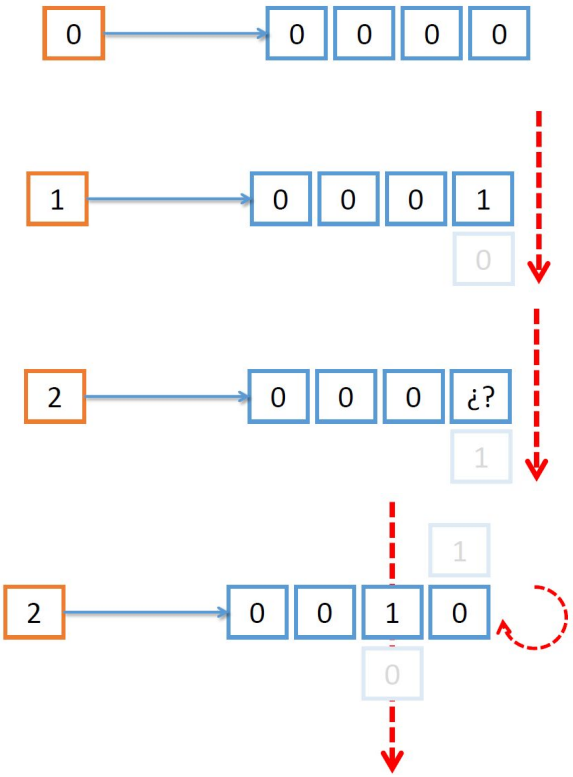


Figura 2.1: Ejemplo de como se forma en binario las cantidades *cero* y *uno*. Para la cantidad *dos* no existe un nuevo símbolo que podamos usar, por lo tanto reutilizamos los símbolos existentes en el dígito más a la derecha y usamos el siguiente en el de su izquierda.

Ejercicio 2.3 Calcula el equivalente en decimal del número 0111_2 expresado en binario. ■

Ejercicio 2.4 Calcula el equivalente en decimal del número 11110111_2 expresado en binario. ■

2.3 De decimal a binario

Existen dos métodos para convertir de decimal a cualquier base: 1) el método de las divisiones sucesivas por la base y 2) el método de las potencias de la base.

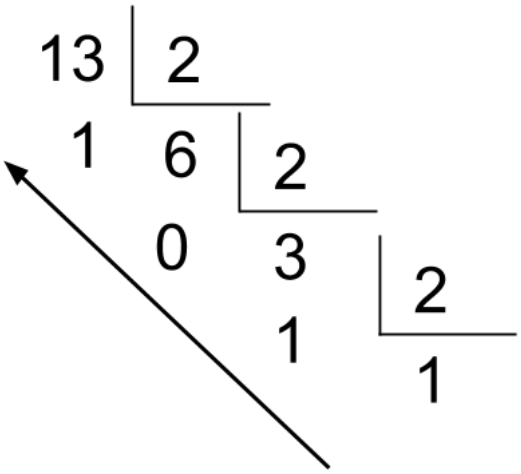
2.3.1 El método de las divisiones sucesivas por la base

El método de las divisiones sucesivas por la base consiste en dividir el número decimal que se desea convertir por la base a la que se quiere convertir, tantas veces como sea necesario, hasta que el resultado de la división sea menor a la base a la que se quiere convertir. El número buscado en la base b estará formado por los restos de las divisiones y por el último cociente.

Veamos un ejemplo de como convertir el número 13_{10} a binario ($b = 2$). En primer lugar hay que dividir el numero 13_{10} por la base a la que se quiere convertir (es decir

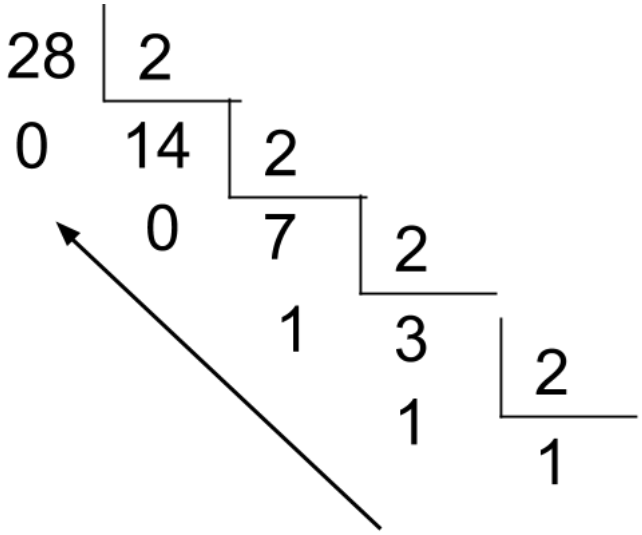
por 2). El resultado de la división es 6_{10} y el resto 1. Este resto será el primer dígito (por la derecha) del número binario resultado del proceso de conversión, es decir el dígito situado en la posición menos significativa del número. Por lo tanto, hasta el momento el número binario buscado es 1_2. El resultado de la división (6_{10}) se vuelve a dividir por 2, obteniendo 3_{10} con resto 0. Este nuevo resto se añadirá por la izquierda al número binario resultado, es decir por ahora el número buscado es 01_2. El último paso será dividir 3_{10} por 2, obteniendo el cociente 1 y el resto 1. El resto se añade al resultado por la izquierda (el resultado es por ahora $x_2 = 011_2$). No es necesario volver a dividir por la base puesto que el último cociente es menor que la base. El último cociente (que será siempre 1 o 0) se añade al resultado convirtiéndose en el dígito más significativo del número. Resumiendo el número 13_{10} es equivalente a 1101_2 en binario.

■ **Ejemplo 2.3** Calcula el equivalente en binario del número 13_{10} expresado en decimal mediante el método de las divisiones sucesivas por la base. Tal como muestra la siguiente figura y la explicación anterior, el resultado es 1101_2.



La flecha muestra el orden en el que hay que usar los símbolos 1 o 0 para componer el numero binario resultado. ■

■ **Ejemplo 2.4** Calcula el equivalente en binario del número 28_{10} expresado en decimal mediante el método de las divisiones sucesivas por la base. Tal como muestra la siguiente figura, el resultado es 11100_2.



■

Ejercicio 2.5 Convierte a binario el número 9_{10} mediante el método de las divisiones sucesivas por la base. ■

Ejercicio 2.6 Convierte a binario el número 33_{10} mediante el método de las divisiones sucesivas por la base. ■

Ejercicio 2.7 Convierte a binario el número 102_{10} mediante el método de las divisiones sucesivas por la base. ■

Ejercicio 2.8 Convierte a binario el número 220_{10} mediante el método de las divisiones sucesivas por la base. ■

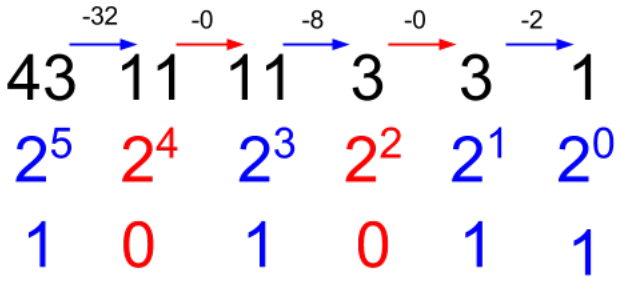
2.3.2 El método de las potencias de la base

El método de las potencias de la base es una forma alternativa, aunque más compleja, de convertir un número decimal en otro expresado en cualquier base (en este caso, en base 2).

Veamos un ejemplo de como convertir el número 43_{10} usando este método a binario. Para ello, lo primero que tenemos que hacer es buscar la potencia de 2 (ver Tabla 2.2) más grande sin superar al número que queremos convertir. En nuestro ejemplo, la potencia buscada es $2^5 = 32$, puesto que la siguiente $2^6 = 64$ supera al número que queremos convertir 43_{10}. Una vez encontrado la potencia de 2 más grande sin superar al número (2^5 en nuestro caso), sabremos que el número binario buscado, al que llamaremos x_2, tiene 6 dígitos ($N = 6$) y que se puede descomponer en potencias de 2 de la siguiente forma (ver Fórmula 1.1): $43_{10} = x_2[5]*2^5 + x_2[4]*2^4 + x_2[3]*2^3 + x_2[2]*2^2 + x_2[1]*2^1 + x_2[0]*2^0$. Además sabemos que $x_2[5] = 1$.

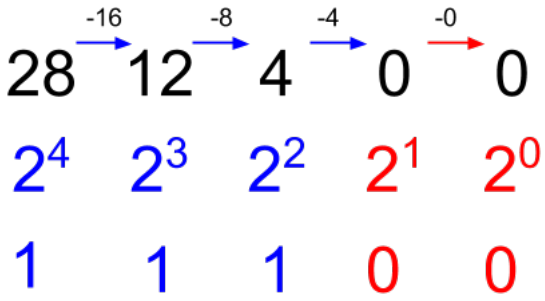
Ahora tenemos que averiguar el valor del resto de dígitos. Para ello, debemos restar a 43_{10} el valor de la potencia encontrada ($2^5 = 32$), obteniendo 11_{10}. El siguiente paso es repetir el proceso anterior buscando la potencia de 2 más grande pero menor a 11_{10}. En este caso, la potencia buscada es $2^3 = 8$ y por lo tanto, $x_2[3] = 1$. Es importante fijarse que la potencia encontrada no es $2^4 = 16$, que es justo la siguiente a 2^5, por lo que $x_2[4] = 0$. Dicho de otra forma, en la descomposición del número 43_{10} en potencias de 2, 2^4 no interviene. Seguimos el proceso, restando a 11_{10} el valor de la potencia encontrada ($2^3 = 8$) para obtener 3_{10}. La siguiente potencia de 2 será $2^1 = 2$, por lo tanto, $x_2[2] = 0$ (al no intervenir 2^2) y $x_2[1] = 1$. Al restar a 3_{10} la potencia de 2 encontrada ($2^1 = 2$) obtenemos 1, lo que hace que el proceso finalice y tanto $x_2[0] = 1$. El número binario buscado es: $x_2 = 101011_2$.

■ **Ejemplo 2.5** Calcula el equivalente en binario del número 43_{10} expresado en decimal mediante el método de las potencias de la base. Tal como muestra la siguiente figura y la explicación anterior, el resultado es 101011_2.



■

■ **Ejemplo 2.6** Calcula el equivalente en binario del número 28_{10} expresado en decimal mediante el método de las potencias de la base. Tal como muestra la siguiente figura, el resultado es 11100_2.



■

Ejercicio 2.9 Convierte a binario el número 17_{10} mediante el método de las potencias de la base. ■

Ejercicio 2.10 Convierte a binario el número 132_{10} mediante el método de las potencias de la base. ■

Ejercicio 2.11 Convierte a binario el número 200_{10} mediante el método de las potencias de la base. ■

2.4 Conversión de números reales

En esta sección, vamos a estudiar como se convierten números reales positivos de decimal a binario y viceversa. Tal como hemos comentado en el Capítulo 1 (ver Definición 1.2.1), todo número real x puede escribirse en la forma $y+z$ donde y es un entero (la parte entera de x) y z es un número real no negativo menor que 1, denominado la parte fraccionaria o parte fraccional de x. En binario también podemos expresar números reales. Por ejemplo, el número binario $x = 1001{,}101_2$ tiene parte entera $y = 1001_2$ y parte fraccionaria $z = 0{,}101_2$.

En realidad, los ordenadores no codifican los números reales usando la coma para separar la parte entera de la fraccionaria (como por ejemplo $101{,}11_2$). La razón principal es que en binario, solo es posible usar dos símbolos, el '1' y el '0', y por lo tanto, no podemos usar ningún símbolo para representar la coma. Los números reales se codifican

Tabla 2.3: Principales potencias negativas de 2

2^{-1}	$1/2^1 = 1/2 = 0{,}5_{10}$	2^{-6}	$1/2^6 = 1/64 = 0{,}015625_{10}$
2^{-2}	$1/2^2 = 1/4 = 0{,}25_{10}$	2^{-7}	$1/2^7 = 1/128 = 0{,}0078125_{10}$
2^{-3}	$1/2^3 = 1/8 = 0{,}125_{10}$	2^{-8}	$1/2^8 = 1/256 = 0{,}00390625_{10}$
2^{-4}	$1/2^4 = 1/16 = 0{,}0625_{10}$	2^{-9}	$1/2^9 = 1/512 = 0{,}001953125_{10}$
2^{-5}	$1/2^5 = 1/32 = 0{,}03125_{10}$	2^{-10}	$1/2^{10} = 1/1024 = 0{,}0009765625_{10}$

usando el formato *IEEE754* que será explicado en detalle en el Capítulo 5. Ahora bien, para convertir números reales expresados en decimal al formato binario *IEEE754* es necesario conocer el proceso de conversión de la parte entera y de la parte fraccionaria tal como se comenta en este capítulo.

2.4.1 De binario a decimal

Para convertir un número real de binario a decimal no hay más que aplicar la Fórmula 1.2 siendo de utilidad la Tabla 2.3 que muestra las 10 primeras potencias negativas de 2.

■ **Ejemplo 2.7** Calcula el equivalente en base 10 del número $101{,}101_2$ expresado en binario.

$$\begin{aligned} 101{,}101_2 = & \; 1*2^2+0*2^1+1*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3} \\ = & \quad 2^2+ \quad 0+ \quad 2^0+ \quad 2^{-1}+ \quad 0+ \quad 2^{-3} \\ = & \quad 4+ \quad 0+ \quad 1+ \quad 0{,}5+ \quad 0+ \; 0{,}125 \\ = & 5{,}625_{10} \end{aligned}$$

■

Ejercicio 2.12 Convierte a decimal el número real expresado en binario $1101{,}1101_2$. ■

Ejercicio 2.13 Convierte a decimal el número real expresado en binario $1001{,}01_2$. ■

2.4.2 De decimal a binario

El proceso de conversión de un número real decimal a binario se tiene que realizar en dos pasos. Veamos como ejemplo la conversión del número $5{,}25_{10}$. En el primer paso, se convierte la parte entera (5_{10} en el ejemplo) tal como hemos comentado en el Apartado 2.3. En este caso, $5_{10} \equiv 101_2$. En el segundo paso, se convierte la parte fraccionaria del número ($0{,}25_{10}$ en el ejemplo). En este caso, el equivalente en binario de $0{,}25_{10}$ es $0{,}01_2$ tal como veremos posteriormente. El resultado final es la suma de ambas cantidades en binario, es decir: $5{,}25_{10} \equiv 101{,}01_2$.

Definición 2.4.1 — Símbolo $\equiv$. El símbolo $\equiv$ se usa en matemáticas para expresar que una cantidad es equivalente a otra.

Para calcular el equivalente en binario de la parte fraccionaria de un número real, podemos usar dos variantes de los métodos explicados en el Apartado 2.3. Estos nuevos

métodos son: el método de las multiplicaciones (en vez de divisiones) sucesivas por la base y el método de las potencias negativas (en vez de las positivas) de la base. En este documento, nos centraremos únicamente en el método del las multiplicaciones sucesivas por la base por su simplicidad.

Para explicar como se convierte a binario la parte fraccionaria de un número real expresado en decimal, mediante el método de las multiplicaciones sucesivas por la base, usaremos el número $0{,}25_{10}$ como ejemplo. Este método consiste en realizar multiplicaciones sucesivas por la base (2 en binario) hasta que el resultado de la multiplicación sea igual o superior a 1. Las partes enteras resultantes de las multiplicaciones (que serán todas 0 salvo la última que será 1), serán los símbolos que usaremos para formar el número binario.

Comenzamos el proceso multiplicando $0{,}25_{10}$ por la base obteniendo un $0{,}5_{10}$. Puesto que el resultado es menor que 1, el primer dígito del número binario resultado después de la coma será un 0, por lo que de momento el resultado es $0{,}0_2$. El siguiente paso es multiplicar el resultado de la multiplicación anterior ($0{,}5_{10}$) por 2, obteniendo $1{,}0_{10}$. Como el resultado es igual o superior a 1, el proceso se detiene y el siguiente dígito a la derecha del número binario resultado será un 1. Es decir, el resultado es $0{,}01_2$. Por lo tanto, obtenemos $0{,}25_{10} \equiv 0{,}01_2$.

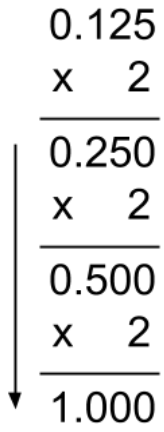
En el ejemplo anterior, el proceso se detiene puesto que la última multiplicación es igual a 1. Pero, ¿qué ocurriría si el resultado fuese mayor a 1? Veamos un ejemplo de este caso, convertiendo el número $0{,}4_{10}$ a binario. El proceso comienza multiplicando $0{,}4_{10}$ por la base, obtiendo $0{,}8_{10}$, por lo tanto el número binario buscado es hasta el momento $0{,}0_2$. Puesto que el resultado de la multiplicación es menor que 1, el proceso continúa multiplicando el resultado $0{,}8_{10}$ por 2, obteniendo $1{,}6_{10}$. El número buscado es ahora $0{,}01_2$. Tal como hemos explicado, al ser el resultado de la multiplicación igual o superior a 1, el proceso debería finalizar. Si así lo hicieramos estaríamos diciendo que $0{,}4_{10} \equiv 0{,}01_2$. Si convertimos el número binario $0{,}01_2$ a decimal (aplicando la Fórmula 1.2) obtenemos que realmente $0{,}01_2 = 0*2^0 + 0*2^{-1} + 1*2^{-2} = 2^{-2} = 0{,}25$. Es decir, $0{,}4_{10} \neq 0{,}25_{10}$. En realidad el resultado que hemos obtenido es una aproximación.

Si queremos mejorar esta aproximación, debemos continuar el proceso con el resultado de la última multiplicación menos la parte entera (el 1), es decir con $0{,}6_{10}$. Multiplicando $0{,}6_{10}$ por 2 obtenemos $1{,}2_{10}$. Con este nuevo paso, añadiremos un 1 al resultado que teniamos hasta el momento, obteniendo $0{,}011_2$. De nuevo podemos finalizar el proceso puesto que el resultado de la última multiplicación es superior a 1. Si así lo hacemos, obtenemos que $0{,}4_{10} \equiv 0{,}011_2$, pero (aplicando de nuevo la Fórmula 1.2) $0{,}011_2 \equiv 0{,}3750_{10}$ y por lo tanto $0{,}4_{10} \neq 0{,}3750_{10}$. Como podemos comprobar, hemos mejorado la aproximación, pero todavía no hemos encontrado el valor exacto.

Podemos mejorar la aproximación continuando con el mismo proceso. Por lo tanto, multiplicando $0{,}2_{10}$ por dos sucesivamente, se obtiene $0{,}4_{10}$, $0{,}8_{10}$ y $1{,}6_{10}$. Lo que resulta en añadir 001_2 al resultado, para obtener $0{,}011001_2$. Este número sigue sin ser exactamente igual a $0{,}4_{10}$, puesto que es $0{,}011001_2 \equiv 0{,}3906_{10}$. De nuevo, podríamos mejorar la aproximación repitiendo el proceso hasta que consiguieramos un resultado de la multiplicación exactamente igual a 1. En este ejemplo, esto no ocurre nunca, por lo que no es posible representar el número $0{,}4_{10}$ de forma exacta en binario. Cuantos más dígitos usemos en la parte fraccionaria, más exacto será el resultado obtenido.

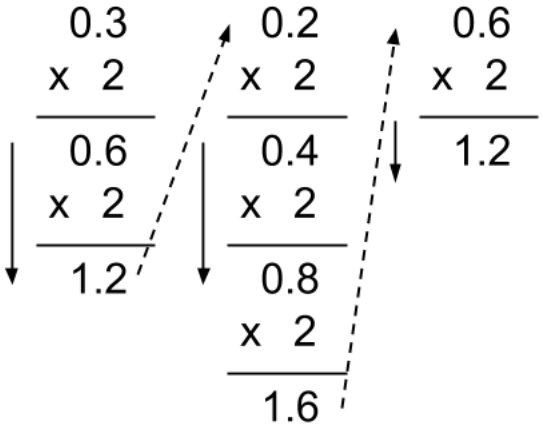
■ **Ejemplo 2.8** Convierte el número real $6{,}125_{10}$ a binario. El resultado es $110{,}001_2$. El

equivalente en binario a la parte entera 6_{10} es 110_2. Tal como muestra la siguiente figura, el equivalente a $0{,}125_{10}$ en binario es $0{,}001_2$



■

■ **Ejemplo 2.9** Convierte el número real $2{,}3_{10}$ a binario. No es posible obtener un resultado exacto, una aproximación usando 6 dígitos en la parte fraccionaria es $10{,}010011_2$ El equivalente en binario a la parte entera 2_{10} es 10_2. Tal como muestra la siguiente figura, una aproximación a $0{,}3_{10}$ en binario (usando únicamente 6 dígitos en la parte fraccionaria) es $0{,}010011_2$.



■

Ejercicio 2.14 Convierte el número real $5{,}75_{10}$ a binario. ■

Ejercicio 2.15 Convierte el número real $6{,}25_{10}$ a binario. ■

2.5 Operar en binario

2.5.1 Sumar en binario

La forma de sumar en binario es idéntica a la forma de sumar en decimal. De hecho, la suma en todos los sistemas de numeración posicionales se realiza de la misma forma. Para ello, es necesario tener una tabla que muestre que símbolo es el resultado de la suma de otros dos y si se produce o no acarreo. Por ejemplo, en decimal, dicha tabla nos

Tabla 2.4: Sumar en binario

Símbolo 1	Símbolo 2	Símbolo resultado	acarreo
0	0	0	no
0	1	1	no
1	0	1	no
1	1	0	si

dice que la suma del símbolo 1 y del símbolo 3 produce como resultado el símbolo 4, y que la suma del símbolo 7 y del símbolo 5 produce como resultado el símbolo 2 con acarreo.

Definición 2.5.1 — Acarreo. Acarreo significa lo mismo que comúnmente nombramos como *llevo 1*.

La Tabla 2.4 muestra los símbolos resultados y si se produce o no acarreo de todas las combinaciones posibles en binario. Como se puede comprobar la suma de los símbolos 1_2 y 1_2 produce como resultado el símbolo 0_2 con acarreo, o lo que es lo mismo, el número binario 10_2 (que es 2_{10} en decimal).

■ **Ejemplo 2.10** Suma los números en binarios 1001_2 y 0011_2.

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

■

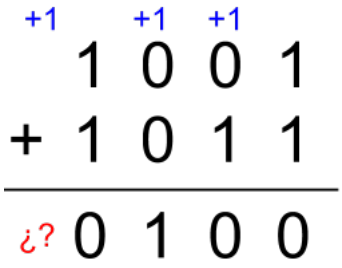
Cuando sumamos dos números, sea la que sea la base, es importante tener en cuenta el número de dígitos disponibles. Si la suma de los dos dígitos más significativos (los de más a la izquierda) produce acarreo, entonces el número resultado no se puede representar con ese número de dígitos, produciéndose lo que se conoce como *desbordamiento* (*overflow* en inglés).

Definición 2.5.2 — Desbordamiento aritmético. El desbordamiento aritmético [Wik14d] se produce cuando se desea almacenar un número cuya representación en binario necesita más dígitos que los que tiene el medio de almacenamiento. Por ejemplo, si el almacenamiento es de cuatro bits, este podrá almacenar desde 0000_2 hasta 1111_2, en binario. Si tenemos 1111_2 almacenado y sumamos 1_2, el resultado será $1\ 0000_2$, que no cabe por ser un código de cinco bits.

Definición 2.5.3

En el siguiente ejemplo se muestra una suma que produce desbordamiento.

■ **Ejemplo 2.11** Suma los números en binarios 1001_2 y 1011_2 con $N = 4$.



Como se puede comprobar en el ejemplo anterior, con 4 dígitos, no es posible sumar 1001_2 y 1011_2, pues produce desbordamiento. Si tuviéramos 8 dígitos, si sería posible y la solución seria: $00001001_2 + 00001011_2 = 00010100_2$ ■

Ejercicio 2.16 Suma los números en binario 0011_2 y 0101_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

Ejercicio 2.17 Suma los números en binario 0111_2 y 0100_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

Ejercicio 2.18 Suma los números en binario 1100_2 y 0111_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

2.5.2 Restar en binario

Al igual que con la suma, realizar restas en binario es un proceso idéntico a realizar restas en cualquier otra base (como en decimal). Para ello, es necesario tener una tabla que muestre que símbolo es el resultado de la resta de otros otros y si se produce o no acarreo. Por ejemplo, en decimal, dicha tabla nos diría que el resultado de restar al símbolo 5_{10} el símbolo 2_{10} es el símbolo 3_{10}. De forma similar, el resultado de restar al símbolo 4_{10} el símbolo 6_{10} es 8_{10} con acarreo. Esto último es posible, siempre que a la izquierda del 4_{10} exista otro símbolo diferente al 0_{10}, es decir, que en realidad estemos restando, por ejemplo, $14_{10} - 6_{10}$. Si no fuera ese el caso, no se podría restar.

Definición 2.5.4 — Minuendo y substraendo. En una resta están involucrados dos números llamados minuendo y subtraendo. El minuendo es el número mayor y se sitúa en la parte de arriba de la resta. El substraendo es la cantidad que queremos quitar al minuendo y se sitúa en la parte inferior de la resta. Si el substraendo es mayor que el minuendo, el resultado es un número negativo.

Hay que tener en cuenta que el acarreo en la resta no es igual al acarreo en la suma, puesto que el *llevo 1* se suma al número de abajo (substraendo) y no al de arriba (minuendo), como ocurre en la suma. También podriamos decir, de forma equivalente, que el acarreo en la resta es -1 (*llevo -1*) y que se suma al número de arriba.

La Tabla 2.5 muestra los símbolos resultados y si se produce o no acarreo de todas las combinaciones posibles en binario. Como se puede comprobar la resta de los símbolos 0_2 y 1_2 produce como resultado el símbolo 1_2 con acarreo.

■ **Ejemplo 2.12** Resta los números en binarios 1001_2 y 0011_2.

Tabla 2.5: Restar en binario

Símbolo 1	Símbolo 2	Símbolo resultado	acarreo
0	0	0	no
0	1	1	si
1	0	1	no
1	1	0	no

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

■

Ejercicio 2.19 Resta los números en binario 1011_2 y 0101_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

Ejercicio 2.20 Resta los números en binario 1010_2 y 0110_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

Al igual que en decimal, para poder realizar la resta, es imprescindible que el minuendo sea superior al subtraendo. En caso contrario no se puede realizar la resta. Más adelante introduciremos los números negativos con lo que sí será posible realizar este tipo de operaciones.

■ **Ejemplo 2.13** Resta los números en binarios 1001_2 y 1011_2.
No es posible realizar esta resta puesto que el minuendo es menor al subtraendo. ■

Ejercicio 2.21 Resta los números en binario 0010_2 y 0110_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal. ■

2.5.3 Multiplicar y dividir en binario

De nuevo, multiplicar y dividir en binario es un proceso similar a multiplicar y dividir en cualquier otra base. Vamos a centrarnos en el caso particular de las multiplicaciones y divisiones por la base. Por ejemplo, en decimal multiplicar por la base, (por 10_{10}), implica añadir un cero por la derecha, desplazando todos los dígitos hacia la izquierda. Obviamente, multiplicar n veces por la base implicará añadir n ceros por la derecha, desplazando los dígitos n posiciones hacia la izquierda. De forma contraria, dividir por la base implica añadir un cero por la izquierda, desplazando todos los dígitos hacia la derecha.

En binario el proceso es el mismo, multiplicar n veces por la base implica añadir n ceros a la derecha desplazando todos los dígitos hacia la izquierda (ver Figura 2.2). Por el contrario, dividir n veces por la base implica añadir n ceros a la izquierda desplazando

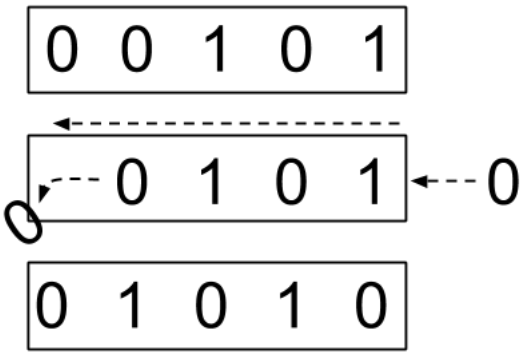


Figura 2.2: Proceso de multiplicación del número binario 00101_2 por una vez la base.

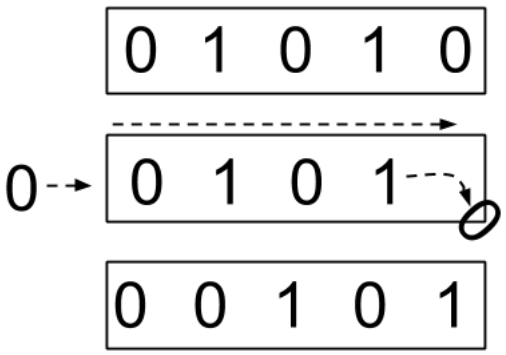


Figura 2.3: Proceso de división del número binario 01010_2 por una vez la base.

todos los dígitos hacia la derecha (ver Figura 2.3).

■ **Ejemplo 2.14** Multiplica el número binario 00111_2 por $2_{10} = 2^1$ (10_2 en binario) con $N = 5$. El resultado es 01110_2. Hemos añadido 1 cero a la derecha y desplazado 1 posición todos los dígitos hacia la izquierda. ■

■ **Ejemplo 2.15** Multiplica el número binario 000101_2 por $8_{10} = 2^3$ (1000_2 en binario) con $N = 6$. El resultado es 101000_2. Hemos añadido 3 ceros a la derecha y desplazado 3 posiciones todos los dígitos hacia la izquierda. ■

■ **Ejemplo 2.16** Divide el número binario 00111_2 por $2_{10} = 2^1$ (10_2 en binario) con $N = 5$. El resultado es 00011_2. Hemos añadido 1 cero a la izquierda y desplazado 1 posición todos los dígitos hacia la derecha. Como se puede comprobar, la división que estamos realizando es una división entera en la que solo obtenemos el cociente. ■

■ **Ejemplo 2.17** Divide el número binario 001101_2 por $8_{10} = 2^3$ (1000_2 en binario) con $N = 6$. El resultado es 000001_2. Hemos añadido 3 ceros a la izquierda y desplazado 3 posiciones todos los dígitos hacia la derecha. Igual que en el ejemplo anterior, la división que estamos realizando es una división entera en la que solo obtenemos el cociente. ■

Ejercicio 2.22 Multiplica el número binario 010101_2 una vez por la base con $N = 6$. ■

Ejercicio 2.23 Multiplica el número binario 000111_2 dos veces por la base con $N = 6$. ■

Ejercicio 2.24 Divide el número binario 010101_2 una vez por la base con $N = 6$. ■

Ejercicio 2.25 Divide el número binario 010111_2 dos veces por la base con $N = 6$. ■

De nuevo es muy importante tener en cuenta el número de dígitos para comprobar si se produce o no desbordamiento.

■ **Ejemplo 2.18** Multiplica el número binario 00111_2 por $8_{10} = 2^3$ (1000_2 en binario) con $N = 5$. La operación produce desbordamiento, pues el resultado 111000_2 no es posible expresarlo con 5 dígitos . ■

Ejercicio 2.26 Multiplica el número binario 010111_2 cuatro veces por la base con $N = 6$. ■

También es importante tener en cuenta, si estamos operando con números enteros o reales. Los ejemplos anteriores eran operaciones de números enteros. Las operaciones con números reales están fuera de los objetivos de este libro.

2.6 Solución a los ejercicios propuestos

Solución al ejercicio 2.1 Calcula el equivalente en decimal del número 101011_2 expresado en binario.

$$\begin{aligned} 101011_2 &= 1*2^5+0*2^4+1*2^3+0*2^2+1*2^1+1*2^0 \\ &= 2^5+ 0+ 2^3+ 0+ 2^1+ 2^0 \\ &= 32+ 0+ 8+ 0+ 2+ 1 \\ &= 43_{10} \end{aligned}$$

■

Solución al ejercicio 2.2 Calcula el equivalente en decimal del número 110111_2 expresado en binario.

$$\begin{aligned} 110111_2 &= 1*2^5+1*2^4+0*2^3+1*2^2+1*2^1+1*2^0 \\ &= 2^5+ 2^4+ 0+ 2^2+ 2^1+ 2^0 \\ &= 32+ 16+ 0+ 4+ 2+ 1 \\ &= 55_{10} \end{aligned}$$

■

Solución al ejercicio 2.3 Calcula el equivalente en decimal del número 0111_2

expresado en binario.

$$\begin{aligned} 0111_2 &= 0*2^3+1*2^2+1*2^1+1*2^0 \\ &= 0+ 2^2+ 2^1+ 2^0 \\ &= 0+ 4+ 2+ 1 \\ &= 7_{10} \end{aligned}$$

■

Solución al ejercicio 2.4 Calcula el equivalente en decimal del número 11110111_2 expresado en binario.

$$\begin{aligned} 11110111_2 &= 1*2^7+1*2^6+1*2^5+1*2^4+0*2^3+1*2^2+1*2^1+1*2^0 \\ &= 2^7+ 2^6+ 2^5+ 2^4+ 0+ 2^2+ 2^1+ 2^0 \\ &= 128+ 64+ 32+ 16+ 0+ 4+ 2+ 1 \\ &= 247_{10} \end{aligned}$$

■

Solución al ejercicio 2.5 Convierte a binario el número 9_{10} mediante el método de las divisiones sucesivas por la base.

La solución es 1001_2. ■

Solución al ejercicio 2.6 Convierte a binario el número 33_{10} mediante el método de las divisiones sucesivas por la base.

La solución es 100001_2. ■

Solución al ejercicio 2.7 Convierte a binario el número 102_{10} mediante el método de las divisiones sucesivas por la base.

La solución es 1100110_2. ■

Solución al ejercicio 2.8 Convierte a binario el número 220_{10} mediante el método de las divisiones sucesivas por la base.

La solución es 11011100_2. ■

Solución al ejercicio 2.9 Convierte a binario el número 17_{10} mediante el método de las potencias de la base.

La solución es 10001_2. ■

Solución al ejercicio 2.10 Convierte a binario el número 132_{10} mediante el método de las potencias de la base.

La solución es 10000100_2. ■

Solución al ejercicio 2.11 Convierte a binario el número 200_{10} mediante el método de las potencias de la base.

La solución es 11001000_2. ■

Solución al ejercicio 2.12 Convierte a decimal el número real expresado en binario $1101{,}1101_2$.

La solución es $13{,}8125_{10}$ ■

Solución al ejercicio 2.13 Convierte a decimal el número real expresado en binario $1001{,}01_2$. La solución es $9{,}25_{10}$. ■

Solución al ejercicio 2.14 Convierte el número real $5{,}75_{10}$ a binario.

La solución es $101{,}11_2$. ■

Solución al ejercicio 2.15 Convierte el número real $6{,}25_{10}$ a binario.

La solución es $110{,}01_2$ ■

Solución al ejercicio 2.16 Suma los números en binario 0011_2 y 0101_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

La solución es: $0011_2 + 0101_2 = 1000_2$. En decimal: $3 + 5 = 8$. ■

Solución al ejercicio 2.17 Suma los números en binario 0111_2 y 0100_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

La solución es: $0111_2 + 0100_2 = 1011_2$. En decimal: $7 + 4 = 11$. ■

Solución al ejercicio 2.18 Suma los números en binario 1100_2 y 0111_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

Con $N = 4$ no es posible sumar estos dos números pues se produce desbordamiento. Si N fuera 8, entonces si sería posible y la solución sería: $00001100_2 + 00000111_2 = 00010011_2$. En decimal: $12 + 7 = 19$. ■

Solución al ejercicio 2.19 Resta los números en binario 1011_2 y 0101_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

La solución es 0110_2. ■

Solución al ejercicio 2.20 Resta los números en binario 1010_2 y 0110_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

La solución es 0100_2. ■

Solución al ejercicio 2.21 Resta los números en binario 0010_2 y 0110_2 con $N = 4$. Comprueba el resultado convirtiendo los números a decimal.

No es posible realizar la resta puesto que el minuendo es más pequeño que el substraendo. ■

Solución al ejercicio 2.22 Multiplica el número binario 010101_2 una vez por la base con $N = 6$.

El resultado es 101010_2. ■

Solución al ejercicio 2.23 Multiplica el número binario 000111_2 dos veces por la base con $N = 6$.

El resultado es 011100_2. ■

Solución al ejercicio 2.24 Divide el número binario 010101_2 una vez por la base con $N = 6$.

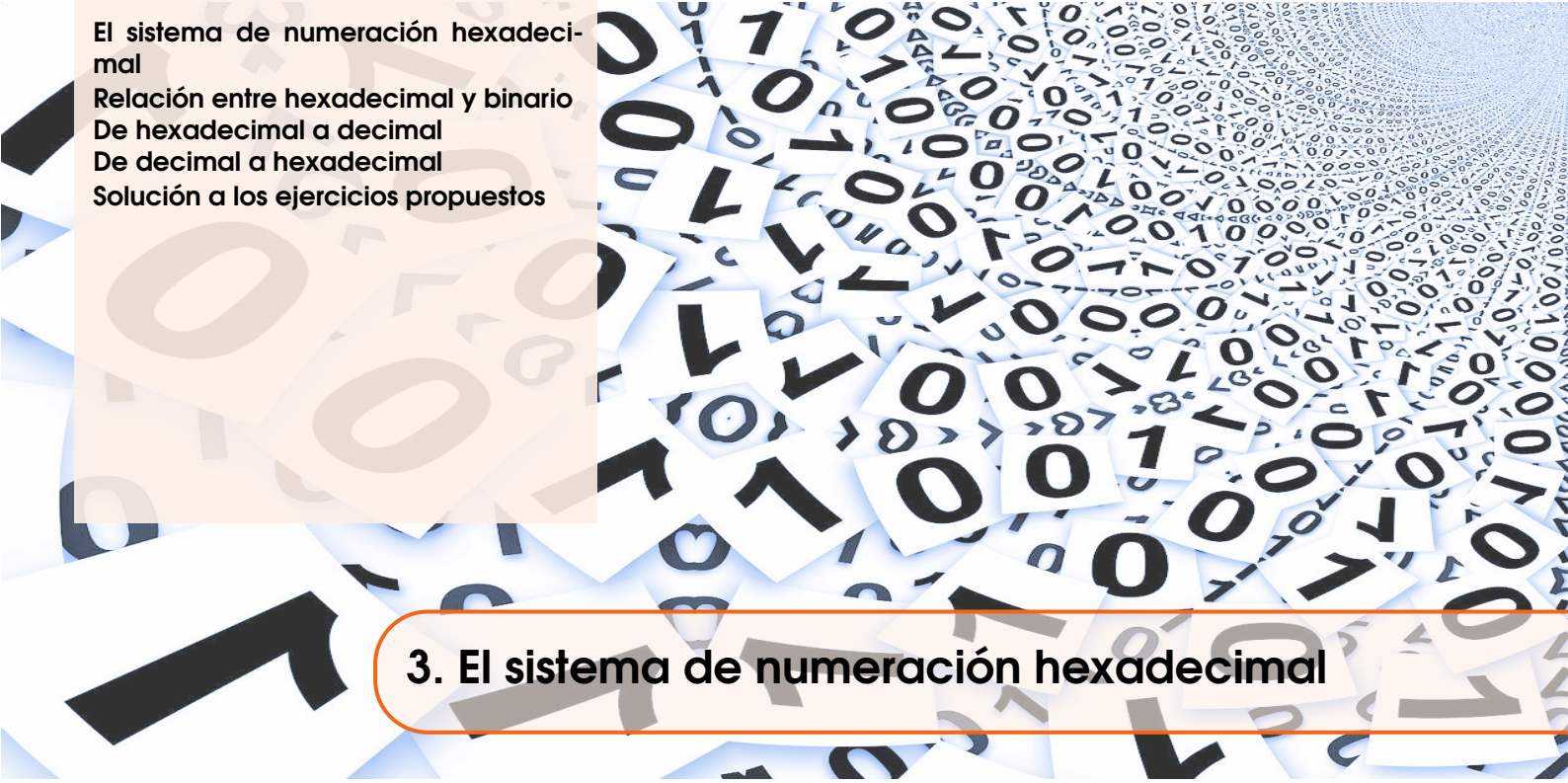
El resultado es 001010_2. ■

Solución al ejercicio 2.25 Divide el número binario 010111_2 dos veces por la base con $N = 6$.

El resultado es 000101_2. ■

Solución al ejercicio 2.26 Multiplica el número binario 010111_2 cuatro veces por la base con $N = 6$.

No es posible expresar el resultado de esta multiplicación con 6 dígitos. ■



3. El sistema de numeración hexadecimal

3.1 El sistema de numeración hexadecimal

El sistema de numeración hexadecimal es un sistema de numeración posicional con base 16 ($b = 16$) que, al igual que el sistema binario, se usa principalmente en el campo de las tecnologías de la información. El sistema hexadecimal tiene 16 símbolos: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E'y 'F'. La Tabla 3.1 muestra los valores decimales de los símbolos del sistema hexadecimal.

Para formar números usando este sistema de numeración, hemos de proceder de la misma forma que hemos hecho anteriormente con los sistemas decimal (ver Apartado 1.1), Tri (ver Apartado 1.2) y binario (ver Capítulo 2). Con $N = 4$ las primeras diez cantidades (del *cero* al *nueve*) se expresan igual que en decimal. Sin embargo, a partir de la cantidad *diez*, las cosas cambian, puesto que en este sistema si tenemos más símbolos para usar y no es necesario (aún) reutilizarlos como pasaba en el sistema decimal. De esta forma, la cantidad *diez* se expresará: $diez \rightarrow 000A_{16}$. Las siguientes cantidades se expresarán como sigue: $once \rightarrow 000B_{16}$, $doce \rightarrow 000C_{16}$, $trece \rightarrow 000D_{16}$, $catorce \rightarrow 000E_{16}$ y $quince \rightarrow 000F_{16}$. La Figura 3.1 muestra este proceso.

Para la cantidad *dieciséis* ocurre en hexadecimal lo mismo que pasaba en decimal con la cantidad *diez* o en binario con la cantidad *dos*, es decir, es necesario reutilizar símbolos. Así, las siguientes cantidades se expresarán como sigue: $dieciséis \rightarrow 0010_{16}$, $diecisiete \rightarrow 0011_{16}$, $dieciocho \rightarrow 0012_{16}$, $diecinueve \rightarrow 0013_{16}$, $veinte \rightarrow 0014_{16}$. Con el resto de cantidades se procederá siguiendo el mismo procedimiento. La Figura 3.1 muestra este proceso.

Como hemos comentado anteriormente, para mostrar que un número está expresado en una base concreta se usa como subíndice la base. Por lo tanto, el número $00AF_{16}$ está expresado en hexadecimal. Es común también el uso del prefijo $0x$ delante del número para expresar que el número está en hexadecimal, por ejemplo $0x00AF \equiv 00AF_{16}$.

El rango de valores que se pueden expresar en hexadecimal se calcula aplicando la Fórmula 1.3. Por lo tanto, con $N = 4$, el rango de valores que se puede expresar en hexadecimal es: $[0, 16^N - 1] = [0, 65536]$. Como se puede comprobar, al ser la base en el sistema hexadecimal mayor a la base en decimal, con menos dígitos se pueden expresar

Tabla 3.1: Valores decimales de los símbolos del sistema hexadecimal.

Símbolo	Valor decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

mayores cantidades.

3.2 Relación entre hexadecimal y binario

Quizás el lector se esté preguntado la razón por la que estamos presentando el sistema hexadecimal cuando, como se ha comentado anteriormente, los ordenadores usan el sistema binario para codificar la información. Para contestar a esta pregunta, resulta interesante fijarse en la Tabla 3.2 que muestra como un dígito en hexadecimal, corresponde a 4 dígitos en binario. Además, con un dígito en hexadecimal es posible representar todas las combinaciones posibles de 4 dígitos en binario. Recordemos que con 4 dígitos en binario es posible representar el rango $[0, 2^4 - 1] = [0, 15]$, es decir 16 números.

El sistema hexadecimal se puede usar para mostrar los números binarios de forma más compacta y, de esta forma, no tener que escribir tantos dígitos a la hora de expresar una cantidad. Por ejemplo, para expresar un número de 32 dígitos en binario, únicamente son necesarios 8 dígitos en hexadecimal.

Convertir un número binario a hexadecimal es muy sencillo, puesto que solo es necesario agrupar los dígitos binarios de 4 en 4 (empezando por la derecha) y buscar en la Tabla 3.2 el equivalente en hexadecimal.

▪ **Ejemplo 3.1** Calcula el equivalente en hexadecimal del número 00101101_2 expresado en binario.

Para ello agrupamos el numero en bloques de 4 dígitos 0010_2 y 1101_2. Ahora miramos la Tabla 3.2 el equivalente en hexadecimal, obteniendo $0010_2 \equiv 2_{16}$ y $1101_2 \equiv D_{16}$. Por lo tanto, el resultado será un número hexadecimal con dos dígitos: $00101101_2 \equiv$

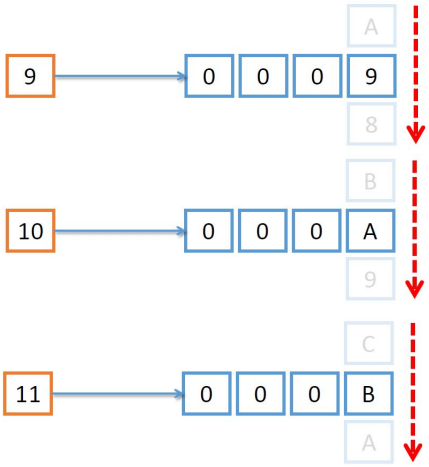


Figura 3.1: Ejemplo de como se forma en hexadecimal las cantidades *nueve*, *diez* y *once*. Para la cantidad *diez* existe un nuevo símbolo que podamos usar 'A'. Lo mismo ocurre con la cantidad *once*

.

$2D_{16}$. ■

■ **Ejemplo 3.2** Calcula el equivalente en hexadecimal del número 1001101011_2 expresado en binario.

Para resolver este ejercicio, debemos agrupar los dígitos en bloques de 4. Pero en este caso el número tiene 10 dígitos y 10 no es múltiplo de 4. Para resolver esta cuestión, lo más fácil es expresar el número con un número de dígitos múltiplo de 4. Para ello, simplemente añadiremos los ceros que sean necesarios a la izquierda. De esta forma, $1001101011_2 = 0010011010112$. Ahora ya podemos agrupar en bloques de 4 dígitos obteniendo: 0010_2, 0110_2 y 1011_2. Mirando la Tabla 3.2, obtenemos $0010_2 \equiv 2_{16}$, $0110_2 \equiv 6_{16}$ y $1011_2 \equiv B_{16}$. Por lo tanto, el resultado será un número en hexadecimal con tres dígitos: $1001101011_2 \equiv 26B_{16}$. ■

Ejercicio 3.1 Convierte a hexadecimal el número binario 100100111110_2. ■

Ejercicio 3.2 Convierte a hexadecimal el número binario 101100001010_2. ■

Ejercicio 3.3 Convierte a hexadecimal el número binario 01101000010_2. ■

Ejercicio 3.4 Convierte a hexadecimal el número binario 100011011_2. ■

Obviamente, para convertir de hexadecimal a binario no hay más que realizar el proceso contrario.

■ **Ejemplo 3.3** Calcula el equivalente en binario del número $A0F4_{16}$ expresado en hexadecimal.

Mirando la Tabla 3.2, obtenemos $A_{16} \equiv 1010_2$, $0_{16} \equiv 0000_2$, $F_{16} \equiv 1111_2$ y $4_{16} \equiv 0100_2$. Por lo tanto, el resultado será un número en binario con dieciséis dígitos:

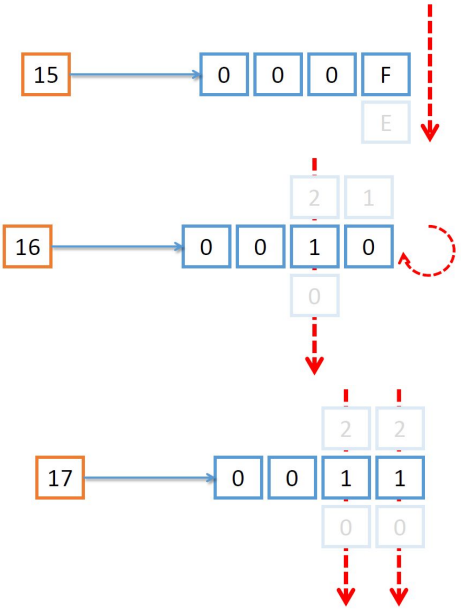


Figura 3.2: Ejemplo de como se forma en binario las cantidades *quince*, *dieciséis* y *diecisiete*. Para la cantidad *dieciséis* no existe un nuevo símbolo que podamos usar, por lo tanto reutilizamos los símbolos existentes en el dígito más a la derecha y usamos el siguiente en el de su izquierda.

$A0F4_{16} \equiv 1010000011110100_2$. ■

Ejercicio 3.5 Convierte a binario el número hexadecimal AFA_{16}. ■

Ejercicio 3.6 Convierte a binario el número hexadecimal $05D_{16}$. ■

3.3 De hexadecimal a decimal

Como en cualquier sistema de numeración posicional, para pasar de hexadecimal a decimal no hay más que aplicar la Fórmula 1.1. En este caso, hay que tener en cuenta que es necesario conocer las potencias de 16. Un truco para recordar las potencias de 16 es tener en cuenta que $16 = 2^4$. Por lo tanto, $16^1 = 2^4$, $16^2 = 2^4 * 2^4 = 2^8$, $16^3 = 2^4 * 2^4 * 2^4 = 2^{12}$, y siguiendo este procedimiento obtendremos que $16^4 = 2^{16}$, $16^5 = 2^{20}$, etc.

■ **Ejemplo 3.4** Convierte el número $BA74_{16}$ a decimal.

$$\begin{aligned} BA74_{16} = & \ 11*16^3+10*16^2+7*16^1+4*16^0 \\ = & 11*4096+10*256+\ 7*16+\ \ 2*1 \\ = & \ \ 45056+\ \ \ 2560+\ \ \ 112+\ \ \ \ 2 \\ = & \ 47\,730_{10} \end{aligned}$$

■

Tabla 3.2: Relación entre los sistemas de numeración binario, hexadecimal y decimal.

Binario	hexadecimal	decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Ejercicio 3.7 Convierte el número $C0A_{16}$ a decimal. ■

Ejercicio 3.8 Convierte el número $FE2_{16}$ a decimal. ■

Otra posibilidad es convertir primero el número hexadecimal a binario y luego aplicar la Fórmula 1.1 para calcular el número equivalente en decimal. Aunque es cierto que seguir esta estrategia supone más trabajo que aplicar directamente la Fórmula 1.1 al número hexadecimal, puede ser un proceso más rápido puesto que: por un lado, pasar de hexadecimal a binario es un proceso muy rápido (tal como hemos comentado en el apartado anterior), y por otro lado, aplicar la Fórmula 1.1 a un número binario es más sencillo que a un número hexadecimal por la simple cuestión de que las potencias de dos son más fáciles de recordar que las de 16.

■ **Ejemplo 3.5** Convierte el número BA_{16} a decimal. Primero lo convertimos en binario: $B_{16} \equiv 1011_2$ y $A_{16} \equiv 1010_2$. Por lo tanto, $BA_{16} \equiv 10111010_2$.

Ahora convertimos el número binario anterior a decimal:

$$\begin{aligned} BA_{16} = & \ 1*2^7+0*2^6+1*2^5+1*2^4+1*2^3+0*2^2+1*2^1+0*2^0 \\ = & 1*128+0*64+1*32+1*16+\ 1*8+\ 0*4+\ 1*2+\ 0*1 \\ = & \ \ 128+\ \ \ 0+\ \ 32+\ \ 16+\ \ \ 8+\ \ \ 0+\ \ \ 2+\ \ \ 0 \\ = & \ 186_{10} \end{aligned}$$

■

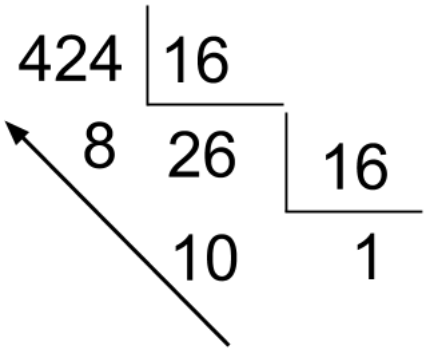
Ejercicio 3.9 Convierte el número 55_{16} a decimal convirtiéndolo primero en binario. ■

Ejercicio 3.10 Convierte el número $2E_{16}$ a decimal convirtiéndolo primero en binario. ■

3.4 De decimal a hexadecimal

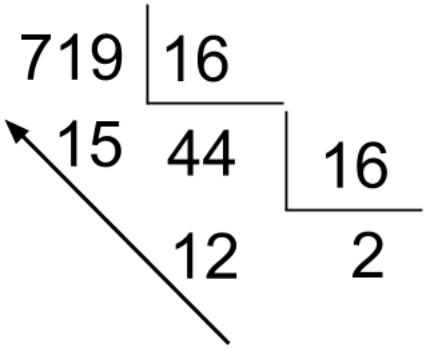
Para convertir una cantidad expresada en decimal a hexadecimal, podemos usar dos estrategias. La primera consiste en aplicar cualquiera de los dos métodos que hemos comentado en el Apartado 2.3: el método de las divisiones sucesivas por la base y el método de las potencias de la base. Vamos a ver dos ejemplos de como convertir de decimal a hexadecimal usando el método de las divisiones sucesivas por la base. Recordar que en este caso la base es 16.

■ **Ejemplo 3.6** Convierte el número 424_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base. Tal como muestra la siguiente figura el resultado es $1A8_{16}$. Recordemos que el valor decimal 10_{10} se corresponde con el símbolo A_{16} en hexadecimal (ver Tabla 3.1).



■

■ **Ejemplo 3.7** Convierte el número 719_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base. Tal como muestra la siguiente figura el resultado es $2CF_{16}$. Recordemos que los valores decimales 12_{10} y 15_{10} se corresponden con los símbolos C_{16} y F_{16} en hexadecimal, respectivamente (ver Tabla 3.1).



■

Ejercicio 3.11 Convierte el número 23_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base. ■

Ejercicio 3.12 Convierte el número 1034_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base. ■

Ejercicio 3.13 Convierte el número 524_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base. ■

La segunda estrategia consiste en convertir primero el número decimal a binario y luego convertir el número binario a hexadecimal. Al igual que en el proceso contrario, realizar la conversión de decimal a hexadecimal pasando por binario puede ser más rápido al ser las potencias de 2 más fáciles de recordar que las potencias de 16.

■ **Ejemplo 3.8** Convierte el número 220_{10} a hexadecimal convirtiendo primero a binario. 220_{10} es equivalente al número binario 11011100_2. Agrupamos de 4 en 4: $1101_2 \equiv D_{16}$ y $1100_2 \equiv C_{16}$. El resultado final es: DC_{16}. ■

Ejercicio 3.14 Convierte el número 110_{10} a hexadecimal convirtiendo primero a binario. ■

Ejercicio 3.15 Convierte el número 40_{10} a hexadecimal convirtiendo primero a binario. ■

3.5 Solución a los ejercicios propuestos

Solución al ejercicio 3.1 Convierte a hexadecimal el número binario 100100111110_2.
El resultado es: $93E_{16}$. ■

Solución al ejercicio 3.2 Convierte a hexadecimal el número binario 101100001010_2.
El resultado es: $B0A_{16}$. ■

Solución al ejercicio 3.3 Convierte a hexadecimal el número binario 01101000010_2.
En este caso, hay que tener en cuenta que el número que se pide convertir tiene un número de dígitos que no es múltiplo de 4. Para que así lo sea, es necesario añadir un cero a la izquierda obteniendo: 001101000010_2. El resultado es: 342_{16} ■

Solución al ejercicio 3.4 Convierte a hexadecimal el número binario 100011011_2.
En este caso, hay que tener en cuenta que el número que se pide convertir tiene un número de dígitos que no es múltiplo de 4. Para que así lo sea, es necesario añadir tres ceros a la izquierda obteniendo: 000100011011_2.
El resultado es: $11B_{16}$. ■

Solución al ejercicio 3.5 Convierte a binario el número hexadecimal AFA_{16}.
El resultado es: 101011111010_2. ■

Solución al ejercicio 3.6 Convierte a binario el número hexadecimal $05D_{16}$.
El resultado es: 000001011101_2. ■

Solución al ejercicio 3.7 Convierte el número $C0A_{16}$ a decimal.
El resultado es: 3082_{10} ■

Solución al ejercicio 3.8 Convierte el número $FE2_{16}$ a decimal.
El resultado es: 4066_{10} ■

Solución al ejercicio 3.9 Convierte el número 55_{16} a decimal convirtiéndolo primero en binario.
El resultado es: 85_{10}. ■

Solución al ejercicio 3.10 Convierte el número $2E_{16}$ a decimal convirtiéndolo primero en binario.
El resultado es: 46_{10}. ■

Solución al ejercicio 3.11 Convierte el número 23_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base.
El resultado es: 17_{16}. ■

Solución al ejercicio 3.12 Convierte el número 1034_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base.
El resultado es: $40A_{16}$. ■

Solución al ejercicio 3.13 Convierte el número 524_{10} a hexadecimal mediante el método de las divisiones sucesivas por la base.
El resultado es: $20C_{16}$. ■

Solución al ejercicio 3.14 Convierte el número 110_{10} a hexadecimal convirtiendo primero a binario.
El resultado es: $6E_{16}$. ■

Solución al ejercicio 3.15 Convierte el número 40_{10} a hexadecimal convirtiendo primero a binario.
El resultado es: 28_{16}. ■

4. Codificación de números enteros positivos

4.1 Introducción

Hasta el momento hemos estudiado los sistemas de numeración posicionales en general (Capítulo 1), el sistema de numeración binario (Capítulo 2) y el sistema de numeración hexadecimal (Capítulo 3). Como hemos comentado anteriormente en varias ocasiones, los sistemas de numeración binario y hexadecimal se usan principalmente en el campo de las tecnologías de la información.

En este capítulo, vamos a estudiar como los ordenadores codifican los números enteros positivos en binario usando un número concreto de bits. Recordemos que si un número se codifica usando N bits, entonces serán necesarios N dígitos en binario para representarlo. Además, podemos usar el sistema hexadecimal para representar dicho número de forma más compacta, teniendo en cuenta que, cada 4 dígitos en binario, tenemos su equivalente de 1 dígito en hexadecimal.

En este capítulo y en los siguientes, escribiremos un pequeño espacio cada cuatro dígitos binarios, empezando a agrupar por la derecha, para de esta forma, facilitar la lectura de los números binarios. Es decir, que en vez de escribir 111010010101_2, escribiremos $1110\ 1001\ 0101_2$, y que en vez de escribir 111010_2, escribiremos $11\ 1010_2$

4.2 Tipos de datos

Para que un ordenador realice una determinada tarea, es necesario transmitirle un conjunto de órdenes. A estas órdenes se les llama *Programas informáticos* [Wik14g].

Los programas informáticos son escritos por programadores usando un *Lenguaje de programación* [Wik15b]. Para realizar programas de alta calidad, es importante que el programador conozca bien los tipos de datos que puede usar y la codificación de los mismos. El desconocimiento de este tema, puede llevar a la realización de programas que puedan tener errores inesperados en el futuro. Por un lado, hemos de intentar elegir aquellos tipos de datos lo suficientemente grandes para que puedan almacenar las cantidades que requiera el problema concreto que se quiere resolver. Pero, por otro

lado, hay que tener en cuenta que cuanto más grande (en bits) sea el tipo de datos, más memoria ocupará y la memoria es un elemento finito en el ordenador.

Además del tamaño, es importante pensar si el concepto que queremos representar puede tener valores negativos o no. Por ejemplo, si queremos contar la cantidad de veces que ocurre algún evento, no tiene sentido usar números negativos. Usar tipos de datos que admitan negativos permite representar, con el mismo número de bits, la mitad de cantidades que si usamos un tipo de datos que solo permite números positivos. En el Capítulo 5 explicaremos la razón.

Veamos un ejemplo real. La popular plataforma web *Youtube* [1] eligió para codificar el contador de visitas de cada vídeo, un número entero de 32 bits pero que admitía tanto números positivos como negativos. Como ya sabemos con 32 bits (es decir, con $N = 32$ y $b = 2$) es posible representar $2^{32} = 4\ 294\ 967\ 296$ cantidades. Pero al usar números con signo, el rango se reduce a la mitad, por lo que el valor máximo que se puede representar es 2 147 483 647. Posiblemente, los diseñadores del sitio web nunca imaginaron que podría existir un vídeo cuyo número de visitas superase ese valor. Pero, el popular vídeo del cantante surcoreano *Psy* lo ha conseguido [Abc14]. La solución que han adoptado es cambiar la codificación del contador por un formato de 64 bits que permite, $2^{64} = 18\ 446\ 744\ 073\ 709\ 551\ 616$ posibles valores. Este número, parece ser una cantidad lo suficientemente grande para que nunca un vídeo pueda superar ese numero de visitas en el futuro. Sin embargo, esta decisión ha tenido una consecuencia importante. Ahora el almacenamiento del contador supone el doble de memoria para cada vídeo. Asumiendo que el número de vídeos incluidos en youtube es también una cifra astronómica, esta decisión ha debido suponer también un aumento considerable de la memoria necesaria para el funcionamiento de la plataforma. En concreto, ahora es necesario justo el doble de memoria.

La codificación de los principales tipos de datos dependen del lenguaje de programación seleccionado. Por ejemplo, el lenguaje de programación *C++* tiene tipos de datos para representar números enteros de 8, 16, 32 y 64 bits, tanto con signo como sin signo. Sin embargo, en el lenguaje *Java*, aunque también existen tipos de 8, 16, 32 y 64 bits, son siempre con signo [Doc15]. Por lo tanto, un entero de 32 bits en *C++* sin signo (es decir solo positivos) permitirá representar el doble de cantidades que su equivalente de 32 bits con signo en *C++* o en *Java* (que son siempre con signo).

Asumiendo que usamos codificación sin signo, es decir siempre positivos, los rangos de valores que se pueden expresar con 8, 16, 32 y 64 bits son los siguientes:

- 8 bits: $[0, 2^8 - 1] = [0, 255]$
- 16 bits: $[0, 2^{16} - 1] = [0, 65\ 535]$
- 32 bits: $[0, 2^{32} - 1] = [0, 4\ 294\ 967\ 295]$
- 64 bits: $[0, 2^{64} - 1] = [0, 18\ 446\ 744\ 073\ 709\ 551\ 615]$

En las siguientes secciones, vamos a explicar como se codifican números enteros positivos (es decir, números sin signo) usando tipos de datos de 8, 16, 32 y 64 bits. Dichos tipos de datos serán llamados *byte* (8 bits), *short* (16 bits), *int* (32 bits) y *long* (64 bits). Hemos elegido esos nombres pues son los que se usan en algunos de los lenguajes de programación más populares.

[1] `http://www.youtube.com`

4.3 Enteros positivos con 8 bits: *byte*

El tipo de datos *byte* tiene 8 bits, por lo que el rango de cantidades que se pueden expresar es: $[0, 2^8 - 1] = [0, 255]$. El número 256 y sucesivos no se pueden expresar con 8 bits.

Para expresar un número usando el tipo *byte*, se deberá convertir a binario tal como se ha comentado en el Apartado 2.3. Hay que tener en cuenta, que hay que usar los 8 bits, por lo que si el número binario resultante de la codificación tiene menos de 8 dígitos, se tendrá que rellenar con el símbolo '0' por la izquierda, tantas veces como sea necesario para que el número binario tenga 8 bits.

Es conveniente expresar el número binario resultado usando el sistema hexadecimal. De esta forma, el número de 8 bits podrá ser representado con únicamente 2 dígitos en hexadecimal. Recordemos que cada 4 dígitos en binario, podemos obtener su equivalente de 1 dígito en hexadecimal (ver Tabla 3.2).

■ **Ejemplo 4.1** Representa usando el tipo de datos *byte* el número 58_{10}. Expresa el resultado tanto en binario como en hexadecimal.

Aplicando, por ejemplo, el método de la divisiones sucesivas por la base (ver Apartado 2.3), obtenemos $58_{10} \equiv 11\ 1010_2$. Puesto que el número resultado tiene 6 dígitos, hemos de añadir dos símbolos '0' a la izquierda. El resultado final es: $0011\ 1010_2$ en binario. Su equivalente en hexadecimal es: $3A_{16}$. ■

Ejercicio 4.1 Representa usando el tipo de datos *byte* el número 18_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.2 Representa usando el tipo de datos *byte* el número 158_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.3 Representa usando el tipo de datos *byte* el número 228_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.4 Representa usando el tipo de datos *byte* el número 298_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

4.4 Enteros positivos con 16 bits: *short*

El tipo de datos *short* tiene 16 bits, por lo que el rango de cantidades que se pueden expresar es: $[0, 2^{16} - 1] = [0, 65\ 535]$. El número 65 536 y sucesivos no se pueden expresar con 16 bits.

Para expresar un número usando el tipo *short*, se deberá proceder de la misma forma que en el caso anterior, pero teniendo en cuenta que el número resultante deberá tener 16 dígitos, y por lo tanto, si fuera necesario, se tendrá que rellenar con ceros por la izquierda hasta llegar a ese número de dígitos. En este caso, el equivalente en hexadecimal tendrá 4 dígitos.

■ **Ejemplo 4.2** Representa usando el tipo de datos *short* el número 58_{10}. Expresa el

resultado tanto en binario como en hexadecimal.

Aplicando, por ejemplo, el método de la divisiones sucesivas por la base (ver Apartado 2.3), obtenemos $58_{10} \equiv 11\ 1010_2$. Puesto que el número resultado tiene 6 dígitos, hemos de añadir 10 símbolos '0' a la izquierda. El resultado final es: $0000\ 0000\ 0011\ 1010_2$ en binario. Su equivalente en hexadecimal es: $003A_{16}$. ■

Ejercicio 4.5 Representa usando el tipo de datos *short* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.6 Representa usando el tipo de datos *short* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.7 Representa usando el tipo de datos *short* el número $22\ 128_{10}$. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.8 Representa usando el tipo de datos *short* el número $70\ 000_{10}$. Expresa el resultado tanto en binario como en hexadecimal. ■

4.5 Enteros positivos con 32 bits: *int*

El tipo de datos *int* tiene 32 bits, por lo que el rango de cantidades que se pueden expresar es: $[0, 2^{32} - 1] = [0, 4\ 294\ 967\ 295]$. El número 4 294 967 296 y sucesivos no se pueden expresar con 32 bits.

Para expresar un número usando el tipo *int*, se deberá proceder de la misma forma que en los casos anteriores. Pero teniendo en cuenta que el número resultante deberá tener 32 dígitos, y por lo tanto, si fuera necesario, se tendrá que rellenar con ceros por la izquierda hasta llegar a ese número de dígitos. En este caso, el equivalente en hexadecimal tendrá 8 dígitos.

■ **Ejemplo 4.3** Representa usando el tipo de datos *int* el número 580_{10}. Expresa el resultado tanto en binario como en hexadecimal.

Aplicando, por ejemplo, el método de la divisiones sucesivas por la base (ver Apartado 2.3), obtenemos $580_{10} \equiv 10\ 0100\ 0100_2$. Puesto que el número resultado tiene 10 dígitos, hemos de añadir 22 símbolos '0' a la izquierda. El resultado final es: $0000\ 0000\ 0000\ 0000\ 0010\ 0100\ 0100_2$ en binario. Su equivalente en hexadecimal es: 00000244_{16}. ■

Ejercicio 4.9 Representa usando el tipo de datos *int* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.10 Representa usando el tipo de datos *int* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

4.6 Enteros positivos con 64 bits: *long*

El tipo de datos *long* tiene 64 bits, por lo que el rango de cantidades que se pueden expresar es: $[0, 2^{64} - 1] = [0, 18\ 446\ 744\ 073\ 709\ 551\ 615]$. El número 18 446 744 073 709 551 616 y sucesivos no se pueden expresar con 64 bits.

Para expresar un número usando el tipo *long*, se deberá proceder de la misma forma que en los casos anteriores. Pero teniendo en cuenta que el número resultante deberá tener 64 dígitos, y por lo tanto, si fuera necesario, se tendrá que rellenar con ceros por la izquierda hasta llegar a ese número de dígitos. En este caso, el equivalente en hexadecimal tendrá 16 dígitos.

■ **Ejemplo 4.4** Representa usando el tipo de datos *long* el número 580_{10}. Expresa el resultado tanto en binario como en hexadecimal.

Aplicando, por ejemplo, el método de la divisiones sucesivas por la base (ver Apartado 2.3), obtenemos $580_{10} \equiv 10\ 0100\ 0100_2$. Puesto que el número resultado tiene 10 dígitos, hemos de añadir 54 símbolos '0' a la izquierda. El resultado final es: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010\ 0100\ 0100_2$ en binario. Su equivalente en hexadecimal es: 0000000000000244_{16}. ■

Ejercicio 4.11 Representa usando el tipo de datos *long* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

Ejercicio 4.12 Representa usando el tipo de datos *long* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal. ■

4.7 Solución a los ejercicios propuestos

Solución al ejercicio 4.1 Representa usando el tipo de datos *byte* el número 18_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $0001\ 0010_2$ en binario y 12_{16} en hexadecimal. ■

Solución al ejercicio 4.2 Representa usando el tipo de datos *byte* el número 158_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $1001\ 1110_2$ en binario y $9E_{16}$ en hexadecimal. ■

Solución al ejercicio 4.3 Representa usando el tipo de datos *byte* el número 228_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $1110\ 0100_2$ en binario y $E4_{16}$ en hexadecimal. ■

Solución al ejercicio 4.4 Representa usando el tipo de datos *byte* el número 298_{10}. Expresa el resultado tanto en binario como en hexadecimal.

No es posible representar este número usando 8 bits. ■

Solución al ejercicio 4.5 Representa usando el tipo de datos *short* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $0000\ 0111\ 0100\ 1011_2$ en binario y $074B_{16}$ en hexadecimal. ■

Solución al ejercicio 4.6 Representa usando el tipo de datos *short* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $0000\ 0101\ 0100\ 1110_2$ en binario y $054E_{16}$ en hexadecimal. ■

Solución al ejercicio 4.7 Representa usando el tipo de datos *short* el número $22\ 128_{10}$. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $0101\ 0110\ 0111\ 0000_2$ en binario y 5670_{16} en hexadecimal. ■

Solución al ejercicio 4.8 Representa usando el tipo de datos *short* el número $70\ 000_{10}$. Expresa el resultado tanto en binario como en hexadecimal.

No es posible representar este número usando 16 bits. ■

Solución al ejercicio 4.9 Representa usando el tipo de datos *int* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es $0000\ 0000\ 0000\ 0000\ 0000\ 0111\ 0100\ 1011_2$ en binario y $0000074B_{16}$ en hexadecimal. ■

Solución al ejercicio 4.10 Representa usando el tipo de datos *int* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal.

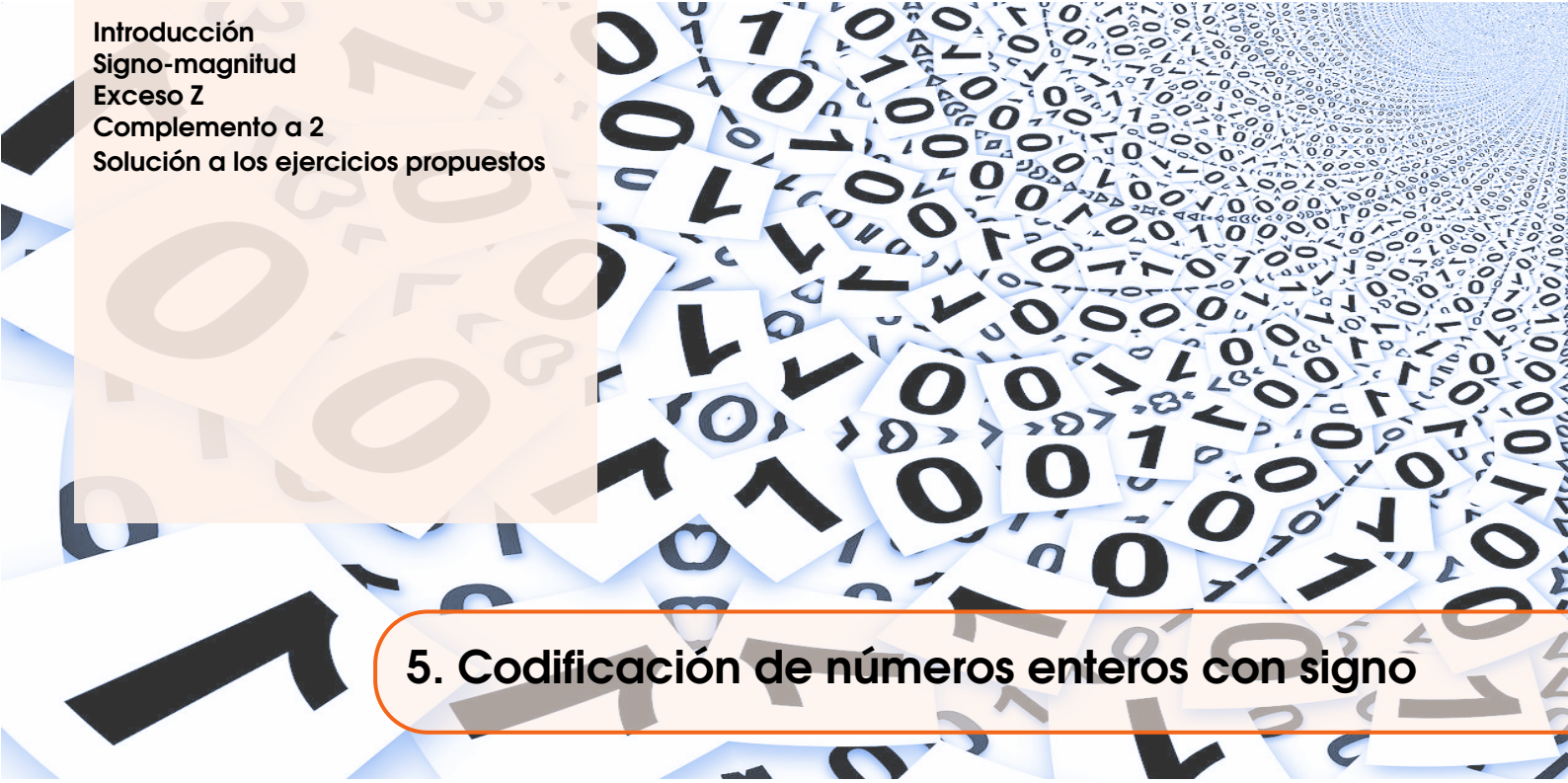
El resultado es $0000\ 0000\ 0000\ 0000\ 0000\ 0101\ 0100\ 1110_2$ en binario y $0000054E_{16}$ en hexadecimal. ■

Solución al ejercicio 4.11 Representa usando el tipo de datos *long* el número 1867_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es:
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111\ 0100\ 1011_2$ en binario y $000000000000074B_{16}$ en hexadecimal. ■

Solución al ejercicio 4.12 Representa usando el tipo de datos *long* el número 1358_{10}. Expresa el resultado tanto en binario como en hexadecimal.

El resultado es:
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101\ 0100\ 1110_2$ en binario y $000000000000054E_{16}$ en hexadecimal. ■



5. Codificación de números enteros con signo

5.1 Introducción

En el Capítulo 2, hemos comentado que no es posible restar dos números si el minuendo (el número de arriba) es menor que el substraendo (el número de abajo). Para poder realizar este tipo de operaciones, son necesarios los números negativos. En decimal, usamos el símbolo "-" delante del número para expresar que un número es negativo. Sin embargo, en binario únicamente podemos usar unos y ceros, por lo que no es posible usar otro símbolo adicional para expresar que un número es negativo.

Existen tres formas principales de expresar números con signo en binario: *Signo-magnitud*, *Exceso Z* y *Complemento a 2*. Esta última, es la forma en la que los ordenadores actuales codifican los números negativos.

Aunque cada método es diferente, el rango de números que se puede expresar es similar. En todos los casos, el mayor número que se puede expresar se reduce a la mitad con respecto a la representación de los números sin signo (enteros positivos). Dicho de forma coloquial, la mitad del *espacio* se usa para representar los números negativos y la otra mitad para los positivos. Por lo tanto, con el mismo número de bits, si se usa codificación con signo, será posible alcanzar un número aproximadamente la mitad de grande que si se usa codificación sin signo. Por ejemplo, con 8 bits sin signo, se puede representar el rango: $[0, 255]$, sin embargo, con signo (en Complemento a 2), el rango será: $[-128, 127]$.

Los procesadores actuales tienen operaciones con signo y sin signo. Si se usa operaciones sin signo, el procesador interpreta los bits como un número que solo puede ser entero positivo. Pero si se usa una operación con signo, entonces el procesador interpreta los bits como si fuera un número que puede ser positivo o negativo (codificado en *Complemento a 2*). Por lo tanto, es posible que un mismo conjunto de bits, pueda expresar dos cantidades diferentes dependiendo de si la operación es con o sin signo.

5.2 Signo-magnitud

La forma que parece más evidente de codificar un número con signo es usar un dígito para expresar si es positivo o negativo. En *Signo-magnitud*, los números se codifican usando el primer dígito (el más significativo) como signo y el resto para codificar el valor absoluto del número. El primer dígito será '0' para números positivos y '1' para números negativos. Al perder un bit, el rango se reduce a la mitad. Así, dado un número de dígitos N, en *Signo-magnitud* el rango de valores que se puede representar es:

$$\Delta = [-2^{N-1} - 1, +2^{N-1} - 1] \tag{5.1}$$

Por ejemplo, con 8 bits, el rango será [-127,+127]. El número positivo más grande será $0111\ 1111_2$ (127_{10}). De forma similar, el número negativo más pequeño será $1111\ 1111_2$ (-127_{10})

■ **Ejemplo 5.1** Expresa el número 27_{10} en binario con codificación *Signo-magnitud* con 8 bits. Como el número es positivo, el primer bit es 0. Para los otros 7, hemos de codificar el número 27_{10} con 7 bits, obteniendo: $001\ 1011_2$. Por lo tanto, 27_{10} es equivalente, en *Signo-magnitud* con 8 bits, a: $0001\ 1011_2$ o $1B_{16}$ en hexadecimal. ■

■ **Ejemplo 5.2** Expresa el número -37_{10} en binario con codificación *Signo-magnitud* con 8 bits. Como el número es negativo, el primer bit es 1. Para los otros 7, hemos de codificar el valor absoluto del número -37_{10}, es decir, hemos de codificar el número 37_{10} con 7 bits, obteniendo: $010\ 0101_2$. Por lo tanto, -37_{10} es equivalente, en *Signo-magnitud* con 8 bits, a: $1010\ 0101_2$ o $A5_{16}$ en hexadecimal. ■

■ **Ejemplo 5.3** Expresa el número -60_{10} en binario con codificación *Signo-magnitud* con 8 bits. Como el número es negativo, el primer bit es 1. Para los otros 7, hemos de codificar el número 60_{10} con 7 bits, obteniendo: $011\ 1100_2$. Por lo tanto, -60_{10} es equivalente, en *Signo-magnitud* con 8 bits, a: $1011\ 1100_2$ o BC_{16} en hexadecimal. ■

■ **Ejemplo 5.4** Expresa el número 130_{10} en binario con codificación *Signo-magnitud* con 8 bits. No es posible codificar este número en *Signo-magnitud* con 8 bits pues está fuera de rango. Hay que tener en cuenta que si estuviéramos codificando como un número sin signo, entonces si que se podría representar con 8 bits. ■

Tal como se ha comentado anteriormente, los procesadores actuales tienen operaciones con signo y sin signo. Imaginemos que queremos sumar los números $1000\ 0011_2$ y $0000\ 0001_2$. Si la operación es sin signo, estaremos sumando los números 131_{10} y 1_{10}, puesto que $1000\ 0011_2 \equiv 131_{10}$ y $0000\ 0001_2 \equiv 1_{10}$. El resultado será $1000\ 0100_2$ (132_{10}). Si la operación es con signo (asumiendo que los números están codificados en *Signo-magnitud*), estaremos sumando los números -3_{10} y 1_{10}, puesto que $1000\ 0011_2 \equiv -3_{10}$ y $0000\ 0001_2 \equiv 1_{10}$. El resultado será $1000\ 0010_2$ (-2_{10}).

Una propiedad importante que tienen los número codificados en *Signo-magnitud* es que el primer bit (el más significativo) nos da una indicación del signo del número. Así, todos los números negativos empiezan por 1 y los positivos por 0.

Un problema que tiene la representación *Signo-magnitud* es la doble representación del cero, puesto que tanto $0000\ 0000_2$ y$1000\ 0000_2$ se pueden usar para representar el cero.

Posiblemente, el lector se esté preguntado cuál es la razón por la que no se usa esta forma de representar los números con signo, siendo una forma muy intuitiva y simple. La razón reside en que la representación de los números con signo en *Signo-magnitud* implica dificultades de diseño en los procesadores a la hora de realizar las operaciones aritméticas. Dicho de otra forma, los circuitos necesarios para tener en cuenta el signo y para darse cuenta de si la operación es una suma o una resta, complican mucho el diseño de los procesadores. Existen otras formas de representar números con signo que ayudan a que el diseño de los procesadores sea lo más simple que sea posible (y por lo tanto rápido).

Ejercicio 5.1 Expresa el número -120_{10} en binario con codificación *Signo-magnitud* con 8 bits. ■

Ejercicio 5.2 Expresa el número 60_{10} en binario con codificación *Signo-magnitud* con 8 bits. ■

Ejercicio 5.3 Expresa el número -10_{10} en binario con codificación *Signo-magnitud* con 8 bits. ■

Ejercicio 5.4 Expresa el número -130_{10} en binario con codificación *Signo-magnitud* con 8 bits. ■

Ejercicio 5.5 ¿Cuál será el resultado de sumar los números $1010\ 1010_2$ y $0000\ 1110_2$ teniendo en cuenta el signo (con codificación *Signo-magnitud* de 8 bits)? ¿Y sin tenerlo en cuenta? ■

Ejercicio 5.6 ¿Cuál será el resultado de sumar los números $1010\ 0010_2$ y $0010\ 1110_2$ teniendo en cuenta el signo (con codificación *Signo-magnitud* de 8 bits)? ¿Y sin tenerlo en cuenta? ■

Para convertir un número expresado en *Signo-magnitud* a decimal hay que primero mirar el bit más significativo (el de más a la izquierda). Si es '0' , el número es positivo y si es '1' es negativo. El número en cuestión será el resultado de convertir a decimal los 7 bits menos significativos tal como se ha comentado anteriormente.

■ **Ejemplo 5.5** Expresa en decimal el número de 8 bits codificado en *Signo-magnitud* $1100\ 1101_2$. En este caso el número es negativo puesto que comienza con el símbolo '1' . Puesto que $100\ 1101_2 \equiv 77_{10}$, la solución al problema es $1100\ 1101_2 \equiv -77_{10}$. ■

5.3 Exceso Z

Otra forma de codificar los números con signo es la codificación en *Exceso Z*. Esta técnica consiste en sumarle un número fijo al número que queremos expresar, de forma que no tengamos números negativos. A la cantidad que sumamos le llamamos *Z*. El

valor de Z se obtiene de la siguiente forma:

$$Z = 2^{N-1} - 1 \tag{5.2}$$

Tal como muestra la ecuación anterior, Z es el valor absoluto del número negativo más pequeño que se puede expresar con un determinado número de bits. Con 8 bits, $Z = 127$. Por lo tanto, para representar un número cualquiera, le deberemos sumar $Z = 127$, y luego expresar ese número en binario. De esta forma, el número más pequeño que se puede representar -127 se convertirá en 0 al sumar Z.

■ **Ejemplo 5.6** Expresa el número 50_{10} en *Exceso Z* con 8 bits. Puesto que tenemos que representar el número con 8 bits, Z será igual a $Z = 2^{8-1} - 1 = 127$. Por lo tanto, tenemos que sumar 127_{10} al número que queremos representar, obteniendo $50_{10} + 127_{10} = 177_{10}$. El siguiente paso es representar 177_{10} en binario con 8 bits, obteniendo $1011\ 0001_2$. ■

■ **Ejemplo 5.7** Expresa el número -50_{10} en *Exceso Z* con 8 bits. Tenemos que sumar 127_{10} al número que queremos representar, obteniendo $-50_{10} + 127_{10} = 77_{10}$. El siguiente paso es representar 77_{10} en binario con 8 bits, obteniendo $0100\ 1101_2$. ■

Ejercicio 5.7 Expresa el número 112_{10} en *Exceso Z* con 8 bits. ■

Ejercicio 5.8 Expresa el número -80_{10} en *Exceso Z* con 8 bits. ■

Ejercicio 5.9 Expresa el número -10_{10} en *Exceso Z* con 8 bits. ■

Ejercicio 5.10 Expresa el número 67_{10} en *Exceso Z* con 8 bits. ■

Como se puede comprobar en los ejemplos anteriores, en la representación *Exceso Z*, los números negativos empiezan por 0 los positivos por 1, al contrario de como ocurría en *Signo-magnitud* (y también ocurrirá en *Complemento a 2*). Sin embargo, una cuestión positiva es que el cero solo tiene una única representación, al contrario que ocurría en el caso de la representación *Signo-magnitud*.

■ **Ejemplo 5.8** Expresa el número 0_{10} en *Exceso Z* con 8 bits. Tenemos que sumar 127_{10} al número que queremos representar, obteniendo $0_{10} + 127_{10} = 127_{10}$. El siguiente paso es representar 127_{10} en binario con 8 bits, obteniendo $0111\ 1111_2$. ■

Para expresar número con signo en *Exceso Z* dado un número de bits diferente a 8, debemos calcular cuanto es Z mediante la Fórmula 5.2.

■ **Ejemplo 5.9** Expresa el número -50_{10} en *Exceso Z* con 16 bits. Puesto que tenemos que representar el número con 16 bits, Z será igual a $Z = 2^{16-1} - 1 = 32\ 767$. Por lo tanto, tenemos que sumar $32\ 767_{10}$ al número que queremos representar, obteniendo $-50_{10} + 32\ 767_{10} = 32\ 717_{10}$. El siguiente paso es representar $32\ 717_{10}$ en binario con 16 bits, obteniendo $0111\ 1111\ 1100\ 1101_2$. ■

■ **Ejemplo 5.10** Expresa el número -50_{10} en *Exceso Z* con 32 bits. Puesto que tenemos que representar el número con 32 bits, Z será igual a $Z = 2^{32-1} - 1 = 2\ 147\ 483\ 647$. Por lo tanto, tenemos que sumar $2\ 147\ 483\ 647_{10}$ al número que queremos representar,

obteniendo $-50_{10}+2\,147\,483\,647_{10}=2\,147\,483\,597_{10}$. El siguiente paso es representar $2\,147\,483\,597_{10}$ en binario con 32 bits, obteniendo $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100\ 1101_2$.
■

Ejercicio 5.11 Expresa el número 112_{10} en *Exceso Z* con 16 bits. ■

Ejercicio 5.12 Expresa el número -80_{10} en *Exceso Z* con 16 bits. ■

Ejercicio 5.13 Expresa el número -10_{10} en *Exceso Z* con 32 bits. ■

Ejercicio 5.14 Expresa el número 67_{10} en *Exceso Z* con 32 bits. ■

Dado un número de dígitos N, en *Exceso Z* el rango de valores que se puede representar es:

$$\Delta = [-Z, +2^N - 1 - Z] \tag{5.3}$$

Por ejemplo, con 8 bits, el rango será [-127,128]. El número positivo más grande será $1111\ 1111_2$ ($128_{10}=255_{10}-127_{10}$). De forma similar, el número negativo más pequeño será $0000\ 0000_2$ ($-127_{10}=0_{10}-127_{10}$).

Para convertir un número expresado en Exceso Z a decimal, debemos conocer el valor Z con el que se codificó el número y realizar el proceso contrario. Es decir, primero obtener el equivalente en binario, como si fuera un número entero positivo, y luego restarle Z.

■ **Ejemplo 5.11** Expresa en decimal el número de 8 bits codificado en *Exceso Z* $0100\ 1101_2$. En este caso $Z=127$. El primer paso es obtener el equivalente de $0100\ 1101_2$ en decimal, que es el número 77_{10}. Ahora restamos $Z=127$, para obtener el valor buscado, obteniendo $77_{10}-127_{10}=-50_{10}$. ■

■ **Ejemplo 5.12** Expresa en decimal el número de 16 bits codificado en *Exceso Z* $0111\ 0001\ 1011\ 0101_2$. En este caso $Z=32\,767$. El primer paso es obtener el equivalente de $0111\ 0001\ 1011\ 0101_2$ en decimal, que es el número $29\,109_{10}$. Ahora restamos $Z=32\,767$, para obtener el valor buscado, obteniendo $29\,109_{10}-32\,767_{10}=-3658_{10}$.
■

Ejercicio 5.15 Expresa en decimal el número de 8 bits codificado en *Exceso Z* $1000\ 0100_2$. ■

Ejercicio 5.16 Expresa en decimal el número de 8 bits codificado en *Exceso Z* $0110\ 1011_2$. ■

Ejercicio 5.17 Expresa en decimal el número de 16 bits codificado en *Exceso Z* $0000\ 1010\ 0111\ 0100_2$. ■

Ejercicio 5.18 Expresa en decimal el número de 16 bits codificado en *Exceso Z* $0111\ 1111\ 1010\ 0100_2$. ■

Aunque el uso de este sistema resuelve aparentemente el problema de la representación de los números con signo, su uso implica dificultades a la hora de realizar operaciones matemáticas. Por ejemplo, al sumar dos números en *Exceso Z*, el resultado estará en *Exceso* $2Z$, por lo tanto, el procesador debe conocer esta circunstancia restando al resultado $2 * Z$. Este tipo de complejidades añade dificultad al diseño de los procesadores y por esta razón no se usa de forma general para representar números con signo. Sin embargo, si que usará para representar exponentes en la notación IEEE754, tal como veremos en el Capítulo 6.

5.4 Complemento a 2

La solución final que se ha adoptado para representar números con signo es conocida como *Complemento a 2*. En este caso, el proceso de representación es diferente dependiendo del signo del número. Los números positivos se expresan en binario de forma igual a como ya se ha explicado en el Capítulo 4. Sin embargo, para convertir un número decimal negativo a *Complemento a 2* hay que seguir 3 sencillos pasos:

1. Convertir el valor absoluto del número a binario como si fuera positivo.
2. Invertir los ceros por unos y viceversa.
3. Sumar 1 al resultado obtenido.

■ **Ejemplo 5.13** Expresa el número 56_{10} en *Complemento a 2* con 8 bits. Puesto que es positivo, el resultado es $0011\ 1000_2$. ■

■ **Ejemplo 5.14** Expresa el número -56_{10} en *Complemento a 2* con 8 bits. Al ser negativo, hay que aplicar los tres pasos. Paso 1: representar 56_{10} en binario con 8 bits, siendo el resultado: $0011\ 1000_2$. Paso 2: Invertir los ceros por unos y viceversa, obteniendo $1100\ 0111_2$. Paso 3: sumar 1, siendo el resultado final $1100\ 1000_2$. Por lo tanto, $56_{10} \equiv 1100\ 1000_2$ en *Complemento a 2*. ■

Como se ha comprobado en los dos ejemplos anteriores, en *Complemento a 2*, los números positivos empiezan con 0 y los negativos por 1. Además, existe una única representación del cero.

El rango que se puede expresar es el siguiente:

$$\Delta = [-2^{N-1}, 2^{N-1} - 1] \tag{5.4}$$

La Tabla 5.1 muestra los posibles valores que se pueden representar con $N = 4$. El número más pequeño es $-8_{10} \equiv 1000_2$, y el más grande $7_{10} \equiv 0111_2$.

Ejercicio 5.19 Expresa 34_{10} en *Complemento a 2* con 8 bits. ■

Ejercicio 5.20 Expresa -120_{10} en *Complemento a 2* con 8 bits. ■

Tabla 5.1: Cantidades que se pueden expresar con $N = 4$ en *Complemento a 2*

Decimal	Binario
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Ejercicio 5.21 Expresa 120_{10} en *Complemento a 2* con 8 bits. ■

Ejercicio 5.22 Expresa -8_{10} en *Complemento a 2* con 8 bits. ■

Es importante fijarse como se representan los números positivos y negativos cuando aumentamos el número de dígitos a usar. Por ejemplo, con $N = 4$, el número 6_{10} se representa como 0110_2, con $N = 8$ como $0000\ 0110_2$ y con $N = 16$ como $0000\ 0000\ 0000\ 0110_2$. Como se puede comprobar, la diferencia entre las tres representaciones del número 6_{10} es el número de ceros que se añaden a la izquierda. Sin embargo, con $N = 4$, el número -6_{10} se representa como 1010_2, con $N = 8$, como $1111\ 1010_2$ y con $N = 16$ como $1111\ 1111\ 1111\ 1010_2$. En este caso, los cuatro bits más a la derecha son iguales, y la diferencia entre las tres representaciones es el número de unos a la izquierda. Resumiendo, dado un número cualquiera expresado en binario con un número concreto de dígitos, si se nos pide representarlo con más dígitos, lo único que tenemos que hacer es añadir ceros por la izquierda, si el número es positivo, y unos, si es negativo.

Ejercicio 5.23 Expresa -8_{10} en *Complemento a 2* con 16 bits. ■

Ejercicio 5.24 Expresa -58_{10} en *Complemento a 2* con 16 bits. ■

La principal ventaja de este método es que simplifica las operaciones aritméticas. En particular, la resta de números binarios se facilita enormemente utilizando el complemento a dos, puesto que la resta de dos números binarios puede obtenerse sumando al minuendo el complemento a dos del sustraendo. Veamos un ejemplo restando al número

12_{10} el número 5_{10}. Lo primero que hay que hacer es expresar ambos números en binario en *Complemento a 2*: $12_{10} \equiv 0000\ 1100_2$ y $-5_{10} \equiv 1111\ 1011_2$. Si sumamos ambos números, el resultado es el siguiente:

$$\begin{array}{rr} & \overset{+1\ +1\ +1\ +1}{0\ 0\ 0\ 0\ 1\ 1\ 0\ 0} \\ + & 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \\ \hline 1 & 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \end{array}$$

Como podemos comprobar, el resultado de la suma (descartando el último acarreo) es el número $7_{10} \equiv 0000\ 0111_2$ que es el resultado de la operación $12_{10} - 5_{10}$.

Veamos ahora otro ejemplo restando al número 6_{10}, el número 9_{10}. Al igual que en ejemplo anterior, tenemos que expresar ambos números en binario en *Complemento a 2*: $6_{10} \equiv 0000\ 0110_2$ y $-9_{10} \equiv 1111\ 0111_2$. Si sumamos ambos números, el resultado es el siguiente:

$$\begin{array}{rr} & \overset{+1\ +1}{0\ 0\ 0\ 0\ 0\ 1\ 1\ 0} \\ + & 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1 \\ \hline & 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1 \end{array}$$

En este caso, el resultado de la suma (sin la necesidad de descartar el último acarreo) es el número $-3_{10} \equiv 1111\ 1101_2$ que es el resultado de la operación $6_{10} - 9_{10}$.

Para convertir de binario en *Complemento a 2* a decimal, hay que realizar el proceso contrario cuando el número empiece por 1. En este caso, los pasos son:

1. Restar 1.
2. Invertir los ceros por unos y viceversa.
3. Convertir el número binario resultado a decimal. El número resultado multiplicado por -1 es el resultado.

■ **Ejemplo 5.15** Convierte el número $0110\ 1000_2$ expresado en *Complemento a 2* con 8 bits a decimal.

Como el número empieza por 0, sabemos que es positivo. Por lo tanto, la solución se obtiene convirtiendo directamente el número a decimal, obteniendo como resultado $0110\ 1000_2 \equiv 104_{10}$. ■

■ **Ejemplo 5.16** Convierte el número $1110\ 1000_2$ expresado en *Complemento a 2* con 8 bits a decimal.

Como el número empieza por 1, sabemos que es negativo. Por lo tanto, debemos aplicar los tres pasos. Paso 1: restar 1 obteniendo $1110\ 1000_2 - 1 = 1110\ 0111_2$, paso 2: intercambiar ceros por unos y viceversa para obtener $0001\ 1000_2$, paso 3: obtener el número decimal correspondiente $0001\ 1000_2 \equiv 24_{10}$. Por lo tanto, la solución es el negativo del número obtenido, es decir $1110\ 1000_2 \equiv -24_{10}$. ■

Ejercicio 5.25 Convierte el número $0000\ 1001_2$ expresado en *Complemento a 2* con 8 bits a decimal. ■

Ejercicio 5.26 Convierte el número $1111\ 0111_2$ expresado en *Complemento a 2* con 8 bits a decimal. ■

Ejercicio 5.27 Convierte el número $1111\ 0001_2$ expresado en *Complemento a 2* con 8 bits a decimal. ■

Ejercicio 5.28 Convierte el número $1110\ 0011_2$ expresado en *Complemento a 2* con 8 bits a decimal. ■

5.5 Solución a los ejercicios propuestos

Solución al ejercicio 5.1 Expresa el número -120_{10} en binario con codificación *Signo-magnitud* con 8 bits.

Puesto que el número es negativo, el primer bit será 1 y los 7 restantes el resultado de codificar el valor absoluto de -120_{10}, es decir 120_{10}, con 7 dígitos. Puesto que $120_{10} \equiv 111\ 1000_2$, la solución es $-120_{10} \equiv 1111\ 1000_2$. ■

Solución al ejercicio 5.2 Expresa el número 60_{10} en binario con codificación *Signo-magnitud* con 8 bits.

Puesto que el número es positivo, el resultado se obtendrá codificando directamente el número en binario con 8 bits. Por lo tanto, la solución es $60_{10} \equiv 0011\ 1100_2$. ■

Solución al ejercicio 5.3 Expresa el número -10_{10} en binario con codificación *Signo-magnitud* con 8 bits.

Puesto que el número es negativo, el primer bit será 1 y los 7 restantes el resultado de codificar el valor absoluto de -10_{10}, es decir 10_{10}, con 7 dígitos. Puesto que $10_{10} \equiv 000\ 1010_2$, la solución es $-10_{10} \equiv 1000\ 1010_2$. ■

Solución al ejercicio 5.4 Expresa el número -130_{10} en binario con codificación *Signo-magnitud* con 8 bits.

No es posible representar este número en *Signo-magnitud* con 8 bits. ■

Solución al ejercicio 5.5 ¿Cuál será el resultado de sumar los números $1010\ 1010_2$ y $0000\ 1110_2$ teniendo en cuenta el signo (con codificación *Signo-magnitud* de 8 bits)? ¿Y sin tenerlo en cuenta?

Teniendo en cuenta el signo, $1010\ 1010_2 \equiv -42_{10}$ y $0000\ 1110_2 \equiv 14_{10}$. Por lo tanto el resultado de la suma será $-42_{10} + 14_{10} = -28_{10}$, que se expresará en *Signo-magnitud* como $1001\ 1100_2$. Sin tener en cuenta el signo, $1010\ 1010_2 \equiv 170_{10}$ y $0000\ 1110_2 \equiv 14_{10}$. Por lo tanto el resultado de la suma será $170_{10} + 14_{10} = 184_{10}$. El número 184_{10} no se puede expresar en *Signo-magnitud* con 8 bits, por exceder el límite. ■

Solución al ejercicio 5.6 ¿Cuál será el resultado de sumar los números $1010\ 0010_2$ y $0010\ 1110_2$ teniendo en cuenta el signo (con codificación *Signo-magnitud* de 8 bits)? ¿Y sin tenerlo en cuenta?

Teniendo en cuenta el signo, $1010\ 0010_2 \equiv -34_{10}$ y $0010\ 1110_2 \equiv 46_{10}$. Por lo tanto el resultado de la suma será $-34_{10} + 46_{10} = 12_{10}$, que se expresará en *Signo-magnitud* como $0000\ 1100_2$. Sin tener en cuenta el signo, $1010\ 0010_2 \equiv 162_{10}$ y $0010\ 1110_2 \equiv 46_{10}$. Por lo tanto el resultado de la suma será $162_{10} + 46_{10} = 208_{10}$. El número 208_{10} no se puede expresar en *Signo-magnitud* con 8 bits, por exceder el límite. ■

Solución al ejercicio 5.7 Expresa el número 112_{10} en *Exceso Z* con 8 bits.

Z es 127. Por lo tanto, el número que debemos convertir a binario es $112_{10} + 127_{10} = 239_{10}$ La solución es $112_{10} \equiv 1110\ 1111_2$. ■

Solución al ejercicio 5.8 Expresa el número -80_{10} en *Exceso Z* con 8 bits.

Z es 127. Por lo tanto, el número que debemos convertir a binario es $-80_{10} + 127_{10} = 47_{10}$ La solución es $-80_{10} \equiv 0010\ 1111_2$. ■

Solución al ejercicio 5.9 Expresa el número -10_{10} en *Exceso Z* con 8 bits.

Z es 127. Por lo tanto, el número que debemos convertir a binario es $-10_{10} + 127_{10} = 117_{10}$ La solución es $-10_{10} \equiv 0111\ 0101_2$. ■

Solución al ejercicio 5.10 Expresa el número 67_{10} en *Exceso Z* con 8 bits.

Z es 127. Por lo tanto, el número que debemos convertir a binario es $67_{10} + 127_{10} = 194_{10}$ La solución es $67_{10} \equiv 1100\ 0010_2$. ■

Solución al ejercicio 5.11 Expresa el número 112_{10} en *Exceso Z* con 16 bits.

Z es 32 767. Por lo tanto, el número que debemos convertir a binario es $112_{10} + 32\ 767_{10} = 32\ 879_{10}$ La solución es $112_{10} \equiv 1000\ 0000\ 0110\ 1111_2$. ■

Solución al ejercicio 5.12 Expresa el número -80_{10} en *Exceso Z* con 16 bits.

Z es 32 767. Por lo tanto, el número que debemos convertir a binario es $-80_{10} + 32\ 767_{10} = 32\ 687_{10}$ La solución es $-80_{10} \equiv 0111\ 1111\ 1010\ 1111_2$. ■

Solución al ejercicio 5.13 Expresa el número -10_{10} en *Exceso Z* con 32 bits.

Z es 2 147 483 647. Por lo tanto, el número que debemos convertir a binario es $-10_{10} + 2\ 147\ 483\ 647_{10} = 2\ 147\ 483\ 637_{10}$.
La solución es $-10_{10} \equiv 0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0101_2$. ■

Solución al ejercicio 5.14 Expresa el número 67_{10} en *Exceso Z* con 32 bits.

Z es 2 147 483 647. Por lo tanto, el número que debemos convertir a binario es $67_{10} + 2\ 147\ 483\ 647_{10} = 2\ 147\ 483\ 714_{10}$.

La solución es $67_{10} \equiv 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 0010_2$. ■

Solución al ejercicio 5.15 Expresa en decimal el número de 8 bits codificado en *Exceso Z* $1000\ 0100_2$.

Z es 127. $1000\ 0100_2 \equiv 132$. Por lo tanto, la solución es $132_{10} - 127_{10} = 5_{10}$. ■

Solución al ejercicio 5.16 Expresa en decimal el número de 8 bits codificado en *Exceso Z* $0110\ 1011_2$.

Z es 127. $0110\ 1011_2 \equiv 107$. Por lo tanto, la solución es $107_{10} - 127_{10} = -20_{10}$. ■

Solución al ejercicio 5.17 Expresa en decimal el número de 16 bits codificado en *Exceso Z* $0000\ 1010\ 0111\ 0100_2$.

Z es 32 767. $0000\ 1010\ 0111\ 0100_2 \equiv 2676_{10}$. Por lo tanto, la solución es $2676_{10} - 32\ 767_{10} = -30\ 091_{10}$. ■

Solución al ejercicio 5.18 Expresa en decimal el número de 16 bits codificado en *Exceso Z* $0111\ 1111\ 1010\ 0100_2$.

Z es 32 767. $0111\ 1111\ 1010\ 0100_2 \equiv 32\ 767_{10}$. Por lo tanto, la solución es $32\ 676_{10} - 32\ 767_{10} = -91_{10}$. ■

Solución al ejercicio 5.19 Expresa 34_{10} en *Complemento a 2* con 8 bits.

La solución es $34_{10} \equiv 0010\ 0010_2$. ■

Solución al ejercicio 5.20 Expresa -120_{10} en *Complemento a 2* con 8 bits.

Como es un número negativo, hay que aplicar los tres pasos. Paso 1: representar en binario el valor absoluto del número $120_{10} \equiv 0111\ 1000_2$, Paso 2: intercambiando unos por ceros y viceversa: $0111\ 1000_2$. Paso 3: sumar 1, $1000\ 0111_2 + 1 = 1000\ 1000_2$. La solución es $-120_{10} \equiv 1000\ 1000_2$. ■

Solución al ejercicio 5.21 Expresa 120_{10} en *Complemento a 2* con 8 bits.

La solución es $120_{10} \equiv 0111\ 1000_2$. ■

Solución al ejercicio 5.22 Expresa -8_{10} en *Complemento a 2* con 8 bits.

Como es un número negativo, hay que aplicar los tres pasos. Paso 1: representar en binario el valor absoluto del número $8_{10} \equiv 0000\ 1000_2$, Paso 2: intercambiando unos por ceros y viceversa: $1111\ 0111_2$. Paso 3: sumar 1, $1111\ 0111_2 + 1 = 1111\ 1000_2$. La solución es $-8_{10} \equiv 1111\ 1000_2$. ■

Solución al ejercicio 5.23 Expresa -8_{10} en *Complemento a 2* con 16 bits.

Puesto que sabemos (del ejercicio anterior) que -8_{10} en *Complemento a 2* con 8 bits es $1111\ 1000_2$, únicamente tenemos que añadir ocho unos por la izquierda. La

solución es $-8_{10} \equiv 1111\ 1111\ 1111\ 1000_2$. ■

Solución al ejercicio 5.24 Expresa -58_{10} en *Complemento a 2* con 16 bits.

Como es un número negativo, hay que aplicar los tres pasos. Paso 1: representar en binario el valor absoluto del número $58_{10} \equiv 0000\ 0000\ 0011\ 1010_2$, Paso 2: intercambiando unos por ceros y viceversa: $1111\ 1111\ 1100\ 0101_2$. Paso 3: sumar 1, $1111\ 1111\ 1100\ 0101_2 + 1 = 1111\ 1111\ 1100\ 0110_2$. La solución es $-58_{10} \equiv 1111\ 1111\ 1100\ 0110_2$. ■

Solución al ejercicio 5.25 Convierte el número $0000\ 1001_2$ expresado en *Complemento a 2* con 8 bits a decimal.

Como es positivo, se convierte directamente a decimal obteniendo: $0000\ 1001_2 \equiv 9_{10}$. ■

Solución al ejercicio 5.26 Convierte el número $1111\ 0111_2$ expresado en *Complemento a 2* con 8 bits a decimal.

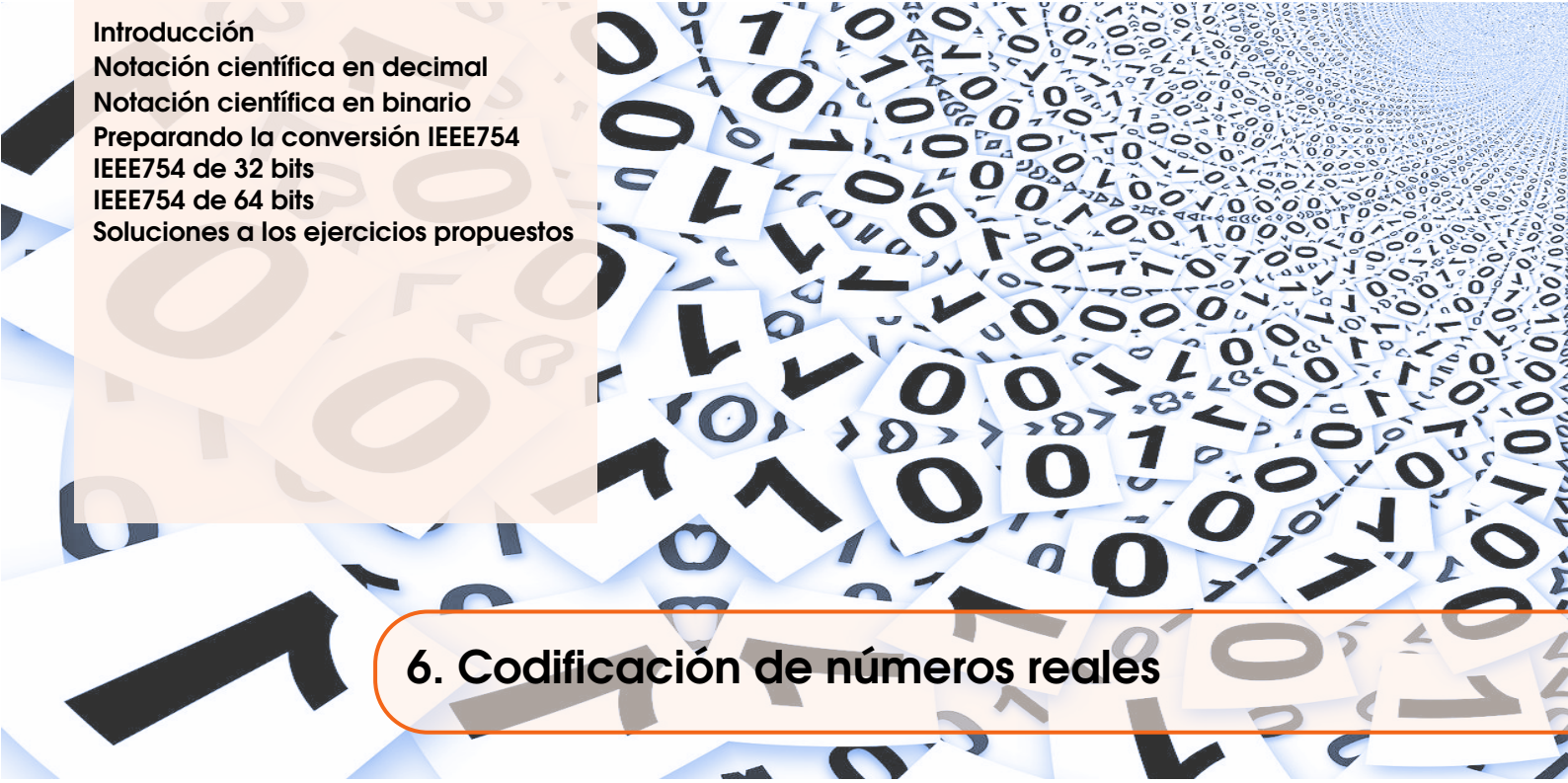
Al ser un número negativo, hay que aplicar los tres pasos, obteniendo como resultado: $1111\ 0111_2 \equiv -9_{10}$. ■

Solución al ejercicio 5.27 Convierte el número $1111\ 0001_2$ expresado en *Complemento a 2* con 8 bits a decimal.

Al ser un número negativo, hay que aplicar los tres pasos, obteniendo como resultado: $1111\ 0001_2 \equiv -15_{10}$. ■

Solución al ejercicio 5.28 Convierte el número $1110\ 0011_2$ expresado en *Complemento a 2* con 8 bits a decimal.

Al ser un número negativo, hay que aplicar los tres pasos, obteniendo como resultado: $1110\ 0011_2 \equiv -29_{10}$. ■



6. Codificación de números reales

6.1 Introducción

En este capítulo, se explicará como se codifican los números reales. En el Capítulo 2 se comentó que, para codificar los números reales, no podemos usar más símbolos que el '0' o el '1'. Por lo tanto, no es posible codificar en un ordenador, por ejemplo, el número $5{,}25_{10}$ como $101{,}01_2$, puesto que no es posible usar el símbolo ',' para separar la parte entera de la parte fraccionaria. La solución a este problema la encontramos en el estándar IEEE754 que define como se deben codificar los números reales en binario. Vamos a ver dos versiones de este estándar, la primera de 32 bits y la segunda de 64 bits.

6.2 Notación científica en decimal

Para poder expresar un número usando el estándar IEEE754, hay que expresar primero el número en *notación científica*, también llamada *notación en coma flotante*. La notación científica es una manera rápida de representar un número utilizando potencias de la base y es muy útil para poder expresar fácilmente números muy grandes o muy pequeños.

En notación científica, los números se escriben como un producto:

$$a * b^e \tag{6.1}$$

donde:

- a es un número real cuya parte entera tiene un único dígito. a recibe el nombre de mantisa o coeficiente.
- b es la base.
- e es un número entero que recibe el nombre de exponente u orden de magnitud.

Ejemplo 6.1 Expresa el número $566{,}03_{10}$ en notación científica. Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por dos veces la base ($b = 10$), obteniendo $566{,}03_{10} = 5{,}6603_{10} * 10^2$. Por lo tanto, el exponente es $e = 2_{10}$, y la mantisa $a = 5{,}6603_{10}$. ■

■ **Ejemplo 6.2** Expresa el número $0{,}0053_{10}$ en notación científica. Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por tres veces la base ($b = 10$), obteniendo $0{,}0053_{10} = 5{,}3_{10} * 10^{-3}$. Por lo tanto, el exponente es $e = -3_{10}$, y la mantisa $a = 5{,}3_{10}$. ■

Ejercicio 6.1 Expresa en notación científica el número $4673{,}57_{10}$. ■

Ejercicio 6.2 Expresa en notación científica el número $0{,}07357_{10}$. ■

Ejercicio 6.3 Expresa en notación científica el número $98{,}157_{10}$. ■

Ejercicio 6.4 Expresa en notación científica el número $0{,}000123157_{10}$. ■

6.3 Notación científica en binario

En binario, también es posible expresar los números reales en notación científica. En este caso, la parte entera de la mantisa es siempre es 1 y la base es obviamente 2.

■ **Ejemplo 6.3** Expresa el número $111{,}001_2$ en notación científica. Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por dos veces la base ($b = 2$), obteniendo $111{,}001_2 \rightarrow 1{,}11001_2 * 2^2$. Por lo tanto, el exponente es $e = 2_{10}$, y la mantisa $a = 1{,}11001_2$. ■

■ **Ejemplo 6.4** Expresa el número $0{,}00101001_2$ en notación científica. Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por tres veces la base ($b = 2$), obteniendo $0{,}00101001_2 \rightarrow 1{,}01001_2 * 2^{-3}$. Por lo tanto, el exponente es $e = -3_{10}$, y la mantisa $a = 1{,}01001_2$. ■

Es importante fijarse que en las expresiones matemáticas $1{,}11001_2 * 2^2$ y $1{,}01001_2 * 2^{-3}$ estamos mezclando un número en binario (la mantisa), con números en decimal (la base y el exponente). Posiblemente, sería más correcto escribir: $111{,}001_2 = 1{,}11001_2 * 100_2$, pero queda más claro, para los intereses que se persiguen en este capítulo, si la magnitud del exponente se expresa en base 10, tal como se ha mostrado en los dos ejemplos anteriores.

Ejercicio 6.5 Expresa en notación científica el número $1101{,}0101_2$. ■

Ejercicio 6.6 Expresa en notación científica el número $0{,}000101_2$. ■

Ejercicio 6.7 Expresa en notación científica el número $111001{,}1101_2$. ■

Ejercicio 6.8 Expresa en notación científica el número $0{,}0000111_2$. ■

6.4 Preparando la conversión IEEE754

Para poder expresar un número real en formato IEEE754, hay que primero convertir el número real a binario. Para ello, convertiremos a binario por un lado, la parte entera y por otro, la parte fraccionaria. Para ello, aplicaremos las técnicas que se han explicado en el Capítulo 2. Una vez realizada la conversión, deberemos expresar el número binario resultante en notación científica.

■ **Ejemplo 6.5** Expresa el número real $4{,}5_{10}$ en binario usando notación científica. El primer paso es convertir a binario la parte entera 4_{10} obteniendo 100_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}5_{10}$, obteniendo $0{,}1_2$. Por lo tanto, $4{,}5_{10} \equiv 100{,}1_2$. Finalmente, expresamos el número binario resultante usando notación científica, obteniendo: $4{,}5_{10} \equiv 1{,}001 * 2^2$. ■

■ **Ejemplo 6.6** Expresa el número real $0{,}125_{10}$ en binario usando notación científica. El primer paso es convertir a binario la parte entera 0_{10} obteniendo 0_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}125_{10}$, obteniendo $0{,}001_2$. Por lo tanto, $0{,}125_{10} \equiv 0{,}001_2$. Finalmente, expresamos el número binario resultante usando notación científica, obteniendo: $0{,}125_{10} \equiv 1{,}0 * 2^{-3}$. ■

Ejercicio 6.9 Expresa el número real $12{,}375_{10}$ en binario usando notación científica. ■

Ejercicio 6.10 Expresa el número real $9{,}875_{10}$ en binario usando notación científica. ■

Ejercicio 6.11 Expresa el número real $112{,}75_{10}$ en binario usando notación científica. ■

Ejercicio 6.12 Expresa el número real $0{,}0078125_{10}$ en binario usando notación científica. ■

Una vez hemos expresado el número real en binario usando notación científica, ya podemos aplicar el estándar IEEE754. Este proceso lo veremos en las dos siguientes secciones.

6.5 IEEE754 de 32 bits

El estándar IEEE754 de 32 bits define que un número real se codifica usando 32 bits, cumpliendo las siguientes normas:

Signo: El signo se codifica con un único bit, situado en la posición 0 (el de más a la izquierda). Este bit será 1 si el número es negativo y 0 si es positivo.

Exponente: El exponente se codifica usando 8 bits, del 1 al 8. El exponente se codifica en Exceso Z, con $Z = 127$ (ver Capítulo 5). De esta forma, es posible representar

números con exponente desde -127 (expresado en binario como $0000\ 0000_2$) hasta 128 (expresado en binario como $1111\ 1111_2$).

Mantisa: La mantisa se codifica usando 23 bits, del 9 al 31. Puesto que todos los números binarios expresados en notación científica empiezan por "1.", solo se almacenan los dígitos situados a la derecha de la coma. Al usar 23 bits, la última potencia de 2 que se tiene en cuenta para expresar el número es 2^{-23}.

■ **Ejemplo 6.7** Expresa el número $5{,}25_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal.

En primer lugar, convertimos a binario la parte entera y la parte fraccionaria del número, obteniendo: $5_{10} \equiv 101_2$ y $0{,}25_{10} \equiv 0{,}01_2$. Por lo tanto, $5{,}25_{10} \equiv 101{,}01_2$. El siguiente paso consiste en expresar el número binario en notación científica: $101{,}01_2 \rightarrow 1{,}0101_2 * 2^2$. Por lo tanto, el signo es positivo, el exponente 2_{10} y la mantisa $1{,}0101_2$. El último paso consiste en codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 127$. Por lo tanto, $2_{10} + 127_{10} = 129_{10} \equiv 1000\ 0001_2$ con 8 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 23 bits, para obtener: $010\ 1000\ 0000\ 0000\ 0000\ 0000_2$.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario $0\ \ 1000\ 0001\ \ 010\ 1000\ 0000\ 0000\ 0000\ 0000_2$. Agrupando de cuatro en cuatro, obtenemos $5{,}25_{10} \equiv 0100\ 0000\ 1010\ 1000\ 0000\ 0000\ 0000\ 0000_2$. En hexadecimal: $5{,}25_{10} \equiv 40A80000_{16}$ ■

Ejercicio 6.13 Expresa el número $12{,}375_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

Ejercicio 6.14 Expresa el número $-9{,}125_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

Ejercicio 6.15 Expresa el número $0{,}03515625_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

6.6 IEEE754 de 64 bits

El estándar IEEE754 de 64 bits define que un número real se codifica usando 64 bits, cumpliendo las siguientes normas:

- **Signo**: El signo se codifica con un único bit, situado en la posición 0 (el de más a la izquierda). Este bit será 1 si el número es negativo y 0 si es positivo.
- **Exponente**: El exponente se codifica usando 11 bits, del 1 al 11. El exponente se codifica en Exceso Z, con $Z = 1023$ (ver Capítulo 5). De esta forma, es posible representar números con exponente desde -1022 (expresado en binario como $000\ 0000\ 0000_2$) hasta 1023 (expresado en binario como $111\ 1111\ 1111_2$).
- **Mantisa**: La mantisa se codifica usando 52 bits, del 12 al 63. Puesto que todos

los números binarios expresados en notación científica empiezan por "1.", solo se almacenan los dígitos situados a la derecha de la coma, es decir la parte fraccionaria de la mantisa. Al usar 52 bits, la última potencia de 2 que se tiene en cuenta para expresar el número es 2^{-52}.

■ **Ejemplo 6.8** Expresa el número $5{,}25_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal.

Como hemos visto en un ejemplo anterior $5{,}25_{10} \equiv 101{,}01_2 \rightarrow 1{,}0101_2 * 2^2$. Por lo tanto, el signo es positivo, el exponente 2_{10} y la mantisa $1{,}0101_2$. El último paso es codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 1023$. Por lo tanto, $2_{10} + 1023_{10} = 1025_{10} \equiv 100\ 0000\ 0001_2$ con 11 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 52 bits, para obtener: $0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario:
$0\ \ 100\ 0000\ 0001\ \ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
Agrupando de cuatro en cuatro, obtenemos:
$5{,}25_{10} \equiv 0100\ 0000\ 0001\ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
En hexadecimal: $5{,}25_{10} \equiv 4015000000000000_{16}$ ■

Ejercicio 6.16 Expresa el número $12{,}375_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

Ejercicio 6.17 Expresa el número $-9{,}125_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

Ejercicio 6.18 Expresa el número $0{,}03515625_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal. ■

6.7 Soluciones a los ejercicios propuestos

Solución al ejercicio 6.1 Expresa en notación científica el número $4673{,}57_{10}$.

Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por tres veces la base ($b = 10$), obteniendo $4673{,}57_{10} = 4{,}67357_{10} * 10^3$. Por lo tanto, el exponente es $e = 3_{10}$, y la mantisa $a = 4{,}67357_{10}$. ■

Solución al ejercicio 6.2 Expresa en notación científica el número $0{,}07357_{10}$.

Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por dos veces la base ($b = 10$), obteniendo $0{,}07357_{10} = 7{,}357_{10} * 10^{-2}$. Por lo tanto, el exponente es $e = -2_{10}$, y la mantisa $a = 7{,}357_{10}$. ■

Solución al ejercicio 6.3 Expresa en notación científica el número $98{,}157_{10}$.

Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por una vez la base ($b = 10$), obteniendo $98{,}157_{10} = 9{,}8157_{10} * 10^1$. Por lo tanto, el exponente es $e = 1_{10}$, y la mantisa $a = 9{,}8157_{10}$. ■

Solución al ejercicio 6.4 Expresa en notación científica el número $0{,}000123157_{10}$.

Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por cuatro veces la base ($b = 10$), obteniendo $0{,}000123157_{10} = 1{,}23157_{10} * 10^-4$. Por lo tanto, el exponente es $e = -4_{10}$, y la mantisa $a = 1{,}23157_{10}$. ■

Solución al ejercicio 6.5 Expresa en notación científica el número $1101{,}0101_2$.

Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por tres veces la base ($b = 2$), obteniendo $1101{,}0101_2 \rightarrow 1{,}1010101_2 * 2^3$. Por lo tanto, el exponente es $e = 3_{10}$, y la mantisa $a = 1{,}1010101_2$. ■

Solución al ejercicio 6.6 Expresa en notación científica el número $0{,}000101_2$.

Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por cuatro veces la base ($b = 2$), obteniendo $0{,}000101_2 \rightarrow 1{,}01_2 * 2^{-4}$. Por lo tanto, el exponente es $e = -4_{10}$, y la mantisa $a = 1{,}01_2$ ■

Solución al ejercicio 6.7 Expresa en notación científica el número $111001{,}1101_2$.

Para conseguir que la parte entera tenga solo un dígito será necesario dividir el número por cinco veces la base ($b = 2$), obteniendo $111001{,}1101_2 \rightarrow 1{,}110011101_2 * 2^5$. Por lo tanto, el exponente es $e = 5_{10}$, y la mantisa $a = 1{,}110011101_2$. ■

Solución al ejercicio 6.8 Expresa en notación científica el número $0{,}0000111_2$.

Para conseguir que la parte entera tenga solo un dígito será necesario multiplicar el número por cinco veces la base ($b = 2$), obteniendo $0{,}0000111_2 \rightarrow 1{,}11_2 * 2^{-5}$. Por lo tanto, el exponente es $e = -5_{10}$, y la mantisa $a = 1{,}11_2$. ■

Solución al ejercicio 6.9 Expresa el número real $12{,}375_{10}$ en binario usando notación científica.

El primer paso es convertir a binario la parte entera 12_{10} obteniendo 1100_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}375_{10}$, obteniendo $0{,}011_2$. Por lo tanto, $12{,}375_{10} \equiv 1100{,}011_2$. Finalmente, expresamos el número binario resultado usando notación científica obteniendo: $12{,}375_{10} \equiv 1{,}100011_2 * 2^3$. ■

Solución al ejercicio 6.10 Expresa el número real $9{,}875_{10}$ en binario usando notación científica.

El primer paso es convertir a binario la parte entera 9_{10} obteniendo 1001_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}875_{10}$, obteniendo $0{,}111_2$. Por lo tanto, $9{,}875_{10} \equiv 1001{,}111_2$. Finalmente, expresamos el número binario resultado

usando notación científica obteniendo: $9{,}875_{10} \equiv 1{,}001111_2 * 2^3$. ■

Solución al ejercicio 6.11 Expresa el número real $112{,}75_{10}$ en binario usando notación científica.

El primer paso es convertir a binario la parte entera 112_{10} obteniendo 1110000_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}75_{10}$, obteniendo $0{,}11_2$. Por lo tanto, $112{,}75_{10} \equiv 1110000{,}11_2$. Finalmente, expresamos el número binario resultado usando notación científica obteniendo: $112{,}75_{10} \equiv 1{,}11000011_2 * 2^6$. ■

Solución al ejercicio 6.12 Expresa el número real $0{,}0078125_{10}$ en binario usando notación científica.

El primer paso es convertir a binario la parte entera 0_{10} obteniendo 0_2. El segundo paso es convertir a binario la parte fraccionaria $0{,}0078125_{10}$, obteniendo $0{,}0000001_2$. Por lo tanto, $0{,}0078125_{10} \equiv 0{,}0000001_2$. Finalmente, expresamos el número binario resultado usando notación científica obteniendo: $0{,}0078125_{10} \equiv 1{,}0_2 * 2^{-7}$. ■

Solución al ejercicio 6.13 Expresa el número $12{,}375_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal.

En primer lugar, convertimos a binario la parte entera y la parte fraccionaria del número, obteniendo: $12_{10} \equiv 1100_2$ y $0{,}375_{10} \equiv 0{,}011_2$. Por lo tanto, $12{,}375_{10} \equiv 1100{,}011_2$. El siguiente paso consiste en expresar el número binario en notación científica: $1100{,}011 \rightarrow 1{,}100011 * 2^3$. Por lo tanto, el signo es positivo, el exponente 3_{10} y la mantisa $1{,}100011_2$. El último paso consiste en codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 127$. Por lo tanto, $3_{10} + 127_{10} = 130_{10} \equiv 1000\ 0010_2$ con 8 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 23 bits, para obtener: $100\ 0110\ 0000\ 0000\ 0000\ 0000_2$.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario $0\ \ 1000\ 0010\ \ 100\ 0110\ 0000\ 0000\ 0000\ 0000_2$. Agrupando de cuatro en cuatro, obtenemos $12{,}375_{10} \equiv 0100\ 0001\ 0100\ 0110\ 0000\ 0000\ 0000\ 0000_2$.
En hexadecimal: $12{,}375_{10} \equiv 41460000_{16}$. ■

Solución al ejercicio 6.14 Expresa el número $-9{,}125_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal.

En primer lugar, convertimos a binario la parte entera y la parte fraccionaria del número, obteniendo: $9_{10} \equiv 1001_2$ y $0{,}125_{10} \equiv 0{,}001_2$. Por lo tanto, $-9{,}125_{10} \equiv 1001{,}001_2$. El siguiente paso consiste en expresar el número binario en notación científica: $1001{,}001 \rightarrow 1{,}001001 * 2^3$. Por lo tanto, el signo es negativo, el exponente 3_{10} y la mantisa $1{,}001001_2$. El último paso consiste en codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 1, puesto que es negativo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 127$. Por lo tanto, $3_{10} + 127_{10} = 130_{10} \equiv 1000\ 0010_2$ con 8 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 23 bits, para obtener: $001\ 0010\ 0000\ 0000\ 0000\ 0000_2$.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario $1\ \ 1000\ 0010\ \ 001\ 0010\ 0000\ 0000\ 0000\ 0000_2$. Agrupando de cuatro en cuatro, obtenemos $1100\ 0001\ 0001\ 0010\ 0000\ 0000\ 0000\ 0000$. $-9{,}125_{10} \equiv 1100\ 0001\ 0001\ 0010\ 0000\ 0000\ 0000\ 0000_2$.
En hexadecimal: $-9{,}125_{10} \equiv C1120000_{16}$.

■

Solución al ejercicio 6.15 Expresa el número $0{,}03515625_{10}$ en binario usando el estándar IEEE754 de 32 bits. Expresa el resultado final tanto en binario como en hexadecimal.

En primer lugar, convertimos a binario la parte entera y la parte fraccionaria del número, obteniendo: $0_{10} \equiv 0_2$ y $0{,}03515625_{10} \equiv 0{,}00001001_2$. Por lo tanto, $0{,}03515625_{10} \equiv 0{,}00001001_2$. El siguiente paso consiste en expresar el número binario en notación científica: $0{,}00001001 \rightarrow 1{,}001 * 2^{-5}$. Por lo tanto, el signo es positivo, el exponente -5_{10} y la mantisa $1{,}001_2$. El último paso consiste en codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 127$. Por lo tanto, $-5_{10} + 127_{10} = 122_{10} \equiv 0111\ 1010_2$ con 8 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 23 bits, para obtener: $001\ 0000\ 0000\ 0000\ 0000\ 0000_2$.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario $0\ \ 0111\ 1010\ \ 001\ 0000\ 0000\ 0000\ 0000\ 0000_2$. Agrupando de cuatro en cuatro, obtenemos $0011\ 1101\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000$. $0{,}03515625_{10} \equiv 0011\ 1101\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
En hexadecimal: $0{,}03515625_{10} \equiv 3D100000_{16}$. ■

Solución al ejercicio 6.16 Expresa el número $12{,}375_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal.

Como hemos visto en un ejemplo anterior $12{,}375_{10} \rightarrow 1{,}1100011 * 2^3$. Por lo tanto, el signo es positivo, el exponente 3_{10} y la mantisa $1{,}1100011_2$. El último paso es codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 1023$. Por lo tanto, $3_{10} + 1023_{10} = 1026_{10} \equiv 100\ 0000\ 0010_2$ con 11 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 52 bits, para obtener: $1000\ 1100\ [0000...]_2$.

Atención: [0000...] significa que el resto de símbolos son ceros hasta completar los 52 bits.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario:
0 100 0000 0010 1000 1100 $[0000...]_2$. Agrupando de cuatro en cuatro, obtenemos: $12{,}375_{10} \equiv 0100000000101000 1100\ [0000...]_2$.
En hexadecimal: $12{,}375_{10} \equiv 4028C00000000000_{16}$ ■

Solución al ejercicio 6.17 Expresa el número $-9{,}125_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal.

Como hemos visto en un ejemplo anterior $9{,}125_{10} \rightarrow 1{,}001001 * 2^3$. Por lo tanto, el signo es negativo, el exponente 3_{10} y la mantisa $1{,}001001_2$. El último paso es codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 1, puesto que es negativo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 1023$. Por lo tanto, $3_{10} + 1023_{10} = 1026_{10} \equiv 100\ 0000\ 0010_2$ con 11 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 52 bits, para obtener: 0010 0100 $[0000...]_2$.

Atención: [0000...] significa que el resto de símbolos son ceros hasta completar los 52 bits.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario:
1 100 0000 0010 0010 0100 $[0000...]_2$. Agrupando de cuatro en cuatro, obtenemos: $-9{,}125_{10} \equiv 0100000001000100100\ [0000...]_2$.
En hexadecimal: $-9{,}125_{10} \equiv C022400000000000_{16}$ ■

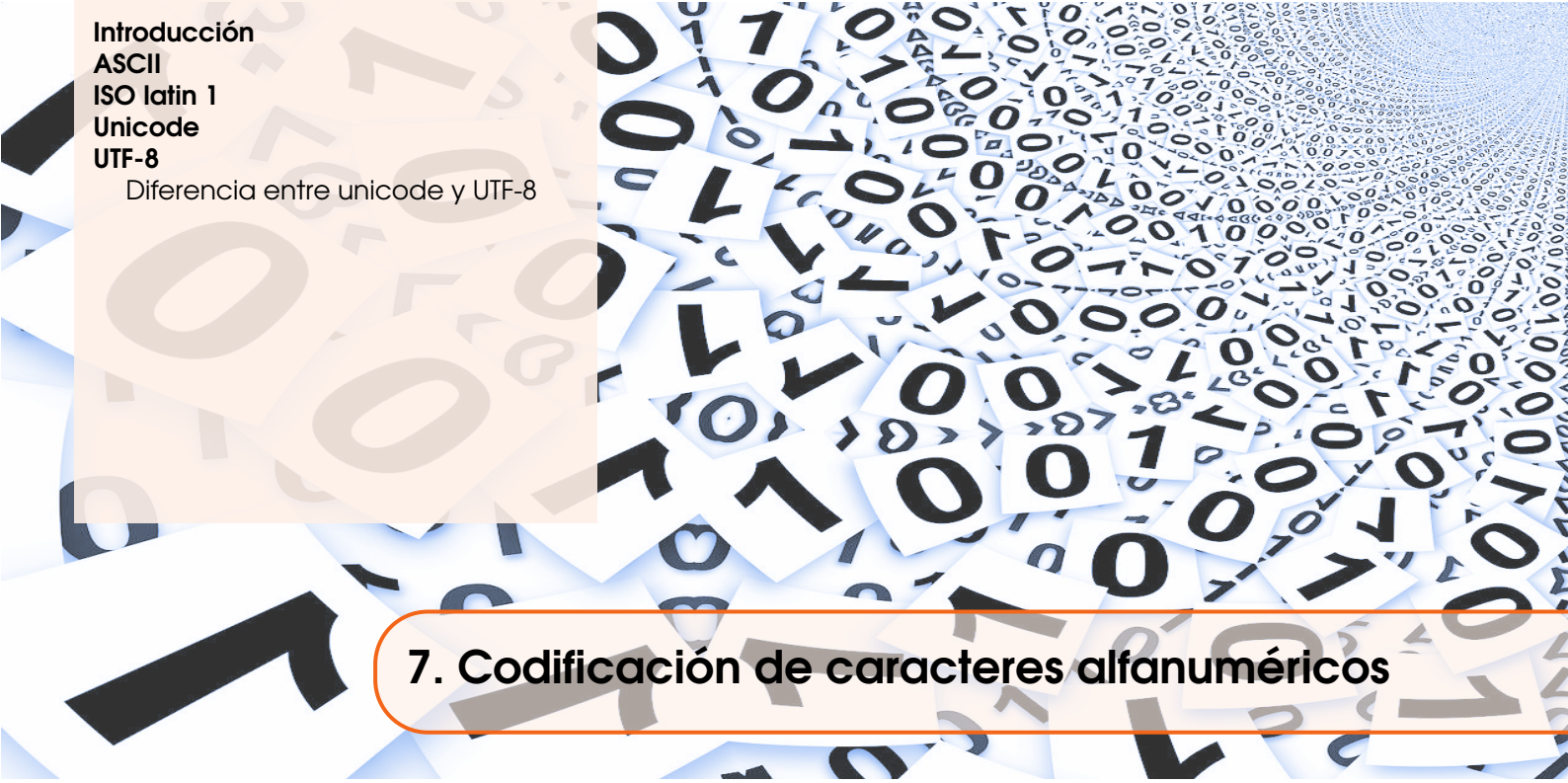
Solución al ejercicio 6.18 Expresa el número $0{,}03515625_{10}$ en binario usando el estándar IEEE754 de 64 bits. Expresa el resultado final tanto en binario como en hexadecimal.

Como hemos visto en un ejemplo anterior $0{,}03515625_{10} \rightarrow 1{,}001 * 2^{-5}$. Por lo tanto, el signo es positivo, el exponente -5_{10} y la mantisa $1{,}001_2$. El último paso es codificar los tres elementos: signo, exponente y mantisa, tal como se ha explicado anteriormente:

- **Signo**: El signo será 0, puesto que es positivo.
- **Exponente**: El exponente se codifica en Exceso Z, con $Z = 1023$. Por lo tanto, $-5_{10} + 1023_{10} = 1018_{10} \equiv 011\ 1111\ 1010_2$ con 11 bits.
- **Mantisa**: Solo se codifica la parte fraccionaria de la mantisa, usando 52 bits, para obtener: 0010 $[0000...]_2$.

Atención: [0000...] significa que el resto de símbolos son ceros hasta completar los 52 bits.

Finalmente, el número resultado es la unión de las tres partes, es decir el número binario:



7. Codificación de caracteres alfanuméricos

7.1 Introducción

En los seis capítulos anteriores, se ha explicado en detalle como codificar en binario números enteros y reales, tanto positivos como negativos. Los lenguajes de programación también incluyen texto para mostrar mensajes por la pantalla para facilitar o requerir información al usuario del programa. Un texto se compone de una secuencia de caracteres. Por ejemplo, el texto "Buenos días" se compone de 11 caracteres: 'B', 'u', 'e', 'n', 'o', 's', ' ', 'd', 'í', 'a'y 's'. Como se puede comprobar, el término carácter se refiere tanto a las letras que usamos para componer los mensajes (incluyendo mayúsculas, minúsculas, vocales acentuadas, etc), como caracteres especiales como el espacio. Existen multitud de caracteres para los que es necesario tener una codificación, como por ejemplo los interrogantes, exclamaciones, los dígitos numéricos, letras griegas (útiles para expresar fórmulas matemáticas), etc. Si tenemos en cuenta todos los posibles símbolos que se pueden usar en cada idioma de entre los muchos existentes en la Tierra, podemos comprender la dificultad de encontrar una forma de poder codificar todos los símbolos existentes.

7.2 ASCII

El primer intento que se ideó para codificar caracteres es el código ASCII [Wik15a] (acrónimo inglés de American Standard Code for Information Interchange). El código ASCII es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. Cada carácter se codifica usando un byte, aunque en realidad solo se usan 7 bits, siendo el bit más significativo siempre cero. Al tener 7 bits, se puede representar $2^7 = 128$ caracteres diferentes.

De los 128 posibles, los 32 primeros se usan para codificar caracteres no imprimibles, de los cuales la mayoría son caracteres de control que tienen efecto sobre cómo se procesa el texto. Los 95 siguientes son caracteres imprimibles (empezando por el carácter espacio). El último código se usa para codificar el carácter DEL (tecla *Delete* del teclado). Los caracteres de control no fueron pensados originalmente para representar

Binario	Dec	Hex	Representación
0010 0000	32	20	espacio ( )
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binario	Dec	Hex	Representación
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binario	Dec	Hex	Representación
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

Figura 7.1: Caracteres imprimibles de la Tabla ASCII ([Wik15a])

información imprimible, sino para controlar dispositivos (como impresoras) que usaban ASCII. Por ejemplo, el carácter codificado como 10_{10} representa la función “nueva línea”, que hace que una impresora avance el papel, y el carácter codificado como 27_{10} representa la tecla “escape” que a menudo se encuentra en la esquina superior izquierda de los teclados comunes. El paso del tiempo ha dejado obsoleto la mayoría de los caracteres de control.

El carácter “espacio” (codificado como 32_{10}), designa al espacio entre palabras, y se produce normalmente por la barra espaciadora de un teclado. Los códigos del 33_{10} al 126_{10} se conocen como caracteres imprimibles, y representan letras, dígitos, signos de puntuación y varios símbolos. La Figura 7.1 muestra los caracteres imprimibles del código ASCII.

Tal como muestra la Figura 7.1, las mayúsculas y las minúsculas se diferencian en únicamente 1 bit, en concreto el tercero (por la izquierda). Por ejemplo, el carácter ‘A’ se codifica en binario como $0100\ 0001_2$ (65_{10} en decimal), y su equivalente en minúsculas ‘a’ se codifica como $0110\ 0001_2$ (97_{10} en decimal). Como se puede comprobar, dado el código de una letra mayúscula, es muy sencillo obtener el de su equivalente en minúsculas, añadiendo 32 en decimal, o reemplazar el “0” por el “1” en el tercer bit (por la izquierda) del código en binario.

Los dígitos se codifican desde el '0' como $0011\ 0000_2$ (48 en decimal) hasta el '9' como $0011\ 1001_2$ (57 en decimal). Como se puede comprobar, los cuatro últimos de la codificación ASCII coinciden con la codificación de los números enteros positivos con $N = 4$. Es importante recalcar, al igual que ocurría con la codificación de los números con signo, que hay que especificar como está codificado un conjunto de bits para saber que significa dicho código. Por ejemplo, dado el código en binario $0101\ 0100_2$, si estamos codificando un número sin signo, entonces $0101\ 0100_2 \equiv 84_{10}$. Pero si estamos codificando un carácter ASCII entonces $0101\ 0100_2$ se refiere al carácter 'T'. También tenemos que darnos cuenta que no es lo mismo el número 7_{10} que el carácter '7'. En el primero de caso, es un número que podemos usar para realizar operaciones matemáticas. En el segundo caso, es un carácter que podemos usar para mostrar texto.

Aunque el código ASCII supuso un gran avance en su momento, tiene una importante limitación. Mirando la tabla de los caracteres imprimibles (Figura 7.1) podemos observar que faltan muchos caracteres necesarios para escribir texto en idiomas diferentes al inglés, como por ejemplo las vocales acentuadas, la letra 'Ñ', el interrogante abierto, etc. Si pensamos en todos los idiomas de la tierra (como por ejemplo, el chino o el japonés, entre muchos otros que no usan el alfabeto latino) podemos comprobar como los 7 bits del ASCII no son suficientes para codificar todos los posibles caracteres de todos los idiomas.

7.3 ISO latin 1

ISO latin 1 [Wik14e], también conocida como ISO 8859-1, es una norma de la ISO que define la codificación del alfabeto latino, incluyendo los diacríticos (como letras acentuadas, ñ, ç), y letras especiales (propias de idiomas de países nórdicos), necesarios para la escritura de la mayoría de las lenguas originarias de Europa occidental, como por ejemplo los idiomas alemán, castellano, catalán, euskera, etc. En [Wik14e] se puede encontrar la lista de los símbolos incluidos en ISO latin 1 no incluidos en ASCII. Por ejemplo, el símbolo 'Á' se codifica como $1100\ 0001_2$ (193_{10} en decimal) y el símbolo 'Ñ' como 1101 0001 (209_{10}). Al igual que ocurría en el código ASCII, las letras mayúsculas y minúsculas se diferencias en 32_{10}, o lo que es lo mismo, el tercer bit (por la izquierda) es '0' en las mayúsculas y '1' en las minúsculas. Por ejemplo, el símbolo 'ñ' se codifica como 1111 0001 (241_{10}).

En este caso se usan los 8 bits, pero manteniendo intacta la codificación de los 128 códigos ASCII. Es decir, ambas codificaciones (ISO latin 1 y ASCII) coinciden en los 7 bits más a la derecha. Los códigos que empiezan por 1 son los caracteres especiales no incluidos en el código ASCII original. Por lo tanto, con ISO latin 1 se pueden codificar 256 caracteres diferentes.

Existe una modificación conocida como ISO 8859-15 que incorpora el símbolo del Euro y algunos caracteres necesarios para dar soporte completo al francés, finés y estonio.

La inclusión de 128 caracteres nuevos soluciona el problema para algunos idiomas, como el castellano, pero mantiene el problema de muchos otros idiomas que no usan el alfabeto latino, como por ejemplo, el chino, el árabe, el japonés o el etíope.

7.4 Unicode

Unicode [Wik15c] es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas, además de textos clásicos de lenguas muertas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad. En su creación se perseguían tres objetivos:

- **Universalidad**: Un repertorio suficientemente amplio que albergue a todos los caracteres probables en el intercambio de texto multilingüe.
- **Eficiencia**: Las secuencias generadas deben ser fáciles de tratar.
- **No ambigüedad**: Un código dado siempre representa el mismo carácter.

La característica más importante es que Unicode especifica un nombre e identificador numérico único para cada carácter o símbolo existente. De esta forma todos los caracteres posibles que se pueden usar, sea cual sea el idioma, tienen un código único. La descripción completa del estándar y las tablas de caracteres están disponibles en la página web oficial de Unicode (`http://www.unicode.org/charts/`).

Unicode incluye todos los caracteres de uso común en la actualidad. La versión 5.1 contenía 100 713 caracteres provenientes de alfabetos, sistemas ideográficos y colecciones de símbolos (matemáticos, técnicos, musicales, iconos, etc.). La cifra crece con cada versión. Unicode incluye sistemas de escritura modernos como: árabe, braille, copto, cirílico, griego, sinogramas (hanja coreano, hanzi chino y kanji japonés), silabarios japoneses (hiragana y katakana), hebreo y latino; escrituras históricas extintas, para propósitos académicos, como por ejemplo: cuneiforme, griego antiguo, micénico, fenicio y rúnico. Entre los caracteres no alfabéticos incluidos en Unicode se encuentran símbolos musicales y matemáticos, fichas de juegos como el dominó, flechas, iconos, etc.

7.5 UTF-8

UTF-8 (8-bit Unicode Transformation Format) [Wik15d] es un formato de codificación de caracteres Unicode. UTF-8 divide los caracteres Unicode en varios grupos, en función del número de bytes necesarios para codificarlos. El número de bytes depende exclusivamente del código de carácter asignado por Unicode y del número de bytes necesario para representarlo.

Las principales ventajas de UTF-8 es que permite codificar cualquier carácter Unicode. Además es compatible con la codificación ASCII original, puesto que la codificación de los caracteres incluidos en ASCII es idéntica en UTF-8. Sin embargo, UTF-8 tiene la desventaja de usar símbolos de longitud variable, eso significa que diferentes caracteres pueden codificarse con distinto número de bytes.

7.5.1 Diferencia entre unicode y UTF-8

Los términos Unicode y UTF-8 se suelen confundir. Unicode es un estándar que asigna a cada posible carácter un identificador numérico, pero no es una codificación para ser usada por el ordenador. UTF-8 es una forma de codificar los caracteres Unicode.

Imaginemos la siguiente secuencia de 5 bytes: $0110\ 1000_2$, $0110\ 0101_2$, $0110\ 1100_2$, $0110\ 1100_2$, $0110\ 1111_2$. Sabiendo que los 5 bytes anteriores son caracteres Unicode codificados en UTF-8 podemos averiguar el texto escrito en esos 5 bytes. El primer paso es convertir de binario a decimal, obteniendo: 104_{10}, 101_{10}, 108_{10}, 108_{10}, 111_{10}. El

siguiente paso es consultar las tablas Unicode para averiguar a que carácter corresponde cada identificador. En este caso, son letras del alfabeto latino, por lo que ocupan un único byte (codificadas en UTF-8). Mirando las tabla Unicode podemos averiguar que el texto es "hello". Para ello, podemos usar la web: `http://unicode-table.com/es/`, donde podemos comprobar que el identificador para el carácter 'h' es 104_{10} (68_{16}), para el carácter 'e' es 101_{10} (65_{16}) y así sucesivamente para el resto de caracteres.