Representación de la información

A. Josep Velasco González

Con la colaboración de:
Ramon Costa Castelló
Montse Peiron Guàrdia

PID_00163598

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción 5

Objetivos 7

1. Los números y los sistemas de representación 9
1.1. Sistemas de representación 9
1.2. Sistemas de numeración posicionales 10
1.3. Cambios de base 13
1.3.1. Método basado en el TFN 13
1.3.2. Método basado en el teorema de la división entera 14
1.3.3. Cambio de base entre b y b^n 17
1.4. Empaquetamiento de la información 18
1.5. Números con signo 21
1.6. Suma en los sistemas posicionales 22
1.7. Resta en los sistemas posicionales 22
1.8. Multiplicación y división por potencias de la base de numeración 23

2. Representación de los números en un computador 26
2.1. Condicionantes físicos 26
2.1.1. Rango de representación 27
2.1.2. Precisión 28
2.1.3. Error de representación 28
2.1.4. Aproximaciones: truncamiento y redondeo 28
2.1.5. Desbordamiento 30
2.2. Números naturales 31
2.3. Números enteros 33
2.3.1. Representación de enteros en signo y magnitud en base 2 33
2.3.2. Suma y resta en signo y magnitud 35
2.3.3. Representación en complemento a 2 36
2.3.4. Cambio de signo en complemento 2 38
2.3.5. Magnitud de los números en complemento a 2 40
2.3.6. Suma en complemento a 2 40
2.3.7. Resta en complemento a 2 42
2.3.8. Multiplicación por 2^k de números en complemento a 2 43
2.4. Números fraccionarios 44

3. Otros tipos de representaciones 53
3.1. Representación de información alfanumérica 53
3.2. Codificación de señales analógicas 55
3.3. Otras representaciones numéricas 58

3.3.1. Representación en exceso a M 58
3.3.2. Representación en coma flotante 60
3.3.3. Representación BCD 64

Resumen 66

Ejercicios de autoevaluación 67

Solucionario 68

Glosario 94

Bibliografía 95

Introducción

Inicialmente, los computadores fueron desarrollados como una herramienta para agilizar la realización repetitiva de operaciones aritméticas y lógicas básicas, que con el tiempo fueron ganando complejidad, tanto por el número de operaciones como por la complejidad propia de los cálculos. Hoy en día, sin haber perdido la utilidad original, los computadores se han ido diversificando, adaptándose a múltiples aplicaciones hasta convertirse en un elemento imprescindible en todos los campos de la ciencia, de la comunicación y del ocio.

A pesar de los grandes cambios que han ido sufriendo las máquinas, el procesamiento de los datos dentro de un computador continúa basado en la realización de operaciones aritméticas y lógicas sencillas sobre datos que se encuentran en la memoria principal. Allí pueden haber llegado de procedencias diversas, pero en todos los casos, la información ha sufrido una transformación: se ha codificado de manera adecuada para poder ser tratada por un procesador digital.

Las características de la tecnología con la que se construyen los computadores obligan a trabajar con sólo dos símbolos diferentes: el 0 y el 1. Toda la información que tenga que procesar un computador se tendrá que codificar usando únicamente estos dos símbolos.

Dentro de un computador, cualquier información (valor numérico, texto, audio, vídeo) está representada como una cadena de 0's y 1's. Ahora bien, una cadena de ceros y unos sólo tiene sentido si conocemos el formato de representación, es decir, la manera como está codificada la información, lo cual incluye saber: el tipo de dato (es un número, un texto, una señal de audio digitalizada, etc.) y el sistema utilizado para representar este tipo de datos (es decir, el sistema de numeración, si es un número; la tabla de codificación de los caracteres, si se trata de un texto; el algoritmo de codificación y/o compresión por información multimedia; etc.)

Nota

Se usan los símbolos 0 y 1, porque son los dígitos binarios, el sistema que emplean los computadores. Además, también se usan para designar los términos *verdad* y *falso* en las operaciones lógicas.

¿Qué codifica la cadena 10100100? Pues depende. ¿De qué tipo de dato se trata? Si es un texto, y se ha usado el código ASCII ISO-8859-15 se trata del carácter "**€**"; si es un número natural, se trata del valor decimal 164; si es un entero codificado en signo y magnitud, es el valor decimal –36; si es un entero codificado en el sistema de complemento a 2, es el valor decimal –92; etc. En todos los casos se trata de la misma cadena, pero en cada caso se está considerando que esta cadena es el resultado de codificar la información de una manera diferente.

Nota

Signo y magnitud y complemento a 2 son sistemas de representación de números con signo que se describen en la segunda sección de este módulo.

La información que procesa un computador digital está codificada en cadenas de ceros y unos, y esto quiere decir que las operaciones que tienen lugar en el

computador son operaciones sobre cadenas de ceros y unos. De hecho, todo el procesamiento que se hace en los computadores se reduce a operaciones aritméticas y lógicas sencillas sobre las cadenas que codifican la información.

Estos son, pues, los puntos de partida:

- Dentro de un computador toda la información se codifica como cadenas de ceros y unos.

- Una cadena de ceros y unos no tiene sentido por ella misma. Hay que conocer la manera como se codifica la información, esto es el formato en que están codificados los datos.

- El procesamiento que lleva a cabo un computador sobre las cadenas de ceros y unos consiste en operaciones aritméticas y lógicas sencillas.

Mayoritariamente, la información dentro de los computadores es tratada como números y operada como tal, por lo tanto, conocer la manera en que se codifican los números es básico para entender el funcionamiento de los computadores.

En este módulo se explican los sistemas básicos de codificación de la información, prestando especial atención a la representación de la información numérica, a la que se dedica la mayor parte del módulo. El módulo se estructura de la forma siguiente. En primer lugar, se hace un análisis del sistema de numeración con el que estamos habituados a trabajar. A continuación, se explican los sistemas de codificación de números más usuales en los computadores, y finalmente, se dan las pautas para la codificación de datos no numéricos.

Objetivos

Se enumeran a continuación los principales objetivos que hay que lograr con el estudio de este módulo:

1. Comprender cómo se puede representar cualquier tipo de información dentro de los computadores y conocer los principios básicos de la codificación.

2. Conocer en profundidad los sistemas ponderados no redundantes de base fija 2, 10 y 16, además de saber representar un mismo valor numérico en bases diferentes.

3. Comprender y saber utilizar los formatos con que se codifica la información numérica en un computador: el sistema ponderado en binario para los números naturales; signo y magnitud y complemento a 2 para los números enteros, y la representación de números fraccionarios en coma fija.

4. Conocer las operaciones aritméticas básicas que lleva a cabo un computador y saber efectuarlas a mano. Estas operaciones son la suma, la resta y la multiplicación y división por potencias de la base de números naturales, enteros y fraccionarios.

5. Comprender los conceptos de rango y precisión de un formato de codificación de la información numérica en un computador, así como los conceptos de desbordamiento y de error de representación.

6. Entender la manera de empaquetar cadenas de unos y ceros a partir de la base 16.

7. Conocer la forma de representar caracteres en formato ASCII.

1. Los números y los sistemas de representación

El objetivo de esta sección es analizar el sistema de numeración que utilizamos, identificando los parámetros que lo definen. Para hacerlo, se introducen los conceptos de raíz o base y de peso asociado a la posición de un dígito. Seguidamente, se explican las técnicas para encontrar la representación de un número en un sistema posicional de raíz fija cuando se cambia la raíz. Finalmente, se hace un análisis de la operación más común, la suma, y de su homóloga, la resta, así como de la multiplicación y de la división de números por potencias de la base de numeración.

Terminología

A lo largo del texto utilizaremos indistintamente los términos *representar* y *codificar* para referirnos a la manera como se escribe un dato según una sintaxis y un conjunto de símbolos determinado.

1.1. Sistemas de representación

La idea de valor numérico es un concepto abstracto que determina una cantidad. Los valores numéricos están sujetos a un orden de precedencia que se utiliza para relacionarlos y llegar a conclusiones. Para trabajar de manera ágil con este tipo de información, tenemos que poder representar los valores numéricos de manera eficiente, por lo cual se han desarrollado los llamados **sistemas de numeración**.

Un **sistema de numeración** es una metodología que permite representar un conjunto de valores numéricos.

El abanico de sistemas de numeración es bastante amplio. Entre otros, podemos encontrar los sistemas de raíz o base, los sistemas de dígitos firmados, los sistemas de residuos y los sistemas racionales. De estos, los sistemas basados en raíz (o base) son los que más se utilizan por las ventajas que aportan en la manipulación aritmética de los valores numéricos, y en ellos centraremos nuestra atención.

Terminología

Se puede utilizar la designación de base o raíz de forma indistinta, a pesar de que es más común el uso de la palabra *base*: hablamos de sistemas de numeración en *base n*.

Un **sistema de numeración basado en raíz** describe los valores numéricos en función de una o varias raíces. La raíz o base del sistema de numeración indica el número de dígitos diferentes de que se dispone.

Cuando trabajamos con base 10, disponemos de diez símbolos diferentes, que denominamos **dígitos**, para la representación: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. Si la base del sistema de numeración es 2, se dispone de dos dígitos, el 0 y el 1. En base 16, hay dieciséis dígitos diferentes, que se representan por: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

Terminología

Dígito: cada uno de los signos gráficos empleados para representar los números en un sistema de numeración.

Los sistemas de numeración que usan sólo una base reciben el nombre de **sistemas de numeración de base fija**. El sistema de numeración que usamos en nuestra aritmética cotidiana es un sistema de numeración de base fija en que la base de numeración es 10.

Los sistemas de numeración de base mixta

Son los que usan más de una base de numeración. Un ejemplo de este tipo de sistema es el sistema horario, donde los valores vienen dados en función de las bases 24, 60 y 60 (horas, minutos y segundos).

Consideremos el número 321 en nuestro sistema de numeración en base 10. Hemos usado el dígito 3, el dígito 2 y el dígito 1, ordenados de una manera determinada. Estos mismos dígitos ordenados de otro modo (por ejemplo, 213) representan un número diferente, pese a estar constituido por los mismos dígitos. Los sistemas de numeración en los cuales el orden de los dígitos es determinante en la representación numérica se denominan **sistemas posicionales**.

Un **sistema de numeración posicional** es aquél en que la representación de un valor numérico está determinada por una secuencia **ordenada** de dígitos.

A partir de este punto, los análisis y los estudios contenidos en el resto de apartados de este módulo hacen referencia a sistemas de numeración posicionales de base fija, que son los que tienen más interés para el estudio de la representación de la información numérica en los computadores.

1.2. Sistemas de numeración posicionales

Entendemos que el 632 en base 10 representa 6 centenas, 3 decenas y 2 unidades. Es decir, los dígitos tienen peso 100, 10 y 1, respectivamente. Un cambio de orden de los dígitos (por ejemplo, 326), cambia los pesos asociados a cada dígito y, por lo tanto, el número representado. En un sistema de numeración posicional, cada dígito tiene asociado un peso que depende de la posición y de la base de numeración.

Un **sistema de numeración posicional de base fija** es aquél en que un valor numérico X se representa como una secuencia **ordenada** de dígitos, de la manera siguiente:

$$x_{n-1}x_{n-2}\cdots x_1x_0, x_{-1}\cdots x_{-m}$$

donde cada x_i es un dígito tal que $0 \le x_i \le b-1$, donde b es la base del sistema de numeración y x_i es el dígito de la posición i-ésima de la secuencia.

Terminología

Utilizaremos X para referirnos al concepto abstracto de valor numérico. La representación del valor numérico X en base b lo escribiremos de la forma $X_{(b}$ donde b es la base en decimal.

Las posiciones con subíndice negativo corresponden a la **parte fraccionaria** del número, mientras que las posiciones con subíndice positivo corresponden a la **parte entera**. La frontera entre la parte entera y la parte fraccionaria se indica con una **coma**. Los dígitos de la parte entera se consignan a la izquierda de la coma y los de la parte fraccionaria a la derecha de la coma.

El sistema de numeración de base 10 con que trabajamos habitualmente recibe el nombre de sistema **decimal**. De manera análoga, se denomina **sistema hexadecimal** el sistema de numeración en base 16, **sistema octal** el que usa base 8 y sistema **binario**, el que usa base 2. Los dígitos binarios reciben el nombre de bits.

Terminología

Evitaremos utilizar la expresión *parte decimal*, para designar la parte fraccionaria de un número y eludiremos, así, la ambigüedad del término *número decimal*. Un **número decimal** es un número en base 10, no un número con parte fraccionaria.

Consideremos, de nuevo, el número $632_{(10}$. Lo podemos escribir en función de los pesos asociados a cada posición:

$$632_{(10} = 6 \cdot 100 + 3 \cdot 10 + 2 \cdot 1$$

Terminología

Un dígito binario recibe el nombre de *bit*, que es un acrónimo de la expresión inglesa *binary digit*.

Según la definición, el 2 ocupa la posición 0, el 3 la posición 1 y el 6 la posición 2. Podemos reescribir la expresión anterior relacionando los pesos con la base de numeración y con la posición que ocupa cada dígito:

$$632_{(10} = 6 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$$

El $34{,}75_{(10}$ también se puede escribir en función de la base y de las posiciones:

$$34{,}75_{(10} = 3 \cdot 10^1 + 4 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Nota

Según la numeración de posiciones definida, el 7 ocupa la posición –1 y el 5 la posición –2, mientras que el 3 y el 4 (dígitos de la parte entera) ocupan las posiciones 1 y 0, respectivamente.

En general, un sistema de representación numérica posicional de base fija permite expresar un valor numérico en función de la base de numeración y de la posición de cada dígito.

Recordemos que $x^{-k} = \frac{1}{x^k}$

La secuencia de dígitos que representa un valor numérico en un sistema posicional debe ser **ordenada** porque cada posición tiene un peso asociado. Este peso depende de la posición y de la base de numeración. El peso asociado a la posición p es b^p, donde b es la base de numeración.

El número X representado por la secuencia de dígitos $x_{n-1}x_{n-2}\cdots x_1x_0, x_{-1}\cdots x_{-m}$ se puede expresar en función de la base de numeración de la forma:

$$X = \sum_{i=-m}^{n-1} x_i\, b^i = x_{n-1}\cdot b^{n-1} + x_{n-2}\cdot b^{n-2} + \cdots + x_{-m}\cdot b^{-m}$$

donde cada x_i es un dígito tal que $0 \le x_i \le b-1$, donde b es la base del sistema de numeración y x_i el dígito de la posición i-ésima de la secuencia.

Esta expresión se conoce como el **teorema fundamental de la numeración (TFN).***

Terminología

Expresar un número en función de la base de numeración equivale a escribirlo de la forma:

$$x_{n-1}\cdot b^{n-1} + x_{n-2}\cdot b^{n-2} + \cdots$$

*** Abreviaremos *teorema fundamental de la numeración* con la sigla TFN.**

De este teorema se desprende que, además de la secuencia de dígitos, en un sistema posicional de raíz fija hay que conocer la base de numeración para determinar el valor numérico representado.

La secuencia de dígitos 235 es válida en todas las bases más grandes que 5 (porque el 5 no es un dígito válido en bases inferiores a 6). Ahora bien, en bases diferentes representa números diferentes. Por lo tanto, $235_{(6} \neq 235_{(10} \neq 235_{(16}$. La tabla siguiente muestra la correspondencia entre las representaciones de algunos valores:

Base 2	Base 4	Base 8	Base 10	Base 16
0	0	0	0	0
1	1	1	1	1
10	2	2	2	2
11	3	3	3	3
100	10	4	4	4
101	11	5	5	5
110	12	6	6	6
111	13	7	7	7
1000	20	10	8	8
1001	21	11	9	9
1010	22	12	10	A
1011	23	13	11	B
1100	30	14	12	C
1101	31	15	13	D
1110	32	16	14	E
1111	33	17	15	F
10000	100	20	16	10
10001	101	21	17	11
10010	110	22	18	12

Elementos de la tabla

En cada columna se representan los valores numéricos desde el 0 hasta el $18_{(10}$ en la base indicada en la casilla superior de la columna. En cada fila disponemos de la representación del mismo valor numérico en diferentes bases.

1.3. Cambios de base

La secuencia ordenada de dígitos que representa un valor numérico cambia según la base del sistema de numeración, pero hay una relación entre las secuencias de dígitos.

Los **métodos de cambio de base** permiten encontrar la secuencia ordenada de dígitos que representa un valor numérico X en el sistema de numeración en base b', a partir de la representación en el sistema de numeración en base b, es decir:

$$canvi\_a\_base\_b'\left(X_{(b}\right) = X_{(b'}$$

Uso de los cambios de base

Utilizaremos los cambios de base para convertir la representación de un número entre las bases 2, 10 y 16.

En los apartados siguientes, se exponen dos técnicas de cambio de base.

1.3.1. Método basado en el TFN

Si aplicamos el TFN al $324_{(10}$ lo podemos escribir como:

$$324_{(10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$$

Haciendo las operaciones de la derecha en base 10, se obtiene la representación en base 10, que es la que tenemos a la izquierda de la igualdad. Ahora bien, si hacemos las operaciones en base 7, tendremos la representación en base 7. En general, si hacemos las operaciones en base b obtenemos la representación en base b.

Como la dificultad es operar en una base que no sea base 10 (porque no estamos acostumbrados), el método será útil para pasar a base 10.

Cambio de base basado en el TFN

Para cambiar a base 10 el $462_{(7}$:

1) Expresamos el número en función de la base (base 7) según el TFN:

$$462_{(7} = 4 \cdot 7^2 + 6 \cdot 7^1 + 2 \cdot 7^0$$

2) Hacemos las operaciones en la base de llegada (base 10):

$$4 \cdot 7^2 + 6 \cdot 7^1 + 2 \cdot 7^0 = 4 \cdot 49 + 6 \cdot 7 + 2 \cdot 1 = 240_{(10}$$

Las secuencias de dígitos $462_{(7}$ y $240_{(10}$ representan el mismo valor numérico, pero en bases diferentes: base 7 la primera y base 10 la segunda.

Para hallar la representación de $X_{(b}$ en base 10 tenemos que:

1) Expresar $X_{(b}$ en función de la base b, siguiendo el TFN;
2) Hacer las operaciones en base 10.

Cuando $b > 10$ los dígitos de la base b se tienen que cambiar a base 10 antes de hacer las operaciones.

Valores decimales	Dígitos Hexadecimales
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

El método es válido tanto para números enteros como para números con parte fraccionaria.

Cambios de base basados en el TFN

Para pasar a base 10 el número $101100{,}01_{(2}$:

1) Expresamos el número en función de la base (base 2):

$$101100{,}01_{(2} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

2) Hacemos las operaciones en base 10:

$$1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} =$$
$$1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 0{,}5 + 1 \cdot 0{,}25 = 44{,}25_{(10}$$

El $101100{,}01_{(2}$ en base 10 es el $44{,}25_{(10}$.

Para pasar a base 10 el número $AF2C{,}2_{(16}$:

1) Expresamos el número en función de la base (base 16):

$$AF2C{,}2_{(16} = A \cdot 16^3 + F \cdot 16^2 + 2 \cdot 16^1 + C \cdot 16^0 + 2 \cdot 16^{-1}$$

2) Hacemos las operaciones en base 10. En este caso, tenemos que cambiar a base 10 los dígitos hexadecimales antes de hacer las operaciones:

$$A \cdot 16^3 + F \cdot 16^2 + 2 \cdot 16^1 + C \cdot 16^0 + 2 \cdot 16^{-1} =$$
$$10 \cdot 16^3 + 15 \cdot 16^2 + 2 \cdot 16^1 + 12 \cdot 16^0 + 2 \cdot 16^{-1} = 44844{,}125_{(10}$$

El $AF2C{,}2_{(16}$ en base 10 es el $44844{,}125_{(10}$.

1.3.2. Método basado en el teorema de la división entera

Este método de cambio de base consiste en hacer divisiones enteras por la nueva base de numeración de manera iterativa. Los residuos de las divisiones enteras son los dígitos de la representación en la nueva base.

Para cambiar a base 7 el número $317_{(10}$, hacemos divisiones enteras por 7:

$$\begin{aligned} 317 &= 45 \cdot 7 + 2 \\ 45 &= 6 \cdot 7 + 3 \\ 6 &= 0 \cdot 7 + 6 \end{aligned}$$

$317_{(10} = 632_{(7}$

La secuencia de residuos en **orden inverso** nos da la representación en la nueva base. El número $317_{(10}$ en base 7 es el $632_{(7}$.

Como las operaciones se hacen en la base inicial, este método es especialmente útil para pasar de base 10 a otra.

Para el cambio de base de números fraccionarios con este método tenemos que tratar por **separado** la parte entera y la parte fraccionaria.

Para hallar la representación de $X_{(10}$ a base b:

1) Parte entera: sucesivamente, hacer en base 10 la división entera por la nueva base b. Paramos la sucesión de divisiones cuando obtenemos un cociente 0. La secuencia de residuos, tomados del último al primero, es la secuencia de dígitos de izquierda a derecha de la parte entera en la nueva base. Cuando $b > 10$, los residuos se tienen que pasar a dígitos de la nueva base.

2) Parte fraccionaria: sucesivamente, se separa la parte fraccionaria y se multiplica por la nueva base b. Las operaciones se hacen en base 10. Paramos la sucesión de multiplicaciones cuando encontramos un comportamiento periódico o cuando tenemos dígitos suficientes. La secuencia de valores enteros obtenidos al hacer las multiplicaciones tomados del primero al último es la secuencia de dígitos de izquierda a derecha en la nueva base de representación. Cuando $b > 10$ los enteros obtenidos se tienen que pasar a dígitos de la nueva base.

Finalmente, hay que unir la parte entera y la parte fraccionaria obtenidas.

Ejemplo

Un número con parte fraccionaria finita no periódica en una base puede tener una parte fraccionaria infinita periódica en otra base. Por ejemplo,

$$0,3_{(10} = 0,0100\overline{1100}_{(2}.$$

Cambios de base por el método de la división entera

Cambiar a base 2 el $44,25_{(10}$:

a) Parte entera: sucesivamente, hacemos divisiones enteras por la nueva base (base 2) hasta obtener un cociente 0, y tomamos los residuos en orden inverso:

$$\begin{array}{rl} 44 = & 22 \cdot 2 + 0 \\ 22 = & 11 \cdot 2 + 0 \\ 11 = & 5 \cdot 2 + 1 \\ 5 = & 2 \cdot 2 + 1 \\ 2 = & 1 \cdot 2 + 0 \\ 1 = & 0 \cdot 2 + 1 \end{array} \uparrow$$

$44_{(10} = 101100_{(2}$

b) Parte fraccionaria: sucesivamente, multiplicamos por la nueva base (base 2):

$$\begin{array}{l} 0,25 \cdot 2 = 0,50 = 0,50 + 0 \\ 0,50 \cdot 2 = 1,00 = 0,00 + 1 \end{array} \downarrow$$

$0,25_{(10} = 0,01_{(2}$

Para completar el cambio de base, unimos la parte entera y la parte fraccionaria que resultan:

$$44,25_{(10} = 101100_{(2} + 0,01_{(2} = 101100,01_{(2}$$

El $44,25_{(10}$ en base 2 es el $101100,01_{(2}$.

Cambiar a base 16 el $44844,12_{(10}$:

a) Parte entera: sucesivamente, hacemos divisiones enteras por 16 hasta obtener un cociente 0:

$$\begin{array}{rl} 44844 = & 2802 \cdot 16 + 12 \\ 2802 = & 175 \cdot 16 + 2 \\ 175 = & 10 \cdot 16 + 15 \\ 10 = & 0 \cdot 16 + 10 \end{array} \uparrow$$

Como la nueva base es mayor que 10, tenemos que convertir los residuos a la nueva base (base 16):

$$12_{(10} = C_{(16}$$
$$2_{(10} = 2_{(16}$$
$$15_{(10} = F_{(16}$$
$$10_{(10} = A_{(16}$$

$$44844_{(10} = AF2C_{(16}$$

b) Parte fraccionaria: sucesivamente, multiplicamos por 16:

$$\begin{aligned}
0{,}12 \cdot 16 &= 1{,}92 = 0{,}92 + 1 \\
0{,}92 \cdot 16 &= 14{,}72 = 0{,}72 + 14 \\
0{,}72 \cdot 16 &= 11{,}52 = 0{,}52 + 11 \\
0{,}52 \cdot 16 &= 8{,}32 = 0{,}32 + 8 \\
0{,}32 \cdot 16 &= 5{,}12 = 0{,}12 + 5 \\
0{,}12 \cdot 16 &= 1{,}92 = 0{,}92 + 1 \\
0{,}92 \cdot 16 &= 14{,}72 = 0{,}72 + 14 \\
&\ldots
\end{aligned}$$

La secuencia de enteros que obtenemos se repite (1,14,11,8,5,1,14,...). Por lo tanto, es un número periódico. Además, los enteros se tienen que convertir a dígitos de base 16:

$$1_{(10} = 1_{(16}$$
$$14_{(10} = E_{(16}$$
$$11_{(10} = B_{(16}$$
$$8_{(10} = 8_{(16}$$
$$5_{(10} = 5_{(16}$$
$$1_{(10} = 1_{(16}$$

$$0{,}12_{(10} = 0{,}1EB851EB851EB\ldots_{(16} = 0{,}\widehat{1EB85}_{(16}$$

Finalmente, uniremos la parte entera y la parte fraccionaria:

$$44844{,}12_{(10} = AF2C{,}\widehat{1EB85}_{(16}$$

El $44844{,}12_{(10}$ en base 16 es el $AF2C{,}\widehat{1EB85}_{(16}$.

Para cambiar de base *b* a base *b'*, donde ni *b* ni *b'* son la base 10, utilizamos la base 10 como base intermedia. Así, usamos el primer método (basado en el TFN) para pasar de base *b* a base 10 y, posteriormente, el segundo método (basado en el teorema de la división entera) para pasar de base 10 a base *b'*.

Cambio entre bases diferentes de la base 10

El cambio a base 6 del $232{,}1_{(4}$ lo tenemos que hacer en dos pasos:

1) Hacemos el cambio a base 10 del $232{,}1_{(4}$ aplicando el método del TFN:

$$\begin{aligned}
232{,}1_{(2} &= 2 \cdot 4^2 + 3 \cdot 4^1 + 2 \cdot 4^0 + 1 \cdot 4^{-1} = \\
&= 32 + 12 + 2 + 0{,}25 = 46{,}25_{(10}
\end{aligned}$$

2) Hacemos el cambio a base 6 del $46{,}25_{(10}$:

$$\begin{aligned}
46 &= 7 \cdot 6 + 4 \\
7 &= 1 \cdot 6 + 1 \\
1 &= 0 \cdot 6 + 1
\end{aligned}$$

$$\begin{aligned}
0{,}25 \cdot 6 &= 1{,}50 = 0{,}50 + 1 \\
0{,}50 \cdot 6 &= 3{,}00 = 0{,}00 + 3
\end{aligned}$$

$$46_{(10} = 114_{(6} \qquad 0{,}25_{(10} = 0{,}13_{(6}$$

El $46{,}25_{(10}$ es equivalente al $114{,}13_{(6}$

Por lo tanto, el $232{,}1_{(4}$ es el $114{,}13_{(6}$ en base 6.

Potencias de 2	
2^{16}	65536
2^{15}	32768
2^{14}	16384
2^{13}	8192
2^{12}	4096
2^{11}	2048
2^{10}	1024
2^{9}	512
2^{8}	256
2^{7}	128
2^{6}	64
2^{5}	32
2^{4}	16
2^{3}	8
2^{2}	4
2^{1}	2
2^{0}	1
2^{-1}	0,5
2^{-2}	0,25
2^{-3}	0,125
2^{-4}	0,0625
2^{-5}	0,03125
2^{-6}	0,015625
2^{-7}	0,0078125
2^{-8}	0,00390625

1.3.3. Cambio de base entre b y b^n

El cambio de base b a base b^n es directo, porque...

> ...un dígito en base b^n se corresponde con n dígitos en base b.

Esta circunstancia se da entre base 2 y base 16 (porque $16 = 2^4$) o entre base 16 y base 4 (porque $16 = 4^2$), pero no entre base 8 y base 16, porque 16 no es potencia de 8.

Cambio de base b a base b^n

En el cambio a base 16 del $10010110{,}01101101_{(2}$, tendremos en cuenta que 16 es potencia de 2: $16 = 2^4$. Esta relación indica que cada dígito de base 16 se corresponde con cuatro dígitos de base 2.

El cambio de base se consigue si hacemos agrupaciones de cuatro dígitos binarios, y convertimos cada agrupación en un dígito hexadecimal. Las agrupaciones se hacen siempre partiendo de la coma fraccionaria, y tienen que ser completas. Si faltan dígitos para completar una agrupación, añadiremos ceros.

Ved la correspondencia entre binario y hexadecimal en la tabla del subapartado 1.2

1001	0110	,	0110	1101	(2
9	6	,	6	D	(16

El $10010110{,}01101101_{(2}$ es en base 16 el $96{,}6D_{(16}$.

En el cambio a base 16 del $101110{,}101101_{(2}$ tenemos que completar las agrupaciones añadiendo ceros (en este caso, tanto en la parte entera como la fraccionaria):

0010	1110	,	1011	01**00**	(2
2	E	,	B	4	(16

$$101110{,}101101_{(2} = 2E{,}B4_{(16}$$

Cambio de base b^n a base b

Cuando el cambio es de base b^n a b, el procedimiento es análogo pero en sentido inverso: cada dígito en base b^n se transforma en n dígitos en base b.

Para cambiar a base 2 el $7632{,}13_{(8}$, tendremos en cuenta que $8 = 2^3$. Por consiguiente, cada dígito en base 8 dará lugar a tres dígitos binarios:

7	6	3	2	,	1	3	(8
111	110	011	010	,	001	011	(2

$$7632{,}13_{(8} = 111110011010{,}001011_{(2}$$

Debemos prestar atención al hecho de que hay que obtener exactamente *n* dígitos en base *b* por cada dígito en base b^n (en este caso, tres dígitos binarios por cada dígito octal), añadiendo para cada dígito los ceros necesarios. Veamos cómo en el cambio a base 2 del $E1B2,4F_{(16}$ cada dígito hexadecimal da lugar a cuatro dígitos binarios.

E	1	B	2	,	4	F	$_{(16}$
1110	0001	1011	0010	,	0100	1111	$_{(2}$

$$E1B2,4F_{(16} = 1110000110110010,01001111_{(2}$$

Errores frecuentes

A menudo se cometen dos errores en estos tipos de cambio de base:

1) Cuando hacemos un cambio de base b^n a base *b*, cada dígito de base b^n tiene que dar lugar, exactamente, a *n* dígitos en base *b*. Hay que evitar el error siguiente:

$$A3_{(16} = 101011_{(2}$$

donde el dígito A ha dado lugar a los bits 1010 y el dígito 3 a los bits 11. En realidad, ha de ser:

$$A3_{(16} = 10100011_{(2}$$

donde se han añadido dos ceros para completar el conjunto de cuatro dígitos que debe generar el dígito hexadecimal 3.

2) Cuando hacemos un cambio de base *b* a base b^n, son necesarios *n* dígitos de base *b* para obtener un dígito en base b^n. Hay que evitar el error siguiente:

$$1100,11_{(2} = C,3_{(16}$$

donde los bits 1100 dan lugar al dígito hexadecimal C y los bits 11 al 3. En realidad, ha de ser:

$$1100,1100_{(2} = C,C_{(16}$$

donde se han añadido 2 ceros a la derecha con objeto de constituir un grupo de cuatro dígitos binarios que dan lugar al dígito hexadecimal C.

1.4. Empaquetamiento de la información

Con los cambios a base 2 tenemos un camino abierto para procesar los números dentro de los computadores. De hecho, dentro de los computadores, toda la in-

formación (no sólo la numérica) se codifica utilizando únicamente el símbolo 1 y el símbolo 0. Por lo tanto, toda la información que procesa un computador está codificada en cadenas de unos y de ceros, es decir, en cadenas de bits.

Ahora bien, disponer sólo de dos símbolos nos lleva a representaciones con un gran número de dígitos, a cadenas de bits largas, que para nosotros (no para los computadores) son difíciles de recordar y de manipular.

Pues bien, podemos aprovechar la técnica de hacer agrupaciones de cuatro bits (en vista de los cambios de base 2 a base 16) para convertir las cadenas de bits en dígitos hexadecimales y compactarlas, así, en cadenas mucho más cortas y manejables. Este proceso recibe el nombre de **empaquetamiento hexadecimal**. El proceso inverso se denomina **desempaquetamiento**.

El **empaquetamiento hexadecimal** consiste en compactar información binaria en cadenas de dígitos hexadecimales.

Habitualmente se coloca el símbolo 'h' al final de la cadena de dígitos, para indicar que son hexadecimales.

Empaquetamiento de una cadena de bits

Para empaquetar la cadena de bits 110100100011, procedemos de la forma siguiente:

1) Dividimos la cadena 110100100011 de derecha a izquierda en grupos de 4 bits:

1101 – 0010 – 0011

2) Codificamos cada grupo de 4 bits como un dígito hexadecimal:

1101– 0010 – 0011
D – 2 – 3

Por lo tanto, si hacemos el empaquetamiento hexadecimal de la cadena de bits 110100100011, se obtiene D23h.

El empaquetamiento hexadecimal es ampliamente utilizado en diferentes ámbitos relacionados con los computadores para facilitar el trabajo con números, instrucciones y direcciones de memoria. Este tipo de empaquetamiento se emplea sobre cadenas de bits, con independencia del sentido que tengan los bits de la cadena.

El proceso inverso, el desempaquetamiento, permite recuperar la cadena de bits original. En este caso, cada dígito hexadecimal da lugar a 4 bits. Así, el dígito hexadecimal 4 daría lugar al grupo de 4 bits 0100 y no al 100.

Desempaquetamiento

Para desempaquetar la cadena D23h, convertimos los dígitos hexadecimales a base 2 usando 4 bits para cada uno:

D – 2 – 3
1101– 0010 – 0011

Por lo tanto, si desempaquetamos la cadena de dígitos hexadecimales D23h, se obtiene la cadena de bits 110100100011.

Es importante diferenciar el concepto de empaquetamiento hexadecimal de cadenas de bits del concepto del cambio de base 2 a base 16. Cuando hagamos un cambio de base 2 a base 16 de un número, debemos tener presente la posición de la coma fraccionaria, porque buscamos la representación del mismo número pero en base 16. Por lo tanto, las agrupaciones de 4 bits se hacen a partir de la coma fraccionaria: hacia la izquierda de la coma, para obtener los dígitos hexadecimales enteros y hacia la derecha para conseguir los dígitos hexadecimales fraccionarios. En cambio, en el empaquetamiento hexadecimal no se tiene en cuenta el sentido de la información codificada y los bits se agrupan de 4 en 4 de derecha a izquierda independientemente de su sentido. En este caso, lo que obtenemos finalmente no es la representación del número en base 16, sino una cadena de dígitos hexadecimales que codifican una cadena de bits.

Veamos esta diferencia según hagamos el cambio a base 16 del número $111010{,}11_{(2}$ o el empaquetamiento hexadecimal. Si queremos hacer el cambio a base 16, tenemos que hacer agrupaciones a partir de la coma fraccionaria, añadiendo los ceros necesarios para completar las agrupaciones tanto por la derecha como por la izquierda:

$$\begin{array}{cccc} \mathbf{0}011 & 1010 & , & 11\mathbf{00} \\ 3 & A & , & C \end{array} \begin{array}{l} {}_{(2} \\ {}_{(16} \end{array}$$

En este caso, el resultado que se obtiene indica que el número $111010{,}11_{(2}$ en base 16 es el $3A{,}C_{(16}$.

En cambio, si queremos hacer un empaquetamiento hexadecimal, las agrupaciones se hacen de derecha a izquierda, sin tener en cuenta la posición de la coma. Se trata como una tira de unos y ceros. El resultado final no guarda información sobre la coma fraccionaria:

$$\begin{array}{cc} 1110 & 10{,}11_{(2} \\ E & B \end{array}$$

En este segundo caso, el resultado que se obtiene indica que el empaquetamiento hexadecimal de la cadena de bits 11101011 es EBh. Podemos comprobar que la secuencia de dígitos hexadecimales que se obtiene en uno y otro caso puede ser diferente.

Actividades

1. Convertid a base 10 los valores siguientes:

a) $10011101_{(2}$
b) $3AD_{(16}$
c) $333_{(4}$

d) $333_{(8}$
e) $B2,3_{(16}$
f) $3245_{(8}$
g) $AC3C_{(16}$
h) $1010,11_{(8}$
i) $110011,11_{(4}$
j) $10011001,1101_{(2}$
k) $1110100,01101_{(2}$

2. Convertid a base 2 los valores siguientes:
a) $425_{(10}$
b) $344_{(10}$
c) $31,125_{(10}$
d) $4365,14_{(10}$

3. Convertid a hexadecimal los números siguientes:
a) $111010011,1110100111_{(2}$
b) $0,1101101_{(2}$
d) $45367_{(10}$
c) $111011,1010010101_{(2}$

4. Convertid los números hexadecimales siguientes a base 2, base 4 y base 8:
a) $ABCD_{(16}$
b) $45,45_{(16}$
c) $96FF,FF_{(16}$

5. Rellenad la tabla siguiente:

Binario	Octal	Hexadecimal	Decimal
1101100,110			
	362,23		
		A1,03	
			74,3

En cada fila veréis un valor numérico expresado en la base que indica la casilla superior de la columna donde se encuentra. Consignad en el resto de casillas la representación correspondiente según la base indicada en la parte superior.

6. Empaquetad en hexadecimal la cadena de bits 10110001.

7. Empaquetad en hexadecimal el número $0100000111,111010_{(2}$ que está en un formato de coma fija de 16 bits, de los cuales 6 son fraccionarios.

8. Desempaquetad la cadena de bits A83h y,
a) Encontrad el valor decimal si se trata de un número natural.
b) Encontrad el valor decimal si se trata de un número en coma fija sin signo de 12 bits, donde 4 son fraccionarios.

9. Consideremos el número $1010,101_{(2}$.
a) Haced el cambio a base 16.
b) Haced el empaquetamiento hexadecimal.

1.5. Números con signo

Cuando representamos magnitudes, a menudo les asignamos un signo (+/–) que precede a la magnitud y que indica si la magnitud es positiva o negativa. El símbolo – identifica las magnitudes negativas y el símbolo + las positivas:

Signo (+/–)

A veces, cuando se trabaja con números con signo, el signo positivo (+) no se escribe y sólo aparece el signo cuando se trata de un número negativo.

$+23_{(10}$ $\quad$ $-456_{(8}$
$-34,5_{(7}$ $\quad$ $+AF,34_{(16}$

Designaremos los números que llevan la información de signo como números con signo, en contraposición a los números sin signo, que sólo nos dan información sobre la magnitud del valor numérico.

1.6. Suma en los sistemas posicionales

El algoritmo de suma de dos números decimales que estamos habituados a utilizar progresa de derecha a izquierda, sumando en cada etapa los dígitos del mismo peso (los que ocupan la misma posición). Si la suma de estos dígitos llega al valor de la base (10 en este caso), genera un **acarreo** (lo que nos "llevamos") que se sumará con los dígitos de la etapa siguiente:

$$
\begin{array}{rrrrrrl}
 & 1 & & 1 & & & \leftarrow \text{dígito de acarreo*} \\
 & & 8 & 3 & 4 & 1 & {}_{(10} \\
+ & & 2 & 4 & 6 & 3 & {}_{(10} \\
\hline
 & 1 & 0 & 8 & 0 & 4 & {}_{(10} \quad \leftarrow \text{resultado}
\end{array}
$$

*** El dígito de acarreo recibe en inglés el nombre de *carry*. Este término es de uso habitual en el entorno de los computadores.**

En hexadecimal, se siguen las mismas pautas de suma, pero teniendo en cuenta que hay 16 dígitos diferentes:

$$
\begin{array}{rrrrrl}
 & & 1 & & & \leftarrow \text{acarreo} \\
 & & 3 & 5 & 8 & 2 \; {}_{(16} \\
+ & & A & F & 1 & 8 \; {}_{(16} \\
\hline
 & & E & 4 & 9 & A \; {}_{(16} \quad \leftarrow \text{resultado}
\end{array}
$$

El proceso de suma en base 2 es análogo:

$$
\begin{array}{rrrrrrrrrrl}
 & 1 & 1 & 1 & 1 & 1 & 1 & & & & \leftarrow \text{acarreo} \\
 & & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & {}_{(2} \\
+ & & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & {}_{(2} \\
\hline
 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & {}_{(2} \quad \leftarrow \text{resultado}
\end{array}
$$

Tabla de suma en base 2 acarreo / bit de suma

+	0	1
0	0/0	0/1
1	0/1	1/0

Cuando se produce un acarreo en la última etapa de suma, el resultado tiene un dígito más que los sumandos.

1.7. Resta en los sistemas posicionales

La operación de resta también se lleva a cabo de derecha a izquierda, operando los dígitos de igual peso, y considerando el acarreo* de la etapa precedente. La

*** En inglés el acarreo en el caso de la resta recibe el nombre de *borrow* .**

particularidad en esta operación es que el número de menor magnitud (sustraendo) es el que hay que restar del número de mayor magnitud (minuendo):

$$
\begin{array}{rcccccl}
 & 8 & 3 & 4 & 1 & {}_{(10} & \leftarrow \text{minuendo} \\
 & 1 & 1 & 1 & & & \leftarrow \text{acarreo} \\
- & 2 & 4 & 6 & 3 & {}_{(10} & \leftarrow \text{sustraendo} \\
\hline
 & 5 & 8 & 7 & 8 & {}_{(10} & \leftarrow \text{resultado}
\end{array}
$$

El procedimiento en otras bases es idéntico. Sólo hay que adecuarse a la nueva base y poner atención en restar la magnitud pequeña de la grande:

$$
\begin{array}{rcccccl}
 & A & F & 1 & 8 & {}_{(16} & \\
 & & 1 & & & & \leftarrow \text{acarreo} \\
- & 3 & 5 & 8 & 2 & {}_{(16} & \\
\hline
 & 7 & 9 & 9 & 6 & {}_{(16} & \leftarrow \text{resultado}
\end{array}
\qquad
\begin{array}{rcccccccccl}
 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & {}_{(2} & \\
 & 1 & 1 & 1 & & & & & & & \leftarrow \text{acarreo} \\
- & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & {}_{(2} & \\
\hline
 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
$$

Tabla de resta en base 2
acarreo / bit de resta

–		minuendo	
		0	**1**
sustraendo	**0**	0/0	0/1
	1	1/1	0/0

En el caso de la operación de resta no se puede producir ningún acarreo en la última etapa. Por este motivo, el resultado de una resta de números necesita, como máximo, los mismos dígitos que el minuendo.

1.8. Multiplicación y división por potencias de la base de numeración

En un sistema posicional de base fija cada dígito tiene un peso b^p donde b es la base de numeración y p la posición que ocupa el dígito. Los pesos asociados a los dígitos de los números decimales son potencias de 10. Por lo tanto, multiplicar por 10 se traduce en aumentar en una unidad la potencia de 10 asociada a cada dígito y dividir por 10 es equivalente a disminuir en una unidad la potencia de 10 asociada a cada dígito.

Consultad los sistemas de numeración posicionales del subapartado 1.2 de este módulo.

Según el TFN, el número $56{,}34_{(10} = 5{\cdot}10^1 + 6{\cdot}10^0 + 3{\cdot}10^{-1} + 4{\cdot}10^{-2}$. Si multiplicamos por 10 tenemos:

$$
\begin{aligned}
56{,}34_{(10} \cdot 10 &= (5 \cdot 10^1 + 6 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2}) \cdot 10 = \\
&= 5 \cdot 10^2 + 6 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} = 563{,}4_{(10}.
\end{aligned}
$$

El efecto que se obtiene es el desplazamiento de la coma fraccionaria. Multiplicar por 10 un número en base 10 es equivalente a desplazar la coma fraccionaria una posición a la derecha, mientras que dividirlo por 10 es equivalente a desplazar la coma una posición a la izquierda. El proceso se puede extender a la multiplicación y división por una potencia de 10: multiplicar por 10^k un número en base 10 equivale a desplazar la coma fraccionaria k posiciones a la derecha, y dividirlo por 10^k equivale a desplazar la coma fraccionaria k posiciones a la izquierda.

Este proceso de multiplicación y división por potencias de la base de numeración es válido en todos los sistemas posicionales de base fija *b*.

Multiplicar por b^k un número en un sistema posicional de base fija *b* equivale a desplazar la coma fraccionaria *k* posiciones a la derecha.

Dividir por b^k un número en un sistema posicional de base fija *b* equivale a desplazar la coma fraccionaria *k* posiciones a la izquierda.

Números sin parte fraccionaria

En un número sin parte fraccionaria desplazar la coma *k* posiciones a la derecha equivale a añadir *k* ceros a la derecha, dado que la parte fraccionaria es cero.

Multiplicación por una potencia de 2 en binario

El resultado de multiplicar el $11010_{(2}$ por 2^4 se consigue si desplazamos la coma fraccionaria 4 posiciones a la derecha:

$$11010_{(2} \cdot 2^4 = 110100000_{(2}$$

Este resultado que obtenemos de forma directa se puede justificar con los cálculos siguientes:

$$\begin{aligned} 11010_{(2} \cdot 2^4 &= (1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0) \cdot 2^4 = \\ &= (1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4) = 110100000_{(2} \end{aligned}$$

Por lo tanto, $11010_{(2} \cdot 2^4 = 110100000_{(2}$.

División por una potencia de 2 en binario

El resultado de dividir el $11100_{(2}$ por 2^2 se consigue si desplazamos la coma fraccionaria 2 posiciones a la izquierda:

$$11100_{(2} / 2^2 = 111_{(2}$$

Este resultado que obtenemos de forma directa se puede justificar con los cálculos siguientes:

$$\begin{aligned} 11100_{(2} / 2^2 &= (1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0) / 2^2 = \\ &= (1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2}) = 111_{(2} \end{aligned}$$

Por lo tanto, $11100_{(2} / 2^2 = 111_{(2}$.

La división por una potencia de la base de numeración de un número sin parte fraccionaria puede dar como resultado un número con parte fraccionaria: $11100_{(2} / 2^4 = 1{,}11_{(2}$. Ahora bien, podemos dar el resultado en forma de dos números enteros que reciben el nombre de cociente y resto, donde el cociente tiene relación directa con la parte entera del resultado y el resto con la parte fraccionaria. En este caso, la operación recibe el nombre de división **entera**, mientras que, por oposición, la primera recibe el nombre de división **real**.

El cociente y el resto de la división entera de $11100_{(2}$ por 2^4 se pueden obtener a partir del resultado de la división real $11100_{(2} / 2^4 = 1{,}11_{(2}$, identificando el cociente con la parte entera (en este caso, el cociente es $1_{(2)}$) y el resto con la parte fraccionaria multiplicada por el divisor (en este caso, el resto es $0{,}11_{(2} \cdot 2^4 = 1100_{(2)}$).

El **cociente** y el **resto** de una división entera de un número entero por una potencia de la base de numeración se pueden obtener a partir del resultado de división real, identificando el cociente con la parte entera y el resto con la parte fraccionaria multiplicada por el divisor.

Actividades

10. Calculad las operaciones siguientes en la base especificada:
a) $111011010_{(2} + 100110100_{(2}$
b) $2345_{(8} + 321_{(8}$
c) $A23F_{(16} + 54A3_{(16}$
d) $111011010_{(2} - 100110100_{(2}$
e) $2345_{(8} - 321_{(8}$
f) $A23F_{(16} - 54A3_{(16}$

11. Calculad las operaciones siguientes en la base especificada:
a) $62{,}48_{(16} + 35{,}DF_{(16}$
b) $111101101{,}11011_{(2} + 100110100{,}111_{(2}$
c) $62{,}48_{(16} - 35{,}DF_{(16}$
d) $111101101{,}11011_{(2} - 100110100{,}111_{(2}$

12. Calculad las multiplicaciones siguientes:
a) $128{,}7_{(10} \cdot 10^4$
b) $AFD_{(16} \cdot 16^2$
c) $1101{,}01_{(2} \cdot 2^2$

13. Hallad el cociente y el residuo de las divisiones enteras siguientes:
a) $52978_{(10} / 10^3$
b) $3456_{(16} / 16^2$
c) $100101001001_{(2} / 2^8$

2. Representación de los números en un computador

En esta segunda sección se describen sistemas para representar números que se usan para codificar información numérica dentro de los computadores.

Terminología

A lo largo del texto, utilizaremos indistintamente **representación** y **codificación** para referirnos a la secuencia de dígitos asociada a un valor numérico en un sistema de representación numérica.

2.1. Condicionantes físicos

A pesar de las continuas mejoras tecnológicas, la capacidad de almacenamiento de los computadores es finita. Esto condiciona la representación numérica dentro de los computadores, sobre todo en números con una parte fraccionaria infinita, como por ejemplo los casos muy conocidos de los números π o e y, en general, en la representación de números irracionales como $\sqrt{2}$.

Estas limitaciones son parecidas a las que encontramos cuando trabajamos con lápiz y papel. En los cálculos hechos a mano usamos el $3{,}14_{(10}$ o el $3{,}1416_{(10}$ como aproximación a π. Dentro de los computadores también se trabaja con aproximaciones de los números que no se pueden representar de manera exacta.

Cuando un número no se puede representar de manera exacta dentro de un computador, se comete un **error de representación**. Este error es la distancia entre el número que queremos representar y el número representado realmente.

Números representables

Los números que se pueden representar de forma exacta reciben el nombre de *números representables*.

Si representamos el número π por el valor $3{,}14_{(10}$, estamos cometiendo un error igual a $|\pi - 3{,}14_{(10}| = 0{,}00159...$, mientras que si trabajamos con el valor $3{,}1416_{(10}$ el error es $|\pi - 3{,}1416_{(10}| = 7{,}3464102...\cdot 10^{-6}$.

Cuando escribimos números lo hacemos de la forma más práctica y adecuada en cada caso. Podemos escribir 03, 3,00, 3,000 o simplemente 3. En cambio, dentro de los computadores hay que seguir una pauta más rígida, un **formato** que especifique y fije el número de dígitos enteros y fraccionarios con que se trabaja. Si suponemos un formato de representación de la forma $x_2x_1x_0{,}x_{-1}x_{-2}$, donde cada x_i es un dígito binario, el número $3_{(10}$ se tiene que representar como $011{,}00_{(2}$.

Un **formato** de representación numérica es la manera específica en que se tienen que representar los valores numéricos con que se trabaja. El formato fija el conjunto de números que se pueden representar.

En los subapartados siguientes se describen los parámetros que nos ayudan a medir la eficiencia de un formato de representación numérica: el rango de representación, la precisión y el error de representación.

2.1.1. Rango de representación

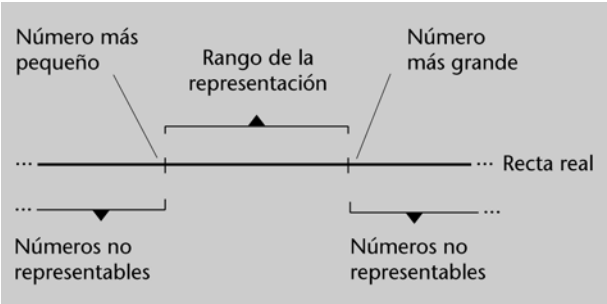
Fijado el formato $x_1x_0,x_{-1}x_{-2}$ en la base 10, sólo podemos representar números entre el $00{,}00_{(10}$ (el número más pequeño representable en este formato) y el $99{,}99_{(10}$ (el número más grande representable en este formato). El número $935_{(10}$ no se puede representar en este formato puesto que no está dentro del intervalo de representación. Los números que se pueden representar en un formato están delimitados dentro de un **intervalo** que recibe el nombre de **rango**.

Atención

En un formato sólo se pueden representar un conjunto de números. En un formato con rango [0, 99,99] el número $34{,}789_{(10}$ no se puede representar de forma exacta, porque tiene 3 dígitos fraccionarios.

El **rango** de un formato de representación numérica es el intervalo más pequeño que contiene todos los números representables. Los límites del intervalo son determinados por el número más grande y el número más pequeño que se pueden representar.

La notación que se usa para definir un rango es [*a, b*] donde *a* y *b* son los límites del intervalo en decimal, y forman parte de él.



Los números que están fuera del rango de representación de un formato no son representables en ese formato.

Hay una limitación inherente al número de bits disponibles en un formato de representación: con *n* dígitos en base *b*, disponemos de b^n códigos o combinaciones de dígitos. Cada una de estas combinaciones puede representar un valor numérico. Por lo tanto, con *n* dígitos en base *b* podremos representar un máximo de b^n números diferentes.

Con 5 bits disponemos de $2^5 = 32$ combinaciones diferentes. Podremos representar 32 números. La codificación que se use determinará cuáles son estos números. En base 10 y 4 dígitos disponemos de 10^4 códigos diferentes (combinaciones de los 4 dígitos decimales). Si empleamos estos códigos para representar números naturales, podremos representar desde el 0000 (0), hasta el 9999 ($10^4 - 1$).

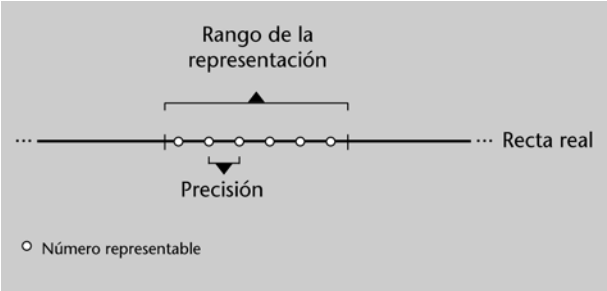
2.1.2. Precisión

Estamos habituados a trabajar de manera dinámica con la precisión y la ajustamos automáticamente a nuestras necesidades. Para medir la longitud de una mesa en metros, por ejemplo, trabajamos con dos dígitos fraccionarios. Un formato de estas características nos permitirá distinguir 1,52 m de 1,53 m, pero no de 1,5234 m. Decimos que la precisión de este formato es de 0,01 m, que es la distancia entre dos valores consecutivos representables en este formato.

La **precisión** de un formato de representación numérica es la distancia entre dos números representables consecutivos.

Nota

En la mayoría de formatos de representación la distancia entre dos números representables consecutivos cualesquiera es la misma.



2.1.3. Error de representación

En un formato de 4 dígitos decimales, de los cuales 2 son fraccionarios, los números son de la forma $x_1x_0{,}x_{-1}x_{-2}$, donde x_i es un dígito decimal. Podemos representar el $12{,}34_{(10}$ o el $45{,}20_{(10}$ de manera exacta, pero no el $15{,}026_{(10}$. Si tenemos que trabajar con este número tendremos que usar una **aproximación**. Podemos aproximarlo por un número representable cercano como el $15{,}03_{(10}$. Trabajar con una aproximación comporta cometer un error. En este caso, el error que se comete es $15{,}03_{(10} - 15{,}026_{(10} = 0{,}004_{(10}$.

Rangos de representación

En el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10, el rango de representación es [0, 99,99]. Un número como el $128_{(10}$, que está fuera del rango de representación, no es representable. No se considera que $99{,}99_{(10}$ sea una aproximación válida para este número en este formato.

El **error de representación** ε es la distancia entre el número X que queremos representar y el número representable $\widehat{X}$ con el que lo aproximamos. Es decir, $\varepsilon = \left|X - \widehat{X}\right|$.

Los números que no están dentro del rango de representación del formato no son representables ni aproximables. !

2.1.4. Aproximaciones: truncamiento y redondeo

En el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10, tanto el $23{,}45_{(10}$ como el $23{,}46_{(10}$ son aproximaciones válidas del $23{,}457_{(10}$. Tenemos que elegir una de las dos po-

sibilidades, por lo cual estableceremos un criterio de elección. Este proceso de elección se denomina **aproximación** o **cuantificación**. Los criterios de elección más habituales son el **truncamiento** y el **redondeo**.

1) Truncamiento

El truncamiento es el criterio de cuantificación más directo y sencillo de aplicar, puesto que no comporta ningún tipo de cálculo y consiste en ignorar los dígitos que sobran. En el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10 este criterio aproxima el número $23{,}457_{(10}$ por el $23{,}45_{(10}$, fruto de ignorar el último dígito, que no cabe en el formato.

La cuantificación o aproximación por **truncamiento** consiste en despreciar los dígitos fraccionarios que no caben en el formato. El proceso de truncamiento no comporta ningún tipo de cálculo.

Truncamiento

La gran ventaja del truncamiento es que no comporta ningún tipo de cálculo aritmético.

Por truncamiento en el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10, tanto el $23{,}451_{(10}$, el $23{,}456_{(10}$ como el $23{,}459_{(10}$ se aproximan por el $23{,}45_{(10}$. Ahora bien, el error cometido en cada caso es diferente. El error es 0,001 para el $23{,}451_{(10}$ (puesto que $|23{,}451_{(10} - 23{,}45_{(10}| = 0{,}001$), 0,006 para el $23{,}456_{(10}$ y 0,009 para el $23{,}459_{(10}$. En todos los casos el error de representación es inferior a la precisión, que es 0,01.

En una aproximación por truncamiento el **error máximo** de representación es igual a la precisión del formato de representación.

2) Redondeo

El $23{,}459_{(10}$ en el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10, se aproxima por truncamiento por el $23{,}45_{(10}$ y el error es 0,009. Ahora bien, si aproximáramos el $23{,}459_{(10}$ por el $23{,}46_{(10}$, el error sería 0,001 ($|23{,}46_{(10} - 23{,}459_{(10}| = 0{,}001$), es decir, un error más pequeño. El $23{,}46_{(10}$ está más cerca y sería más exacto trabajar con él. Esta aproximación recibe el nombre de **redondeo** o *aproximación* **al más próximo**.

Nota

El número que se obtiene por truncamiento coincide con el que se obtiene por redondeo, siempre que el número resultante por truncamiento sea el número representable más cercano al número que queremos representar.

La cuantificación o aproximación por **redondeo** consiste en escoger el número representable más cercano al número que queremos representar. El proceso de redondeo comporta operaciones aritméticas.

Si aplicamos el criterio de redondeo en el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10, el $23{,}451_{(10}$ se aproxima por el $23{,}45_{(10}$, mientras que el $23{,}456_{(10}$ o el $23{,}459_{(10}$ se aproximan por el $23{,}46_{(10}$ que les es más cercano. El error es 0,001 para el

$23{,}451_{(10}$, 0,004 para el $23{,}456_{(10}$ y 0,001 para el $23{,}459_{(10}$. El error cometido es inferior a la mitad de la precisión, es decir, inferior a 0,005 en este caso.

En una aproximación por redondeo el **error máximo** de representación es igual a la mitad de la precisión del formato de representación.

Una manera sencilla de aplicar el redondeo al número $23{,}459_{(10}$ en el formato $x_1x_0{,}x_{-1}x_{-2}$ en base 10 es sumarle la mitad de la precisión (es decir, 0,005) y a continuación hacer el truncamiento del resultado: $23{,}459_{(10} + 0{,}005_{(10} = 23{,}464_{(10}$, que truncado a dos dígitos fraccionarios es el $23{,}46_{(10}$.

Para aproximar un número por redondeo tenemos que:

1) Sumar la mitad de la precisión del formato de representación al número que se quiere aproximar.
2) Truncar el resultado de la suma según el número de dígitos fraccionarios disponibles en el formato de representación.

Aproximación por redondeo

Para aproximar por redondeo el número $1{,}526246_{(10}$ en el formato $x_1x_0x_{-1}x_{-2}x_{-3}x_{-4}$ en base 10 procederemos de la forma siguiente:

1) Sumar la mitad de la precisión del formato de representación al número que se quiere aproximar:

$$1{,}526264_{(10} + 0{,}00005_{(10} = 1{,}526314_{(10}$$

2) Truncar el resultado de la suma según el número de dígitos fraccionarios disponibles en el formato de representación:

$$1{,}526314_{(10} \rightarrow 1{,}5263_{(10}$$

Por lo tanto, el número $1{,}526264_{(10}$ se aproxima por redondeo en este formato por el $1{,}5263_{(10}$.

El inconveniente del redondeo es que, a diferencia del truncamiento, comporta operaciones aritméticas.

2.1.5. Desbordamiento

Al hacer operaciones aritméticas con números en un formato determinado, nos podemos encontrar con que el resultado esté fuera del rango de representación. Es lo que se conoce como **desbordamiento**.

El **desbordamiento** aparece cuando el resultado de una operación supera el rango de representación.

Terminología

En inglés, el término *desbordamiento* recibe el nombre de *overflow* .

En un formato de 6 bits, la operación de suma siguiente produce desbordamiento, porque el resultado no cabe en 6 bits:

	1		1	1				← acarreo
		1	0	1	1	1	0	(2
+		1	0	0	1	0	1	(2
	1	**0**	**1**	**0**	**0**	**1**	**1**	(2 ← resultado
	↑ desbordamiento							

El desbordamiento puede aparecer en todos los sistemas de representación numérica, pero se manifiesta de maneras diferentes.

Hay un tipo especial de desbordamiento que recibe el nombre de **desbordamiento a cero** y que aparece cuando un número de magnitud menor que la precisión del formato, pero diferente de cero, finalmente se acaba representando, debido al error de representación, como cero. Esta situación es relevante porque operaciones como la división que, a priori, no tendrían que presentar ninguna dificultad pueden volverse irresolubles.

Terminología

En inglés, el desbordamiento a cero recibe el nombre de *underflow* .

Actividades

14. Determinad el rango y la precisión de los formatos de coma fija sin signo $x_1x_0{,}x_{-1}x_{-2}x_{-3}$ y $x_2x_1x_0{,}x_{-1}x_{-2}$ donde x_i es un dígito decimal.

15. Determinad si el número 925,4 se puede representar en los formatos indicados en la actividad 14.

16. Representad en el formato de coma fija y sin signo $x_1x_0{,}x_{-1}x_{-2}$, donde x_i es un dígito decimal, los números siguientes:
a) $10_{(10}$
b) $10{,}02_{(10}$
c) $03{,}1_{(10}$
d) $03{,}2_{(10}$

17. Determinad la cantidad de números que se pueden representar en el formato $x_2x_1x_0{,}x_{-1}x_{-2}x_{-3}$, donde x_i es un dígito decimal.

18. Calculad el error de representación que se comete cuando representamos en el formato $x_2x_1x_0{,}x_{-1}x_{-2}$, donde x_i es un dígito decimal, los números siguientes:
a) $223{,}45_{(10}$
b) $45{,}89_{(10}$
c) $55{,}6356_{(10}$
d) $23{,}56_{(10}$

19. Escoged el formato hexadecimal que use el mínimo número de dígitos y que permita representar el número $16{,}25_{(10}$ de manera exacta. ¿Cuál es el rango y la precisión del formato?

20. ¿Cuál es el número más pequeño que hay que sumar a $8341_{(10}$ para que se produzca desbordamiento en una representación decimal (base 10) de cuatro dígitos?

2.2. Números naturales

Los números naturales son los números sin parte fraccionaria y sin signo. Es decir, son los miembros de la sucesión: 0, 1, 2, 3, 4, 5, 6...

Dentro de los computadores los números naturales se representan en base 2, la precisión es 1 (puesto que no hay bits fraccionarios) y el rango depende del número de bits disponibles en el formato.

El **rango** de representación de los números naturales en un formato de n bits es, en decimal, $[0,2^n - 1]$ y su **precisión** es 1.

El rango de representación se puede ampliar si aumentamos el número de bits de la representación. La ampliación del número de bits de un formato de representación recibe el nombre de **extensión**.

La **extensión** de los números naturales representados en un formato de n bits a un formato de m bits, con $m > n$, se consigue añadiendo, a la izquierda de la codificación, los ceros necesarios hasta completar los m bits del formato nuevo.

La representación del número natural $10_{(10}$ en un formato de 5 bits es $01010_{(2}$. La extensión de esta codificación a un formato de 8 bits se consigue añadiendo ceros a la izquierda hasta completar los 8 bits del formato nuevo, con lo cual la nueva codificación será $00001010_{(2}$.

Cambio de base del número $10_{(10}$ a base 2

Siguiendo el método basado en el teorema de la división entera:

$10 = 5 \cdot 2 + 0$
$5 = 2 \cdot 2 + 1$
$2 = 1 \cdot 2 + 0$
$1 = 0 \cdot 2 + 1$

$10_{(10} = 1010_{(2}$

Las operaciones de suma y de resta siguen las pautas expuestas anteriormente. Si se produce un acarreo en la última etapa de suma, hay desbordamiento:

```
  1 1                    ← acarreo
    1 1 1 0 1 0 (2
+   0 1 0 1 0 1 (2
-----------------
  1 0 0 1 1 1 1 (2   ← resultado
  ↑
desbordamiento
```

Consultad la suma y la resta en los sistemas posicionales en los subapartados 1.6 y 1.7.

La suma de dos números naturales de n bits da lugar a un resultado que como máximo requiere $n + 1$ bits para su representación.

El **desbordamiento** en la suma de dos números naturales se produce cuando tenemos un acarreo en la última etapa de suma. La operación de resta de números naturales no puede dar lugar a desbordamiento.

Atención

La resta de dos naturales no puede producir desbordamiento porque restamos la magnitud pequeña de la grande. Restar la magnitud grande de la pequeña no es una operación válida dentro de los naturales, porque el resultado sería un número con signo.

Las operaciones de multiplicación y división entera por potencias de la base de numeración se ajustan a los procedimientos ya descritos.

Consultad multiplicación y división por potencias de la base en el subapartado 1.8.

La división entera por una potencia de la base no produce desbordamiento, porque el resultado son dos números naturales (cociente y resto) más pequeños que el dividendo. En la multiplicación hay desbordamiento si el resultado supera el rango del formato.

La división de dos naturales

La operación de división sobre números naturales debe ser la división entera dado que los números naturales no tienen parte fraccionaria.

2.3. Números enteros

Los enteros son los números con signo y sin parte fraccionaria, incluyendo el cero: ... –3, –2, –1, 0, +1, +2, +3 ... Se diferencian de los naturales por la presencia de un signo que indica si la magnitud es positiva o negativa. Este signo se puede incorporar a la codificación de los números dentro de los computadores de varias maneras. En los apartados siguientes describimos las más utilizadas en los computadores: signo y magnitud, y complemento a 2.

2.3.1. Representación de enteros en signo y magnitud en base 2

Signo y magnitud es, probablemente, la forma más intuitiva de representar números con signo. En **signo y magnitud**, el bit más significativo (MSB) almacena el signo y el resto codifica la magnitud. Un 1 en el dígito más significativo indica signo negativo, mientras que un 0 indica signo positivo.

MSB y LSB

MSB es la abreviación de *most significant bit*, es decir, el bit más significativo de la representación, que se corresponde con el dígito del extremo izquierdo. LSB es la abreviación de *least significant bit*, es decir, el bit menos significativo de la representación, que se corresponde con el dígito del extremo derecho.

Así, si la cadena de bits 101001 es un número en signo y magnitud, sabremos que es un número negativo, porque el bit más significativo es 1; y que la magnitud es $01001_{(2}$, que en base 10 es el $9_{(10}$. Esta cadena de bits codifica el $-9_{(10}$.

Un número codificado en **signo y magnitud** con n bits viene dado por la cadena de bits $x_{n-1}x_{n-2}\cdots x_1x_0$, donde x_{n-1} codifica el signo y $x_{n-2}\cdots x_1x_0$, la magnitud. El signo es positivo si x_{n-1} es 0, y negativo si es 1.

A lo largo del texto usaremos la notación $X_{(SM2}$ en identificar un número codificado en signo y magnitud en base 2.

La codificación en signo y magnitud también se usa para números fraccionarios con signo, tal y como se explica más adelante.

Representación en signo y magnitud

Para representar el $-12_{(10}$ en signo y magnitud, 6 dígitos y base 2, tenemos que pasar la magnitud $12_{(10}$ a base 2 ($12_{(10} = 1100_{(2}$) y poner el bit más significativo de la representación (el bit de más a la izquierda) a 1. La representación con 6 bits es, pues, $101100_{(SM2}$.

El $+12_{(10}$ se representa en el mismo formato como $001100_{(SM2}$. Sólo cambia el bit más significativo, porque la magnitud es la misma.

Cambio de base del número $12_{(10}$ a base 2

Aplicando la división entera:

$12 = 6 \cdot 2 + 0$
$6 = 3 \cdot 2 + 0$
$3 = 1 \cdot 2 + 1$
$1 = 0 \cdot 2 + 1$

$12_{(10} = 1100_{(2}$

Rango de representación en signo y magnitud y base 2

El formato de signo y magnitud es simétrico, es decir, se pueden representar tantos valores positivos como negativos. Con 4 bits y signo y magnitud tendremos 1 bit (el más significativo) para el signo y 3 para la magnitud:

- Valores posibles de signo:

$$0 \rightarrow +$$
$$1 \rightarrow -$$

- Valores posibles para la magnitud:

$000_{(2} = 0_{(10}$	$100_{(2} = 4_{(10}$
$001_{(2} = 1_{(10}$	$101_{(2} = 5_{(10}$
$010_{(2} = 2_{(10}$	$110_{(2} = 6_{(10}$
$011_{(2} = 3_{(10}$	$111_{(2} = 7_{(10}$

Combinando signo y magnitud podemos representar los valores enteros entre –7 y +7. Por lo tanto, el rango de representación es [–7 , +7].

Ventajas del formato de signo y magnitud

El formato de signo y magnitud tiene ventajas a la hora de hacer multiplicaciones: se operan por separado las magnitudes y los signos y, posteriormente, se juntan los resultados obtenidos de manera independiente.

En general, en signo y magnitud en base 2, el **rango** de enteros representable con n bits es, en decimal,

$$[-(2^{n-1} - 1), 2^{n-1} - 1]$$

Si aplicamos esta expresión al caso de 4 bits, tenemos:

$$[-(2^{4-1} - 1), 2^{4-1} - 1] = [-(2^3 - 1), 2^3 - 1] = [-7, +7]$$

que es el rango al que habíamos llegado de manera experimental.

La precisión en la codificación de enteros en signo y magnitud es 1, porque se pueden codificar todos los enteros del rango de representación.

Si fuera necesario ampliar el rango de representación tendríamos que hacer una extensión del formato de signo y magnitud.

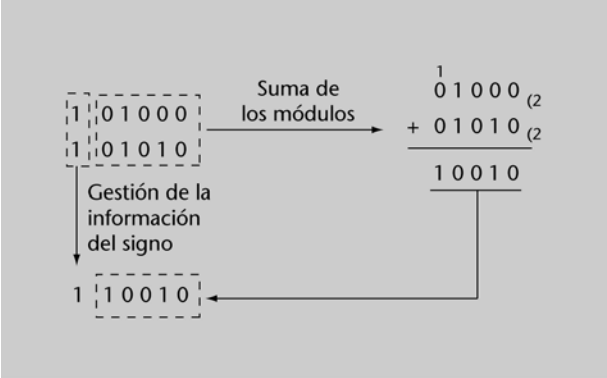
La **extensión** de n a m bits, con $m > n$, de los números en signo y magnitud se consigue añadiendo, a la izquierda de la magnitud, los ceros necesarios para completar los m bits, manteniendo el bit del extremo izquierdo para la codificación del signo.

Por consiguiente, el entero negativo $11010_{(SM2}$ codificado en signo y magnitud y 5 bits se puede extender a un formato de 8 bits añadiendo ceros a la derecha del signo, de forma que la codificación de este mismo número en el nuevo formato sería $1\mathbf{000}1010_{(SM2}$. La extensión de números positivos se hace del mismo modo. La extensión a 8 bits de la codificación en signo y magnitud del entero positivo $01010_{(SM2}$ nos lleva al $0\mathbf{000}1010_{(SM2}$.

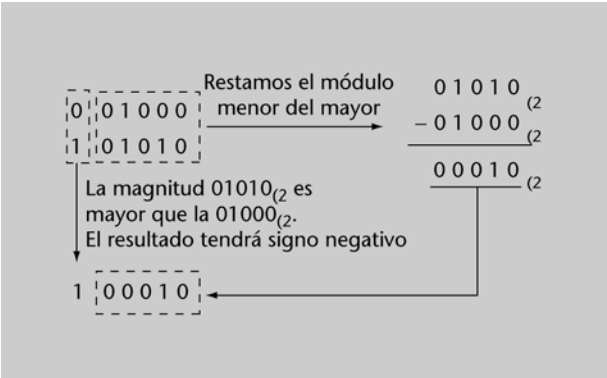
2.3.2. Suma y resta en signo y magnitud

La suma de dos números positivos o dos negativos en signo y magnitud es sencilla. Tenemos que hacer la suma de las magnitudes y dar al resultado el signo de los operandos. La suma de las magnitudes puede producir desbordamiento.

La suma de los números $101000_{(SM2}$ y $101010_{(SM2}$ codificados en signo y magnitud y 6 bits, es la siguiente:



La suma de un positivo y un negativo es más compleja: hay que analizar las magnitudes para saber cuál es la mayor, restar la magnitud pequeña de la grande y aplicar al resultado el signo de la magnitud mayor. El procedimiento de suma de los números $001000_{(SM2}$ y $101010_{(SM2}$ codificados en signo y magnitud y 6 bits, es el siguiente:



La suma de dos números de mismo signo y la resta de números de signo contrario puede producir desbordamiento.

En signo y magnitud, hay **desbordamiento** en la suma de dos números del mismo signo o en la resta de números de signo contrario cuando aparece un acarreo en la última etapa de suma o resta de las magnitudes.

Ni la suma de un positivo y un negativo, ni la resta de números del mismo signo pueden producir desbordamiento.

En la suma de $101010_{(SM2}$ y el $111010_{(SM2}$, en signo y magnitud y 6 bits, examinamos, en primer lugar, los signos. Son dos números negativos, puesto que el bit de mayor peso de ambos es 1. Por lo tanto, procederemos a la suma de las magnitudes:

```
  1 | 1     1          ← acarreo
    |
    | 0 1 0 1 0 (2
+   | 1 1 0 1 0 (2
-------------------
  1 | 0 0 1 0 0 (2     ← resultado
  ↑ |
desbordamiento
```

La suma de las magnitudes produce desbordamiento, puesto que tenemos un acarreo en la última etapa. Por lo tanto, el resultado no cabe en el formato definido y no se puede representar.

Los inconvenientes principales del sistema de signo y magnitud son la complejidad de las operaciones de suma y resta y la existencia de dos representaciones para el 0: un "0 positivo", cuando la magnitud es 0 y el signo también; y un "0 negativo", cuando la magnitud es 0 y el signo 1.

2.3.3. Representación en complemento a 2

El **complemento a 2**, abreviado habitualmente por **Ca2** o **C2**, es un sistema de representación de números con signo en base 2. Actualmente, el Ca2 es el sistema más empleado para codificar números enteros en los computadores porque presenta dos ventajas: una codificación única para el cero, y simplicidad en las operaciones de suma y resta.

Los **números positivos en Ca2** se codifican de la misma forma que en signo y magnitud: el bit del extremo izquierdo es 0, para indicar signo positivo, y el resto contiene la magnitud.

La codificación de un **número negativo** $-X$ en Ca2 es el resultado en binario de la operación $2^n - |X|$, donde $|X|$ es el valor absoluto de X.

Inconvenientes del formato de Ca2

Sin que afecte a la eficiencia de los computadores, los valores de las magnitudes negativas codificadas en Ca2 son más difíciles de reconocer para nosotros.

A lo largo del texto utilizaremos la notación $X_{(Ca2}$ para identificar un número codificado en complemento a 2.

Codificación de números negativos en Ca2

Para hallar la codificación en Ca2 y 6 bits del valor $-11010_{(2}$, hacemos la operación siguiente:

$$2^6 - |X| = 1000000_{(2} - 11010_{(2} = 100110_{(Ca2}$$

$$\begin{array}{rcccccccl}
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & {\scriptstyle (2} \\
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & \\
- & & & 1 & 1 & 0 & 1 & 0 & {\scriptstyle (2} \\
\hline
 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & {\scriptstyle (Ca2}
\end{array}$$

Así pues, la codificación en Ca2 y 6 bits del valor $-11010_{(2}$ es $100110_{(Ca2}$.

La codificación del valor $+11010_{(2}$ en Ca2 coincide con la codificación en signo y magnitud. Tendremos un 0 para el signo y a continuación 5 bits con la magnitud: el valor $+11010_{(2}$ se codifica en Ca2 como $011010_{(Ca2}$.

Para hallar la codificación en Ca2 y 8 bits del valor $-11010_{(2}$, hacemos la operación siguiente:

$$2^8 - |X| = 100000000_{(2} - 11010_{(2} = 11100110_{(Ca2}$$

$$\begin{array}{rcccccccccl}
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & {\scriptstyle (2} \\
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & \\
- & & & & & 1 & 1 & 0 & 1 & 0 & {\scriptstyle (2} \\
\hline
 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & {\scriptstyle (Ca2}
\end{array}$$

Así pues, la codificación en Ca2 y 8 bits del valor $-11010_{(2}$ es $11100110_{(Ca2}$.

También se puede obtener la codificación en Ca2 de una magnitud negativa, haciendo un cambio de signo en la codificación de la magnitud positiva. Consultad, más adelante, el subapartado 2.3.4.

Rango de representación en Ca2

La tabla siguiente muestra los enteros representables con 4 bits en signo y magnitud y en complemento a 2 y su correspondencia:

Decimal	Signo y magnitud	Ca2
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
0	0000 1000	0000
–1	1001	1111
–2	1010	1110

Tabla

En la tabla vemos que los positivos se codifican igual en Ca2 y signo y magnitud. El rango de los positivos es el mismo en los dos sistemas. En cambio, en Ca2 tenemos un negativo más que en signo y magnitud. Esto es debido a que en Ca2 hay una representación única del cero, mientras que en signo y magnitud tiene dos.

Decimal	Signo y magnitud	Ca2
–3	1011	1101
–4	1100	1100
–5	1101	1011
–6	1110	1010
–7	1111	1001
–8	no es representable	1000

En Ca2 y 4 bits se puede representar desde el $-8_{(10}$ hasta el $+7_{(10}$.

En general, el **rango** de enteros representables con *n* bits en Ca2 es, en decimal:

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

Con 4 bits, el rango es: $[-2^{4-1}, 2^{4-1} - 1] = [-2^3, 2^3 - 1] = [-8, +7]$

En Ca2, para aumentar el número de bits con que se codifica un entero positivo se puede seguir el mismo procedimiento que en signo y magnitud. En cambio, la extensión para los enteros negativos es diferente. El $-10_{(10}$ en Ca2 y 5 bits es el $10110_{(Ca2}$, mientras que con 8 bits se codifica como $11110110_{(Ca2}$. La diferencia entre las codificaciones es que en la segunda se han añadido tres 1 a la izquierda.

Ejemplo

En Ca2, el $-10_{(10}$ se codifica con 5 bits, por el 10110:

$2^5 - |-10_{(10}| = 32_{(10} - 10_{(10} = 22_{(10} = 10110_{(Ca2}$

En Ca2, el $-10_{(10}$ se codifica con 8 bits, por el 11110110:

$2^8 - |-10_{(10}| = 256_{(10} - 10_{(10} = 246_{(10} = 11110110_{(Ca2}$

Fijémonos que en los dos casos los bits que se añaden coinciden con el valor del bit de mayor peso: ceros para los positivos y unos para los negativos.

En Ca2, **para extender** un formato de *n* bits a *m* bits, con $m > n$, se copia a la izquierda el bit de más peso las veces necesarias para completar los *m* bits. Este proceso recibe el nombre de **extensión del signo**.

En Ca2 el bit de mayor peso indica el signo

En Ca2, un 1 en el bit de mayor peso indica que el número es negativo, mientras que un 0 indica que es positivo.

2.3.4. Cambio de signo en complemento 2

Haremos un cambio de signo de un número en Ca2, si seguimos los pasos siguientes:

1) Hacer el complemento bit a bit de la codificación en Ca2.
2) Sumar 1 al bit menos significativo de la codificación.

Complemento bit a bit

Se entiende por *complemento bit a bit*, la sustitución de cada bit por su complementario.

En base 2, el complementario del 0 es el 1 y el del 1 es el 0.

Cambio de signo en complemento a 2

Para hacer el cambio de signo del valor numérico $11000110_{(2}$ (que si seguimos el procedimiento explicado en el apartado siguiente veríamos que se trata del $-58_{(10)}$), que está codificado en complemento a 2 y 8 dígitos, hacemos la operación siguiente:

	1	1	0	0	0	1	1	0	(Ca2	← valor numérico inicial
							1			← acarreo
	0	0	1	1	1	0	0	1	(Ca2	← complemento bit a bit de la expresión inicial
+								1		← sumamos 1 al bit menos significativo del formato
	0	0	1	1	1	0	1	0	(Ca2	

De esta operación resulta el $00111010_{(Ca2}$, que codifica la misma magnitud, pero con signo positivo.

El cambio de signo de un número en Ca2 también se puede conseguir si examinamos los bits de derecha a izquierda y:

1) Mantenemos los mismos bits hasta encontrar el primer 1 (incluyéndolo).
2) Hacemos el complemento bit a bit del resto.

Cambio de signo en complemento a 2

Para hacer el cambio de signo del 11000110 que está en Ca2 y 8 bits, lo examinamos de derecha a izquierda, haciendo el complementando bit a bit después del primer 1:

```
11000110
        ^ se mantienen los bits hasta aquí (primer 1 que encontramos incluido)
        10
       ^ se complementan los bits a partir de este punto
001110
```

De esta operación resulta el 00111010, que codifica la misma magnitud pero con signo positivo.

El cambio de signo del 00011110, que está en Ca2 y 8 bits, se obtiene siguiendo el mismo procedimiento:

```
00011110
        ^ se mantienen los bits hasta aquí (primer 1 que encontramos incluido)
        10
       ^ se complementan los bits a partir de este punto
111000
```

El resultado es 11100010, que codifica la misma magnitud pero con signo negativo.

El cambio de signo se puede usar como alternativa a la operación $2^n - |X|$ para encontrar la codificación en Ca2 de una magnitud negativa.

La codificación en Ca2 de una magnitud negativa se puede obtener aplicando un cambio de signo a la codificación en Ca2 de la magnitud positiva.

La operación es reversible. Si aplicamos un cambio de signo a la codificación en Ca2 de una magnitud negativa, encontraremos la codificación de la positiva.

2.3.5. Magnitud de los números en complemento a 2

Como en signo y magnitud, la magnitud decimal de un número positivo codificado en Ca2 se puede conocer aplicando el TFN:

$$0101_{(2} = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = +5_{(10}$$

En Ca2, la cadena de bits 0101 codifica el valor $+5_{(10}$.

Para encontrar la magnitud decimal de un número negativo en Ca2, disponemos de dos alternativas:

1) Aplicando el TFN, como en el caso de los positivos, pero considerando que el bit de mayor peso es negativo.

Magnitud decimal de un valor negativo codificado en Ca2 aplicando el TFN

Aplicamos el TFN para encontrar la magnitud decimal que codifica la cadena de bits 10001010 en Ca2, considerando que el primer bit es negativo:

$$10001010_{(Ca2} = -1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = \\ = -128 + 10 = -118_{(10}$$

El 10001010 en Ca2 codifica el valor decimal –118.

2) Aplicar un cambio de signo a la representación en Ca2 del valor negativo y encontrar la magnitud positiva.

Magnitud decimal de un valor negativo en Ca2 por cambio de signo

Para conocer la magnitud decimal que codifica el 10001010 en Ca2:

1) Aplicamos un cambio de signo:

	1	0	0	0	1	0	1	0 $_{(Ca2}$	← Valor numérico inicial
							1		← Acarreo
	0	1	1	1	0	1	0	1 $_{(Ca2}$	← Complemento bit a bit de la expresión inicial
+								1	← Sumamos 1 al bit menos significativo del formato
	0	1	1	1	0	1	1	0 $_{(Ca2}$	

2) Aplicamos el TFN al resultado:

$$01110110_{(Ca2} = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = +118_{(10}$$

Por lo tanto, la cadena de bits 10001010 codifica en Ca2 el entero decimal –118.

2.3.6. Suma en complemento a 2

El mecanismo de suma en Ca2 es el mismo que el utilizado en cualquier otro sistema posicional. Tenemos que saber reconocer, sin embargo, cuándo se produce desbordamiento.

Consultad la suma y la resta en los sistemas posicionales en los subapartados 1.6 y 1.7.

Suma de dos valores positivos en Ca2

Consideremos la suma de dos números positivos en Ca2 y 6 bits siguiente:

Podemos encontrar la correspondencia entre los números en Ca2 y los valores decimales, aplicando cualquiera de los dos métodos expuestos en el apartado 2.3.5.

$$\begin{array}{rcccccl} & 0 & 0 & 1 & 0 & 1 & 0 \;_{(Ca2} \\ + & 0 & 0 & 0 & 1 & 0 & 1 \;_{(Ca2} \\ \hline & 0 & 0 & 1 & 1 & 1 & 1 \;_{(Ca2} \end{array} \quad \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \quad \begin{array}{rrl} & +10 & _{(10} \\ + & +5 & _{(10} \\ \hline & +15 & _{(10} \end{array}$$

(resultado correcto)

Sabemos que el resultado es correcto porque hemos hecho la suma de dos positivos y obtenemos una magnitud positiva. Cuando el resultado supera el rango de representación, la suma de dos positivos genera una magnitud negativa, como en el caso siguiente:

$$\begin{array}{rcccccl} & 1 & 1 & 1 & 1 & & \leftarrow \text{acarreo} \\ & 0 & 1 & 0 & 1 & 1 & 0 \;_{(Ca2} \\ + & 0 & 0 & 1 & 1 & 1 & 1 \;_{(Ca2} \\ \hline & 1 & 0 & 0 & 1 & 0 & 1 \;_{(Ca2} \end{array} \quad \begin{array}{c} \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \quad \begin{array}{rrl} & +22 & _{(10} \\ + & +15 & _{(10} \\ \hline & -27 & _{(10} \end{array}$$

(desbordamiento)

En Ca2 y 6 bits, el rango es $[-2^{6-1}, +2^{6-1} - 1] = [-32, +31]$. El resultado de esta suma tendría que ser $+37_{(10}$ ($22_{(10}+15_{(10} = +37_{(10}$), que queda fuera del rango.

Hay **desbordamiento** en la suma de dos números positivos codificados en Ca2 cuando el resultado es negativo.

Suma de dos valores negativos en Ca2

De manera análoga, el resultado de la suma de dos negativos en Ca2 es correcto cuando se obtiene una magnitud negativa, y erróneo cuando se obtiene una positiva.

Consideremos la suma de dos números negativos en Ca2 y 6 bits siguiente:

Podemos encontrar la correspondencia entre los números en Ca2 y los valores decimales aplicando cualquiera de los dos métodos expuestos en el apartado 2.3.5.

$$\begin{array}{rcccccl} 1 & 1 & & & & & \leftarrow \text{acarreo} \\ & 1 & 1 & 1 & 0 & 1 & 0 \;_{(Ca2} \\ + & 1 & 1 & 0 & 1 & 0 & 1 \;_{(Ca2} \\ \hline 1 & 1 & 0 & 1 & 1 & 1 & 1 \;_{(Ca2} \end{array} \quad \begin{array}{c} \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \quad \begin{array}{rrl} & -6 & _{(10} \\ + & -11 & _{(10} \\ \hline & -17 & _{(10} \end{array}$$

(resultado correcto)

El resultado es correcto, aunque haya un acarreo en la última etapa de suma, porque se obtiene una magnitud negativa en la suma de dos negativos. En cambio, el resultado de la suma siguiente es erróneo:

	1		1	1	1			← acarreo				
		1	0	0	1	1	0	(Ca2	→		−26	(10
+		1	0	1	1	1	1	(Ca2	→	+	−17	(10
	1	0	1	0	1	0	1	(Ca2	→		+21	(10
								(desbordamiento)				

Como en el caso anterior, el acarreo en la última etapa se tiene que despreciar, pero sumando dos negativos hemos obtenido un positivo. Por lo tanto, hay desbordamiento.

> Hay **desbordamiento** en la suma de dos números negativos codificados en Ca2 cuando el resultado es positivo. El acarreo de la última etapa no indica desbordamiento y se tiene que despreciar en todos los casos.

Suma de un valor positivo y un valor negativo en Ca2

El resultado de la suma de un positivo y un negativo en Ca2 siempre es correcto.

Podemos encontrar la correspondencia entre los números en Ca2 y los valores decimales, aplicando cualquiera de los dos métodos expuestos en el apartado 2.3.5.

1	1							← acarreo																
	1	1	1	0	1	0	(Ca2	→	−6	(10		1	0	0	1	1	0	(Ca2	→	−26	(10
+	0	1	0	1	0	1	(Ca2	→	+ +21	(10	+	0	1	0	0	0	1	(Ca2	→	+ +17	(10
1	0	0	1	1	1	1	(Ca2	→	+15	(10		1	1	0	1	1	1	(Ca2	→	−9	(10
							(resultado correcto)											(resultado correcto)			

La aparición de un acarreo en la última etapa, como la operación de la izquierda, no indica desbordamiento y se tiene que despreciar.

> La suma de un número positivo y un número negativo codificados en Ca2 no puede producir desbordamiento. El acarreo que se puede producir en la última etapa no indica desbordamiento y se tiene que despreciar en todos los casos.

2.3.7. Resta en complemento a 2

El procedimiento que se sigue para hacer la resta en Ca2 es aplicar un cambio de signo al sustraendo (operando que queremos restar) y hacer una operación de suma. Sumamos el minuendo con el sustraendo al cual hemos cambiado el signo.

La operación de resta en Ca2 se reduce a una operación de suma una vez se ha cambiado el signo del sustraendo.

La resta $011010_{(Ca2} - 001011_{(Ca2}$ ($26_{(10} - 11_{(10}$) nos servirá de ejemplo para ilustrar el procedimiento:

1) Aplicamos un cambio de signo al sustraendo:

a) Hacemos el complemento bit a bit de 001011, con el que se obtiene 110100.
b) Sumamos 1 al bit menos significativo:

$$\begin{array}{rr} & 1\ 1\ 0\ 1\ 0\ 0\ {}_{(Ca2} \\ + & 1\ {}_{(Ca2} \\ \hline & 1\ 1\ 0\ 1\ 0\ 1\ {}_{(Ca2} \end{array}$$

2) Hacemos la operación de suma:

Podemos encontrar la correspondencia entre los números en Ca2 y los valores decimales, aplicando cualquiera de los dos métodos expuestos en el apartado 2.3.5.

$$\begin{array}{rrlcr} 1 & 1 & \leftarrow \text{acarreo} & & \\ & 0\ 1\ 1\ 0\ 1\ 0\ {}_{(Ca2} & & \rightarrow & +26\ {}_{(10} \\ + & 1\ 1\ 0\ 1\ 0\ 1\ {}_{(Ca2} & & \rightarrow & +\ -11\ {}_{(10} \\ \hline 1 & 0\ 0\ 1\ 1\ 1\ 1\ {}_{(Ca2} & & \rightarrow & +15\ {}_{(10} \\ & & \text{(resultado correcto)} & & \end{array}$$

El resultado es correcto. El acarreo de la última etapa se tiene que despreciar y no se produce desbordamiento. Recordamos que la suma de un número positivo y un número negativo no puede dar lugar a desbordamiento.

2.3.8. Multiplicación por 2^k de números en complemento a 2

Como hemos visto, multiplicar por 2^k en sistemas de numeración posicionales de base 2 equivale a desplazar la coma fraccionaria *k* posiciones a la derecha. En el caso de los enteros, este efecto se consigue añadiendo a la derecha *k* ceros:

Ved la multiplicación y la división por potencias de la base de numeración en el subapartado 1.8 de este módulo.

$$000101_{(Ca2} \cdot 2^2 = 010100_{(Ca2} \qquad \text{(en decimal, } +5_{(10} \cdot 2^2 = +20_{(10})$$

El procedimiento también es válido para números negativos en Ca2:

$$111011_{(Ca2} \cdot 2^2 = 101100_{(Ca2} \qquad \text{(en decimal, } -8_{(10} \cdot 2^2 = -32_{(10}\ )$$

Por aplicación del TFN

$000101_{(2} = 2^2 + 2^0 = 5_{(10}$
$010100_{(2} = 2^4 + 2^2 = 20_{(10}$
$111011_{(2} = -2^5 + 2^4 + 2^3 + 2^1 + 2^0 = -5_{(10}$
$101100_{(2} = -2^5 + 2^3 + 2^2 = -20_{(10}$

El resultado de **multiplicar por 2^k un número en Ca2** se consigue añadiendo *k* ceros a la derecha.

Una vez hemos fijado un formato de n bits, añadir k ceros a la derecha nos obliga a perder k bits de la izquierda, lo cual puede producir desbordamiento. En Ca2 y 6 bits, las operaciones siguientes producen desbordamiento:

$$000101_{(Ca2} \cdot 2^4 = 010000_{(Ca2} \quad \text{(en decimal, } +5_{(10} \cdot 2^4 = +16_{(10})$$
$$111000_{(Ca2} \cdot 2^4 = 000000_{(Ca2} \quad \text{(en decimal, } -8_{(10} \cdot 2^4 = 0_{(10}\text{)}$$

Se produce **desbordamiento** al multiplicar un número en Ca2 por 2^k cuando cambia el bit de signo o bien si se pierde uno o más bits significativos. Los bits significativos son, para los positivos los 1 y para los negativos los 0.

Actividades

21. Convertid los valores decimales siguientes a binarios en los sistemas de representación de signo y magnitud y complemento a 2, con un formato entero de 8 bits:
a) 53
b) –25
c) 93
d) –1
e) –127
f) –64

22. Si tenemos los números binarios 00110110, 11011010, 01110110, 11111111 y 11100100, ¿cuáles son los equivalentes decimales considerando que son valores binarios representados en signo y magnitud?

23. Repetid el ejercicio anterior considerando que las cadenas de bits son números en complemento a 2.

24. Si tenemos las cadenas de bits siguientes A = 1100100111, B = 1000011101 y C = 0101011011, haced las operaciones que proponemos a continuación considerando que son números binarios en formato de signo y magnitud: $A + B$, $A - B$, $A + C$, $A - C$, $B - C$, $B + C$.

25. Repetid la actividad anterior considerando que las cadenas representan números en complemento a 2.

26. Si tenemos la cadena de bits 10110101, haced las conversiones siguientes:
a) Considerando que representa un número en Ca2, representad el mismo número en signo y magnitud y 16 bits.
b) Considerando que representa un número en signo y magnitud, representad el mismo número en Ca2 y 16 bits.

2.4. Números fraccionarios

Los números fraccionarios son los que tienen una parte más pequeña que la unidad, como por ejemplo el $0{,}03_{(10}$ o el $15{,}27_{(10}$. La representación de los números fraccionarios dentro de los computadores se suele hacer destinando un número fijo de bits, del total de bits del formato, a la representación de la parte

Número decimal

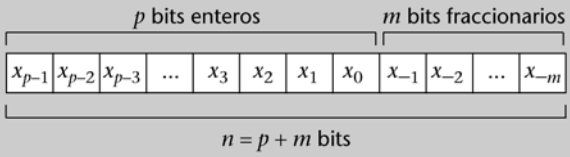
Usamos la expresión *número decimal* para designar un número en base 10, no un número con parte fraccionaria.

fraccionaria. Este tipo de representación recibe el nombre genérico de **representación de coma fija**.

Representación binaria de coma fija

Las representaciones de coma fija no almacenan la posición de la coma de manera explícita. Es en la definición del formato donde se especifica la posición de la coma, y se asume que siempre es la misma.

En un formato de representación de coma fija de n bits ($n = p + m$), donde m bits son fraccionarios, los números representados son de la forma:

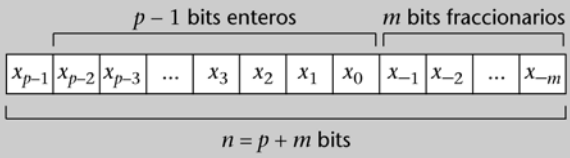


La magnitud decimal del número fraccionario representado es $X_{(10} = \sum_{i=p-1}^{-m} x_i \cdot 2^i$, donde x_i es el bit de la posición i-ésima.

Signo y magnitud en coma fija

Las magnitudes fraccionarias también pueden llevar asociado un signo. En coma fija, lo más habitual es trabajar con una representación de signo y magnitud.

En un formato de coma fija de n bits ($n = p + m$), donde m bits son fraccionarios, y en signo y magnitud, los números son de la forma:



donde x_{p-1} es el bit de signo y el resto de los bits codifican la magnitud.

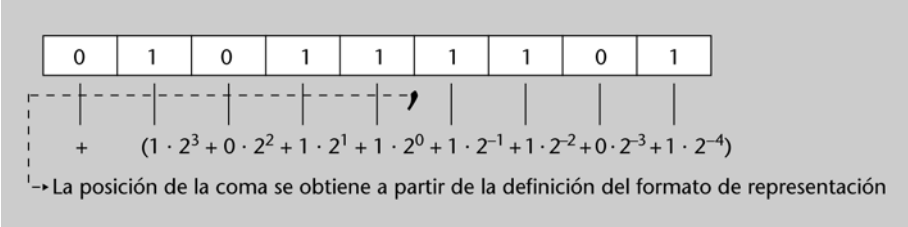
Otras maneras de representar números

Hay otras maneras de representar números fraccionarios con signo. Por ejemplo, se podría emplear la representación de complemento a la base (Ca2 en binario), pero no es habitual.

Para conocer en decimal el valor numérico representado por la codificación 010111101 que está en signo y magnitud en un formato de 9 bits (n = 9), de los cuales 4 son fraccionarios (m = 4), tenemos que aplicar el TFN:

Formato en coma fija

Esta forma de representar los números fraccionarios es una extensión directa de la representación en signo y magnitud de números enteros. Por lo tanto, presenta las mismas ventajas e inconvenientes. Así, por ejemplo, el cero tiene dos representaciones, una con signo negativo y otra con signo positivo.



Por lo tanto, el número codificado es $+1011{,}1101_{(2} = 11{,}8125_{(10}$.

Magnitud decimal de un número codificado en coma fija y signo y magnitud

Para encontrar la magnitud decimal que representa la codificación 10010010 que está en un formato de 8 bits, 4 de los cuales son fraccionarios y en signo y magnitud, haremos las operaciones siguientes:

1) Separamos el bit de signo que es 1 (bit del extremo izquierdo) y que indica signo negativo. El resto de bits 0010010 codifica la magnitud.

2) El valor decimal de la magnitud lo podemos conocer aplicando el TFN:

$$0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} = 2 + 0{,}125 = 1{,}125_{(10}$$

Por lo tanto, el número representado en decimal es $-1{,}125_{(10}$.

Codificación de un valor decimal en coma fija y signo y magnitud

Para codificar el número $-14{,}75_{(10}$, en un formato de coma fija de 8 bits donde 3 son fraccionarios y signo y magnitud haremos las operaciones siguientes:

1) Cambiar a base 2 el número $14{,}75_{(10}$, siguiendo el método basado en el teorema de la división entera.

Ved el método basado en el teorema de la división entera en el subapartado 1.3.2 de este módulo.

a) Codificar en binario la parte entera ($14_{(10}$) en 4 bits (puesto que de los 8 bits, 3 son fraccionarios y 1 codifica el signo; restan 4 por la parte entera), aplicando el algoritmo de divisiones sucesivas:

$$\begin{aligned} 14 &= 7 \cdot 2 + 0 \\ 7 &= 3 \cdot 2 + 1 \\ 3 &= 1 \cdot 2 + 1 \\ 1 &= 0 \cdot 2 + 1 \end{aligned}$$

Con el que obtenemos que $14_{(10} = 1110_{(2}$.

b) Codificar en binario la parte fraccionaria en 3 bits:

$$\begin{aligned} 0{,}75 \cdot 2 &= 1{,}50 = 1 + 0{,}5 \\ 0{,}50 \cdot 2 &= 1{,}0 = 1 + 0{,}0 \end{aligned}$$

Con el que obtenemos que $0{,}75_{(10} = 0{,}110_{(2}$

c) Juntar las partes entera y fraccionaria en el formato de 7 bits, 3 de los cuales son fraccionarios:

$$14{,}75_{(10} = 1110{,}110_{(2}$$

2) Añadir el bit de signo a la magnitud. El bit de signo es 1 puesto que el signo es negativo. La representación en un formato de coma fija de 8 bits donde 3 son fraccionarios y signo y magnitud del $-14{,}75_{(10}$ es el siguiente:

$$-14{,}75_{(10} = 11110{,}110_{(SM2}$$

Recordamos que la coma no se almacena, sino que una vez especificado el formato se conoce su posición. En realidad, un computador almacenaría el código 11110110 sin coma ni especificación de base, que están fijados en la definición del formato.

Cuando la parte fraccionaria excede el número de bits fraccionarios disponibles en el formato en signo y magnitud definido, el número no se podrá representar de manera exacta. Habrá que aplicar uno de los métodos de aproximación explicados: el truncamiento o el redondeo.

Ved las aproximaciones por truncamiento y redondeo en el subapartado 2.1.4 de este módulo.

A modo de ejemplo, intentemos representar el número $+8{,}9453125_{(10}$ en un formato de coma fija y signo y magnitud, con 8 bits de los cuales 3 son fraccionarios. Si hacemos el cambio de base, encontramos que $8{,}9453125_{(10} = 1000{,}1111001_{(2}$. Tendremos que usar uno de los métodos de aproximación, puesto que la parte fraccionaria no cabe en los 3 bits disponibles en el formato:

- Por **truncamiento**: Se trata, sencillamente, de despreciar los bits que no tienen cabida. El $1000{,}1111001_{(2}$ se aproximará por el $1000{,}111_{(2}$. Añadimos el bit de signo y la codificación final será 01000111. El error de representación que se comete en este caso es:

$$|1000{,}1111001_{(2} - 1000{,}111_{(2}| = 0{,}0001001_{(2} = 0{,}0703125_{(10}$$

- Por **redondeo**: Se suma la mitad de la precisión:

$$1000{,}1111001_{(2} + 0{,}0001_{(2} = 1001{,}0000001_{(2}$$

A continuación truncamos a 3 bits fraccionarios, de forma que el $1000{,}1111001_{(2}$ se aproximará por el $1001{,}000_{(2}$. Añadimos el bit de signo y la codificación final será 01001000. En este caso, el error de representación que se comete es menor:

$$|1000{,}1111001_{(2} - 1001{,}000_{(2}| = 0{,}0000111_{(2} = 0{,}0546875_{(10}$$

Ejemplo

Cambio de base 10 a base 2 del número $8{,}9453125_{(10}$:

1. Parte entera:

$8 = 4 \cdot 2 + 0$
$4 = 2 \cdot 2 + 0$
$2 = 1 \cdot 2 + 0$
$1 = 0 \cdot 2 + 1$

2. Parte fraccionaria:

$0{,}9453125 \cdot 2 = 0{,}890625 + 1$
$0{,}890625 \cdot 2 = 0{,}781250 + 1$
$0{,}781250 \cdot 2 = 0{,}5625 + 1$
$0{,}5625 \cdot 2 = 0{,}125 + 1$
$0{,}125 \cdot 2 = 0{,}25 + 0$
$0{,}25 \cdot 2 = 0{,}5 + 0$
$0{,}5 \cdot 2 = 0 + 1$

$8{,}9453125_{(10} = 1000{,}1111001_{(2}$

La precisión

La precisión es la distancia entre dos números representables consecutivos (ved el subapartado 2.1.2). Fácilmente, se puede comprobar que la precisión con 3 bits fraccionarios es $0{,}001_{(2}$ (distancia entre el 0,000 y el 0,001).

La mitad de este valor lo conseguimos desplazando la coma una posición a la izquierda (que equivale a dividir por 2 en base 2). Por lo tanto, la mitad de la precisión es $0{,}0001_{(2}$.

Rango y precisión en coma fija

La magnitud binaria más grande que podemos representar en un formato de coma fija es la que se obtiene poniendo todos los bits que representan la magnitud a 1. Si el bit de signo lo ponemos a cero, tendremos la representación de la mayor magnitud positiva que se puede representar. Si el bit de signo es 1, se tratará de la mayor magnitud negativa que se puede representar. Estos números, el mayor y el menor, delimitan el intervalo que contiene los números que se pueden representar, es decir, el rango.

El número mayor representable en signo y magnitud y un formato de coma fija de 9 bits con 3 fraccionarios, es el $011111{,}111_{(2} = +31{,}875_{(10}$; el menor es el $111111{,}111_{(2} = -31{,}875_{(10}$. Por lo tanto, el rango en decimal de este formato es:

$$[-31{,}875_{(10}, +31{,}875_{(10}]$$

En general, el número mayor que se puede representar con n bits de los cuales m son fraccionarios se puede calcular suponiendo que todos los bits valen 1 y aplicando el TFN:

$$1 \cdot 2^{n-m-2} + 1 \cdot 2^{n-m-3} + ... + 1 \cdot 2^{0} + 1 \cdot 2^{-1} + ... + 1 \cdot 2^{-m} = 2^{n-m-1} - 2^{-m}$$

donde hemos aplicado la propiedad siguiente:

$$\overset{k}{111\cdots 11} = 2^{k-1} + 2^{k-2} + \cdots 2^{1} + 2^{0} = \frac{2^{k} - 2^{0}}{2-1} = \frac{2^{k}-1}{1} = 2^{k} - 1.$$

El **rango** de una representación en signo y magnitud y un formato en coma fija de n bits, donde m son fraccionarios, es

$$\left[-2^{n-m-1} + 2^{-m}, +2^{n-m-1} - 2^{-m}\right].$$

Del mismo modo, el **rango** de una representación de números fraccionarios sin signo en un formato de coma fija de n bits, donde m son fraccionarios, es el siguiente:

$$\left[0, +2^{n-m} - 2^{-m}\right].$$

Ampliación del número de bits de un formato de coma fija

La ampliación de un formato en coma fija ha tener en cuenta los posibles cambios en la posición de la coma. De hecho, tan sólo hay que conocer cuántos bits se destinan a la ampliación de la parte fraccionaria y cuántos a la ampliación de la parte entera. Una ampliación de k bits de la parte fraccionaria comporta añadir k ceros a la derecha de la magnitud. Una ampliación de p bits de la parte entera se consigue si añadimos p ceros a la izquierda de la magnitud. Si trabajamos en signo y magnitud, tendremos que separar el signo de la magnitud para hacer los cambios y después añadirlo de nuevo al extremo izquierdo.

Ejemplo

Para ampliar en 3 bits la parte fraccionaria y en 2 bits la parte entera del número $111{,}001_{(SM2}$ que está en coma fija y signo y magnitud, añadiremos 3 ceros a la derecha de la magnitud y 2 ceros a la izquierda de la magnitud. La nueva codificación en el caso de signo y magnitud es 1**00**11,001**000**$_{(SM2}$, donde se marcan en negro los dígitos añadidos.

Esta ampliación, en caso de que el número $111{,}001_{(2}$ fuera una magnitud sin signo, daría lugar a la codificación **00**111,001**000**$_{(2}$.

La extensión o ampliación de k bits por la parte fraccionaria y p bits por la parte entera de un formato de coma fija, tanto en signo y magnitud como sin signo, se consigue si añadimos k ceros a la derecha de la magnitud y p ceros a la izquierda de la magnitud.

Precisión de un formato de coma fija

La precisión es la distancia más pequeña entre dos números representables consecutivos. Si trabajamos con una representación en coma fija de 3 bits donde 1 es fraccionario y signo y magnitud, los números que se pueden repre-

sentar son: $-1{,}1_{(2}$, $-1{,}0_{(2}$, $-0{,}1_{(2}$, $0{,}0_{(2}$, $+0{,}1_{(2}$, $+1{,}0_{(2}$ y $+1{,}1_{(2}$. Como podemos observar, todos ellos están separados por una distancia de $0{,}1_{(2}$. Por este motivo la precisión es $0{,}1_{(2}$.

En las representaciones de coma fija, la precisión viene dada por el bit menos significativo de la representación.

> La **precisión** de una representación en coma fija de n bits, donde m son fraccionarios, es 2^{-m}.

Suma y resta en coma fija

Las operaciones de suma y de resta en coma fija se hacen a partir del algoritmo habitual descrito en el apartado 1.6, como podemos ver en los ejemplos siguientes:

$$\begin{array}{rrrrrrrrl}
 & 1 & 1 & & 0 & 0 & & & \leftarrow \text{acarreo} \\
 & & 1 & , & 1 & 0 & 1 & {}_{(2} & \\
+ & & 0 & , & 1 & 0 & 0 & {}_{(2} & \\
\hline
 & 1 & 0 & , & 0 & 0 & 1 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
\qquad
\begin{array}{rrrrrrrrl}
 & 1 & , & 1 & 0 & 1 & {}_{(2} & & \\
 & 0 & & 0 & 0 & & & & \leftarrow \text{acarreo} \\
- & 0 & , & 1 & 0 & 0 & {}_{(2} & & \\
\hline
 & 1 & , & 0 & 0 & 1 & {}_{(2} & & \leftarrow \text{resultado}
\end{array}$$

Fijémonos en que un número fraccionario, como por ejemplo el $1{,}101_{(2}$, se puede escribir de la forma $1101_{(2} \cdot 2^{-3}$. Mediante este procedimiento podemos asociar un número entero, el $1101_{(2}$ en este caso, a un número fraccionario.

Si aplicamos esta transformación a los números $1{,}101_{(2}$ y $0{,}100_{(2}$ de la suma anterior, obtenemos: $1{,}101_{(2} = 1101_{(2} \cdot 2^{-3}$ y $0{,}100_{(2} = 0100_{(2} \cdot 2^{-3}$. La suma se puede hacer de la forma:

$$1101_{(2} \cdot 2^{-3} + 0100_{(2} \cdot 2^{-3} = (1101_{(2} + 0100_{(2}) \cdot 2^{-3} = 10001_{(2} \cdot 2^{-3}$$

Observemos que mediante este procedimiento hemos transformado una suma de números fraccionarios en una suma de números enteros.

> Con la transformación anterior, las operaciones entre números fraccionarios se pueden llevar a cabo a través de operaciones entre números enteros.

Fijémonos también en el hecho de que en la representación no aparece la ubicación de la coma, pero, dado que todos los números con los que trabajaremos tendrán la coma en la misma posición, no hay que saber la posición para operar.

Es decir, las operaciones para sumar los números $11{,}01_{(2}$ y $00{,}01_{(2}$ o los números $1{,}101_{(2}$ y $0{,}001_{(2}$ son idénticas. La primera es:

$$11{,}01_{(2} + 00{,}01_{(2} = 1101_{(2} \cdot 2^{-2} + 0001_{(2} \cdot 2^{-2} = (1101_{(2} + 0001_{(2}) \cdot 2^{-2}$$

y la segunda:

$$1{,}101_{(2} + 0{,}001_{(2} = 1101_{(2} \cdot 2^{-3} + 0001_{(2} \cdot 2^{-3} = (1101_{(2} + 0001_{(2}) \cdot 2^{-3}$$

En ambos casos la operación realizada finalmente es la suma de los dos números enteros $1101_{(2}$ y $0001_{(2}$.

A continuación sumamos los números $11{,}111_{(2}$ y $01{,}111_{(2}$, que están en un formato de coma fija de 5 bits, donde 3 son fraccionarios:

$$\begin{array}{rl|ccccc}
 & & & & & & \\
 & 1 & 1 & 1 & 1 & & & \leftarrow \text{acarreo} \\
 & & 1 & 1 & 1 & 1 & 1 & _{(2} \\
+ & & 0 & 1 & 1 & 1 & 1 & _{(2} \\
\hline
 & 1 & 0 & 1 & 1 & 1 & 0 & _{(2} & \leftarrow \text{resultado}
\end{array}$$

Para representar el resultado nos hace falta un dígito más de los que tenemos disponibles en el formato definido. Por eso, el resultado no es representable en el formato especificado. Se ha producido desbordamiento, que podemos reconocer de la misma forma que en la suma de números naturales.

El desbordamiento en la suma de números sin signo en coma fija se puede detectar de la misma forma que en el caso de la suma de números naturales. Recordamos que la resta no puede dar lugar a desbordamiento.

El desbordamiento en la suma y la resta de números en coma fija y signo y magnitud se puede detectar de la misma forma que en el caso de la suma y la resta de números enteros representados en signo y magnitud.

La resta de números sin signo

La resta de números sin signo no puede dar lugar a desbordamiento porque tenemos que restar la magnitud pequeña de la grande. Restar la magnitud grande de la magnitud pequeña no es una operación válida para números sin signo, porque el resultado tendría que ser un número con signo.

Multiplicación y división por 2^k en coma fija binaria

Cuando la base de numeración es 2 en un sistema posicional de base fija, los pesos asociados a los dígitos son potencias de 2. Por consiguiente, multiplicar por 2 se traduce en aumentar en una unidad la potencia de 2 asociada a cada bit y dividir por 2 es equivalente a disminuir en una unidad la potencia de 2 asociada a cada bit.

Ved la multiplicación y la división por potencias de la base de numeración en el subapartado 1.8 de este módulo.

Aplicando el TFN, podemos escribir el número $0011{,}01_{(2}$ de la forma:

$$0011{,}01_{(2} = 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

Si multiplicamos por 2, se obtiene:

$$0011{,}01_{(2} \cdot 2 = (0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}) \cdot 2 =$$
$$= 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} = 00110{,}1_{(2}$$

En un formato de coma fija, la posición de la coma siempre es la misma y, por lo tanto, para multiplicar o dividir por la base desplazamos los bits a izquierda o derecha, añadiendo los ceros necesarios.

Multiplicar por 2^k un número en coma fija sin signo equivale a desplazar los bits k posiciones a la izquierda, completando los n bits del formato con la adición a la derecha de k ceros.

Dividir por 2^k un número en coma fija sin signo equivale a desplazar los bits k posiciones a la derecha, completando los n bits del formato con la adición a la izquierda de k ceros.

En coma fija y 6 bits, donde 2 son fraccionarios, el resultado de multiplicar $0011{,}01_{(2}$ por 2^2 se obtiene desplazando los bits 2 posiciones a la izquierda y añadiendo 2 ceros a la derecha:

$$0011{,}01_{(2} \cdot 2 = 1101{,}\mathbf{00}_{(2}$$

Si el resultado no cabe en el formato, se produce desbordamiento. Esto se da cuando se pierden bits significativos (en coma fija sin signo, bits a 1).

Se produce **desbordamiento** al multiplicar un número sin signo en coma fija por 2^k cuando se pierden uno o más bits significativos (bits a 1) al desplazar los bits k posiciones.

La división por 2^k no produce desbordamiento, pero el cociente puede necesitar más bits fraccionarios que los disponibles en el formato:

$$0101{,}00_{(2} / 2^4 \approx 0000{,}01_{(2} \qquad \text{(en decimal, } + 5_{(10} / 2^4 \approx + 0{,}25_{(10})$$

El símbolo ≈

El símbolo ≈ indica que se trata de una aproximación, no de una igualdad.

La pérdida de bits por la derecha equivale a una aproximación por truncamiento.

En coma fija y signo y magnitud, las operaciones de multiplicación y división por 2^k tienen las mismas características que las descritas más arriba para coma fija sin signo si separamos el bit de signo. El bit de signo se añade al término de la operación.

Actividades

27. Determinad qué valor decimal codifica la cadena de bits 1010010 en los supuestos siguientes:
a) Si se trata de un número en coma fija sin signo de 7 bits donde 4 son fraccionarios.
b) Si se trata de un número en coma fija sin signo de 7 bits donde 1 es fraccionario.

28. Codificad los números $+12{,}85_{(10}$, $+0{,}7578125_{(10}$ y $11{,}025_{(10}$ en una representación fraccionaria binaria en signo y magnitud de 8 bits donde 3 son fraccionarios. Utilizad una aproximación por redondeo en caso de que sea necesario.

29. Si tenemos una representación en coma fija binaria en signo y magnitud de 8 bits donde 3 bits son fraccionarios, determinad los números codificados por las cadenas de bits 01001111, 11001111, 01010100, 00000000 y 10000000.

30. Si las cadenas de bits 00101010, 11010010 y 10100010 representan números en coma fija sin signo de 8 bits donde 3 son fraccionarios, representadlos en un formato de coma fija sin signo de 12 bits donde 4 son fraccionarios.

31. Repetid la actividad anterior considerando que se trata de números en signo y magnitud.

32. Determinad el rango de representación y la precisión en los formatos siguientes:
a) Coma fija en signo y magnitud con 8 bits donde 3 son fraccionarios.
b) Coma fija en signo y magnitud con 8 bits donde 4 son fraccionarios.
c) Coma fija sin signo con 8 bits donde 3 son fraccionarios.
d) Coma fija sin signo con 8 bits donde 4 son fraccionarios.

33. Determinad la precisión necesaria para poder representar el número $+0{,}1875_{(10}$ de forma exacta (sin error de representación) con un formato de coma fija en base 2.

34. Determinad las características de rango y precisión, así como el número de dígitos enteros y fraccionarios necesarios en un formato de coma fija en signo y magnitud, para poder representar de forma exacta los números $+31{,}875_{(10}$ y $16{,}21875_{(10}$

35. Calculad la suma y la resta de los pares de números siguientes, asumiendo que están en coma fija en signo y magnitud con 8 bits donde 3 son fraccionarios. Verificad si el resultado es correcto:
a) $00111000_{(2}$ y $10100000_{(2}$
b) $10111010_{(2}$ y $11101100_{(2}$

3. Otros tipos de representaciones

El funcionamiento de los computadores actuales se basa en la electrónica digital, cuya característica distintiva es que toda la información con la que trabaja se codifica en base a dos únicos valores, que representamos simbólicamente con el 1 y el 0. Por lo tanto, todos los datos que procesa un computador digital tienen que estar representados exclusivamente por cadenas de unos y de ceros, es decir, por cadenas de bits. Entonces, el procesamiento de los datos consiste en aplicar operaciones aritméticas o lógicas a cadenas de bits.

En la sección precedente hemos expuesto las limitaciones inherentes a la tecnología empleada en los computadores digitales actuales y las formas más usuales en las que se codifican los valores numéricos. No ha sido una descripción exhaustiva de las formas de codificación de la información numérica, pero sí una muestra representativa de la manera como la información numérica se codifica para ser procesada dentro de los computadores digitales.

En los apartados siguientes se muestra, por un lado, cómo codificar información que inicialmente no es numérica, usando en último término los símbolos 1 y 0; y, por otro, algunos sistemas de numeración alternativos que tienen especial interés en determinadas circunstancias.

3.1. Representación de información alfanumérica

Se denomina *información alfanumérica* a la información no numérica constituida, básicamente, por el conjunto de letras, cifras y símbolos que se utilizan en las descripciones textuales y que reciben el nombre genérico de *caracteres*.

El número de caracteres empleado en los textos es relativamente grande: letras mayúsculas, letras minúsculas, vocales acentuadas, símbolos de puntuación, símbolos matemáticos, etc.

La representación de los caracteres se lleva a cabo asignando una cadena de bits única y específica para cada carácter, es decir, asignando a cada carácter un código binario. La asignación de códigos podría ser arbitraria, pero en la práctica es conveniente seguir unos criterios que faciliten el procesamiento de la información codificada. Por ejemplo, una asignación de códigos ascendente a las letras del alfabeto facilita la ordenación alfabética: el resultado de una sencilla operación de resta entre los códigos permitirá establecer el orden alfabético.

Siguiendo criterios que faciliten el tratamiento de los datos y con la intención de compatibilizar la información procesada por sistemas diferentes, se han es-

tandarizado unas pocas codificaciones, de entre las cuales, la codificación ASCII es la más extendida.

La codificación ASCII tiene una versión básica de 128 símbolos que forma el estándar *de facto* que usan la mayoría de los computadores, y una versión extendida, no tan estandarizada, que incluye 128 símbolos más. En la versión extendida se usan 8 bits para codificar los 256 símbolos que incluye, mientras que en la básica se usan 7.

En la tabla siguiente se puede ver la asignación de códigos ASCII*. La representación numérica asociada a cada símbolo se obtiene a partir de sus coordenadas. El índice de columna es el dígito decimal menos significativo, y el de fila, el más significativo. Por ejemplo, el carácter alfanumérico "3", se representa por el valor decimal $51_{(10}$ (5u, d1), que en binario es el $00110011_{(2}$.

*** ASCII son las siglas de la expresión inglesa *American standard code for information interchange.***

u / d	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
0u	NULL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1u	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2u	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3u	RS	US	SP	!	"	#	$	%	&	'
4u	(	)	*	+	,	-	.	/	0	1
5u	2	3	4	5	6	7	8	9	:	;
6u	<	=	>	?	@	A	B	C	D	E
7u	F	G	H	I	J	K	L	M	N	O
8u	P	Q	R	S	T	U	V	W	X	Y
9u	Z	[	\	]	^	_	`	a	b	c
10u	d	e	f	g	h	i	j	k	l	m
11u	n	o	p	q	r	s	t	u	v	w
12u	x	y	z	{	|	}	~	DEL		

Nota

Es habitual usar las comillas para distinguir los números (3, 25, 234) de los caracteres o cadenas de caracteres ("3", "25", "234").

Los primeros 31 códigos y el último no corresponden a símbolos del lenguaje o caracteres visibles (el carácter 32 –identificado como SP– representa el espacio en blanco). Estos códigos son caracteres de control utilizados para dar formato al texto o como comandos para los dispositivos periféricos (terminales alfanuméricos o gráficos, impresoras, etc.).

Caracteres de control no visualitzables

Algunos caracteres de control no visualizables son: DEL (borrar), ESC (escapada), HT (tabulador horizontal), LF (final de línea), CR (regreso a primera columna), FF (final de página), STX (inicio de texto) o ETX (final de texto).

Cada vez es más habitual que en los computadores se codifique texto en más de una lengua. Por este motivo, se ha ido popularizando la extensión de la codificación ASCII de los caracteres alfanuméricos a 2 bytes (16 bits), que utiliza el estándar llamado *Unicode* y que incluye los caracteres de las grafías más importantes.

Los códigos ASCII de 8 bits de los caracteres visibles (no así los correspondientes a caracteres de control) se pueden convertir a Unicode añadiendo 8 ceros a la izquierda para completar los 16 bits del estándar Unicode.

El estándar Unicode

El Unicode está estandarizado por la ISO/IEC (*International Organization for Standardization / International Electrotechnical Commission*) con el identificador 10646.

Ejemplo de procesamiento de códigos ASCII

Analizamos los códigos ASCII para averiguar cómo transformar el código de una letra mayúscula en su equivalente en minúscula. Los códigos consecutivos a partir del código 65 siguen el orden de las letras del alfabeto inglés tanto para las mayúsculas como, a partir del código 97, para las minúsculas. Por lo tanto, la distancia entre símbolos de mayúsculas y minúsculas es constante.

En concreto, el carácter "A" tiene el código 65, mientras que el carácter "a" tiene el 97. La diferencia entre los códigos es $32_{(10}$. Por lo tanto, para transformar el código ASCII de una letra mayúscula al código ASCII de la misma letra en minúscula, tenemos que sumar $32_{(10}$, al código ASCII en binario.

Formato Unicode

La codificación de textos en formato Unicode está presente en muchos de los procesadores de textos actuales, como por ejemplo, el Wordpad de Windows.

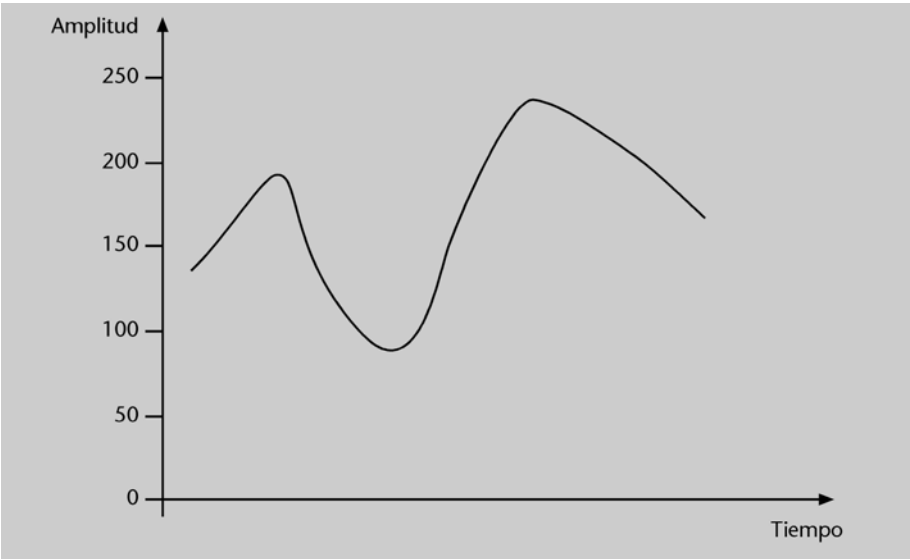
3.2. Codificación de señales analógicas

A veces, los datos que tiene que procesar un computador provienen de dispositivos que recogen información del entorno. Un micrófono, por ejemplo, capta las ondas sonoras que llegan hasta él. Los sensores de estos dispositivos son analógicos, es decir, generan una señal eléctrica de salida que se ajusta de manera continua a la variación del estímulo que reciben. El resultado es una señal eléctrica, cuya variación en el tiempo refleja la variación del estímulo que ha ido llegando al sensor del dispositivo.

Una señal eléctrica analógica es la que codifica la información mediante una variación continua de un parámetro eléctrico (tensión, frecuencia, intensidad) que se ajusta de manera proporcional al estímulo original.

La figura 1 muestra una señal analógica, en la que la amplitud varía de forma continua en el tiempo:

Figura 1



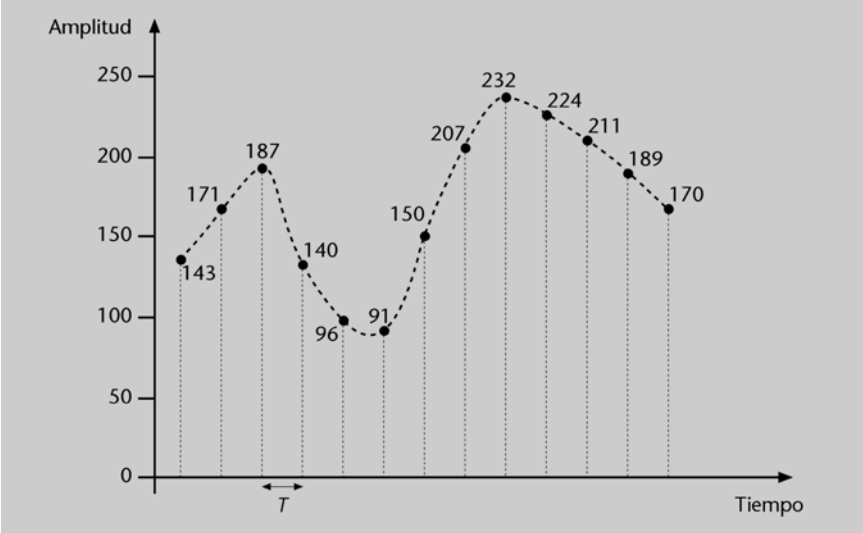
Los procesadores digitales no pueden tratar directamente las señales analógicas y, como en el caso de los números o de los caracteres alfanuméricos, se tienen que codificar utilizando únicamente los símbolos 1 y 0. El proceso que permite esta transformación es la **digitalización**.

La **digitalización** consiste en convertir una representación analógica a una representación digital binaria, es decir, a una secuencia ordenada de números binarios.

El proceso de digitalización consta de tres etapas, que son el muestreo, la cuantificación y la codificación binaria:

1) El **muestreo** (o **discretitzación**) consiste en tomar muestras de la señal analógica a intervalos de tiempos regulares.

Figura 2

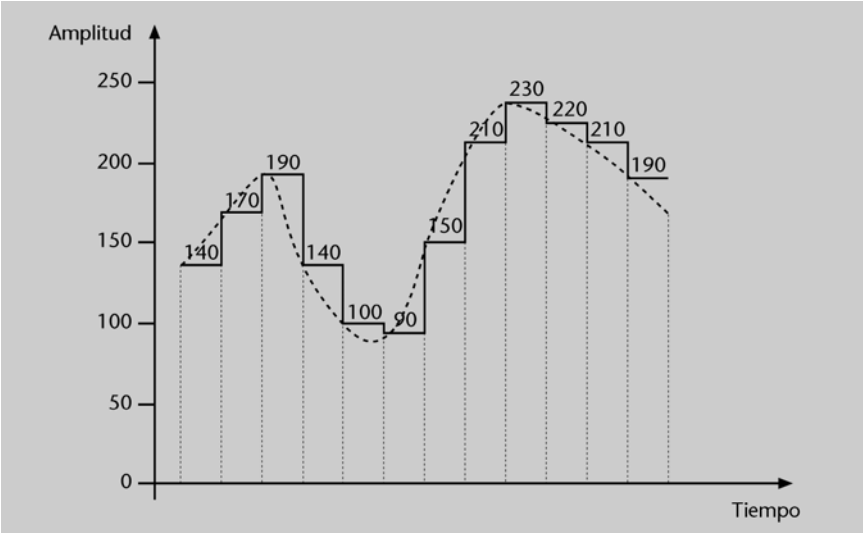


Muestreo de una señal analógica

Un muestreo de la señal dibujada en la figura 1 a intervalos de período *T* daría el resultado que se ve a la figura 2.

2) La **cuantificación** consiste en asignar un valor, de entre un conjunto finito, a la amplitud de la señal en cada intervalo de muestreo.

Figura 3



Cuantificación de una señal

Si admitimos únicamente valores múltiples de 10, la cuantificación del muestreo de la figura 2 da lugar a la secuencia de valores numéricos: 140, 170, 190, 140, 100, etc. que representa una aproximación "escalonada" a la señal continua original, como se muestra en la figura 3.

Cuanto más pequeños sean los intervalos de tiempo del muestreo (hasta un cierto límite más allá del cual ya no ganamos nada), y cuanto mayor sea el conjunto de valores admitidos en la cuantificación, más cercana será la información digitalizada a la información analógica original.

Ventajas de la digitalización

La digitalización es la técnica que se usa, por ejemplo, para grabar música en un CD. El sonido se muestrea, se codifica en binario y se graba en el CD haciendo muescas: un 1 se traduce en hacer una muesca, un 0 se traduce en no hacer muesca.

La tecnología actual permite hacer el muestreo y la cuantificación suficientemente acotados para que la distancia entre los escalones provocados por la digitalización no sean perceptibles por el oído humano. Por otro lado, la digitalización evita los ruidos y distorsiones que se introducen con medios analógicos, cosa que permite que el sonido digital sea de más calidad que el analógico.

3) La **codificación binaria** consiste en traducir los valores de las muestras a un sistema binario, es decir, expresar los valores mediante ceros y unos.

Ejemplo de codificación binaria

Los valores de las amplitudes que aparecen en la figura 3 van desde el 90 hasta el 230. Podemos codificar este rango de valores decimales en binario usando 8 bits (puesto que $2^7 < 230 < 2^8$). La tabla siguiente muestra la codificación en binario de los valores decimales de la cuantificación de la figura 3:

Amplitud	Codificación binaria
90	01011010
100	01100100
140	10001100
150	10010110
170	10101010
190	10111110
210	11010010
220	11011100
230	11100110

De hecho, conviene tener en cuenta que en la cuantificación sólo se han permitido múltiplos de 10. Así, en los valores de amplitud podemos despreciar el cero de la derecha y considerar el rango de valores [9, 23]. Para codificar en binario los números dentro de este rango son suficientes 5 bits (puesto que $2^4 < 23 < 2^5$):

Amplitud	Codificación binaria
90	01001
100	01010
140	01110
150	01111
170	10001
190	10011
210	10101
220	10110
230	10111

Esta codificación es más eficiente, porque utiliza un menor número de bits. Usando esta codificación, la información que en la figura 3 se expresaba mediante una línea curva ahora se expresa por la secuencia de códigos binarios siguiente:

01110

10001

10011

01110

01010

01001

01111

10101

10111

10110

10101

10011

Esta secuencia de valores binarios constituye una aproximación escalonada a la curva continua de la figura 1.

La digitalización permite transformar en números cualquier señal analógica de nuestro entorno y conseguir, así, que se pueda procesar dentro de un computador digital.

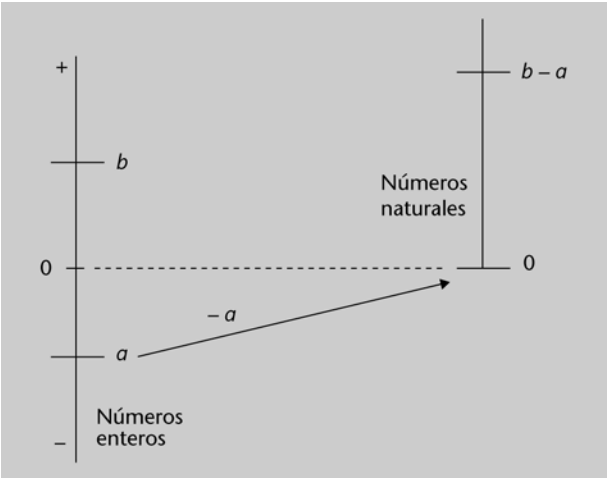
3.3. Otras representaciones numéricas

La sección 2 está dedicada a la descripción de las codificaciones más usuales de números enteros y fraccionarios tanto con signo como sin signo. Allí se han descrito las codificaciones más utilizadas para representar información numérica dentro de los computadores, sin embargo, hay algunas representaciones más que conviene conocer y que se describen en los apartados siguientes.

3.3.1. Representación en exceso a *M*

El exceso a *M* es un tipo de representación de números enteros, donde la estrategia que se sigue es transformar el conjunto de valores numéricos enteros que se quiere representar en un conjunto de números naturales, donde el valor más negativo esté codificado por el cero. El resto de valores se codifican a partir del cero en orden ascendente.

La figura siguiente muestra gráficamente esta estrategia:



Consideremos el intervalo [–5, +5]. Para desplazar este rango de valores enteros a un conjunto de valores naturales, sólo tenemos que sumar 5 a cada número entero del intervalo. De esta forma, los números pasan a estar en el intervalo [0, 10]. Con este desplazamiento, el valor entero –5 da lugar al valor natural 0, puesto que –5 – (–5) = –5 +5 = 0; el valor +2 da lugar al valor natural 7, puesto que +2 – (–5) = +2 +5 = 7; etc.

El intervalo [*a* , *b*] de números enteros se puede desplazar al intervalo de números naturales [0, *b* – *a*] restando *a* a cada entero del intervalo [*a* , *b*].

Este tipo de estrategia se denomina representación en **exceso a *M***, donde *M* es el desplazamiento que se aplica al intervalo de enteros que se quiere codificar.

La **representación en exceso a *M*** de un número entero *X* se obtiene sumando el desplazamiento *M* al valor numérico *X*. Por consiguiente, encontraremos el valor de un número codificado en exceso a *M*, restando *M* a la codificación.

A modo de ejemplo, podemos decir que en una representación en exceso a 10, el número entero –4 se representa mediante el número natural 6 (puesto que –4 + 10 = 6), y que el 0 se representa mediante el número 10 (puesto que 0 + 10 = 10).

Dado que dentro del computador la información se codifica en binario, los números naturales empleados en la representación en exceso a *M* se codifican en binario.

Representación en exceso a *M*

Para representar el valor $-6_{(10}$ en exceso a 7 y 4 bits procederemos de la forma siguiente:

1) Sumamos el desplazamiento para encontrar la codificación en decimal $-6_{(10} + 7_{(10} = 1_{(10}$.
2) Codificamos en binario y 4 bits el número obtenido $1_{(10} = 0001_{(2}$.

El valor $-6_{(10}$ se codifica en exceso a 7 y 4 bits como 0001.

Para saber qué valor representa la cadena de bits 1100 que está codificada en exceso a 7 y 4 bits,procederemos de la forma siguiente:

1) Hacemos un cambio de base para encontrar la codificación en decimal $1100_{(2} = 12_{(10}$.
2) Restamos el desplazamiento para encontrar el valor codificado $12_{(10} - 7_{(10} = +5_{(10}$.

El valor que representa la cadena de bits 1101 codificada en exceso a 7 y 4 bits es el $+5_{(10}$.

El exceso a *M* es un tipo de representación de números enteros empleado para codificar el valor del exponente cuando se trabaja en coma flotante.

3.3.2. Representación en coma flotante

A menudo se tienen que representar números muy grandes (como por ejemplo la velocidad de transmisión de la luz en el vacío c=299792500 m/s) o bien números muy pequeños (como la masa de un electrón m_e = 0,00000000000000000000000000000091095 kg), y quizás de forma simultánea. Para evitar el uso de un gran número de dígitos en la representación de estos números, se emplea el formato de **coma flotante**.

Coma flotante

La ventaja de la coma flotante es la capacidad de representar con pocos dígitos números que en otros formatos necesitan muchos dígitos para ser representados.

Los números en **coma flotante** toman la forma:

$$\pm R \cdot b^e$$

donde + o – indica el signo de la magnitud representada, R es un número fraccionario que recibe el nombre de **mantisa** , b es la base de numeración y e es un número entero que recibe el nombre de **exponente**.

Terminología

La **coma flotante** también recibe el nombre de **notación científica**.

La mantisa contiene los dígitos significativos de la magnitud y viene precedida por el signo de esta magnitud. El exponente indica el número de posiciones a la derecha (exponente positivo) o a la izquierda (exponente negativo) que tenemos que desplazar la coma fraccionaria de la mantisa para obtener el valor numérico representado. El número $+32{,}74_{(10} \cdot 10^2$ es equivalente al $+3274_{10}$, mientras que $+32{,}74_{(10} \cdot 10^{-1}$ es equivalente a $+3{,}274_{(10}$. El valor del exponente indica la posición relativa de la coma fraccionaria.

Atención

A lo largo del texto usaremos las letras ***S***, ***e*** y ***R*** para referirnos, respectivamente, al signo, el exponente y la mantisa.

En el trabajo manual adaptamos dinámicamente el número de dígitos empleados en la mantisa y el exponente de la representación en coma flotante. Podemos utilizar $+2{,}995 \cdot 10^8$ o $+0{,}02995 \cdot 10^{10}$ según convenga a nuestras necesidades. Dentro de los computadores tenemos que ceder esta flexibilidad y adoptar restricciones que simplifiquen el procesamiento de datos. Por eso, se asume que la base de numeración es 2 y que el número de bits destinados a la mantisa y al exponente se fija en la especificación del formato.

Símbolo +

El símbolo + no suele aparecer delante del exponente o de la mantisa cuando son positivos.

Para la representación en coma flotante dentro de los computadores se asume que **la base de numeración es 2**, y que la definición del formato fija el número de bits de la mantisa y el número de bits del exponente.

Por otro lado, la representación de un valor numérico en coma flotante no es única. Por ejemplo, algunas representaciones en coma flotante del número $26300_{(10}$ son: $2{,}63_{(10} \cdot 10^4$, $0{,}263_{(10} \cdot 10^5$, $263_{(10} \cdot 10^2$, $2630_{(10} \cdot 10^1$, $26300_{(10} \cdot 10^0$ o $263000_{(10} \cdot 10^{-1}$. De nuevo, simplificamos el tratamiento de estos números, si se restringe esta flexibilidad, fijando el formato de la mantisa. Por ejemplo, el formato puede determinar que la coma está a la derecha del primer dígito no nulo. Con esta limitación, el $26300_{(10}$ tiene una representación única que es $2{,}63 \cdot 10^4$.

Nota

En coma flotante, la representación de un valor numérico no es única.

Nota

Fijando la posición de la coma de la mantisa se facilita la comparación de números.

Para evitar la multiplicidad de representaciones de un valor numérico, propia de la representación en coma flotante, en la definición del formato **se fija la posición de la coma fraccionaria respecto al primer dígito no nulo** de la mantisa.

Cuando la mantisa tiene fijada la posición de la coma fraccionaria, recibe el calificativo de **mantisa normalizada**.

Las posiciones más habituales en las que se fija la coma de la mantisa son la izquierda del primer dígito no nulo y, especialmente, la derecha del primer dígito no nulo.

Los valores numéricos representados en coma flotante con mantisa normalizada y coma a la derecha del primer bit no nulo son de la forma:

$$\pm 1, x_{-1}x_{-2}\cdots x_{-k}\cdot 2^{e}$$

donde x_i son dígitos binarios (bits), 2 es la base de numeración en decimal y e es el exponente.

Coma flotante con mantisa normalizada

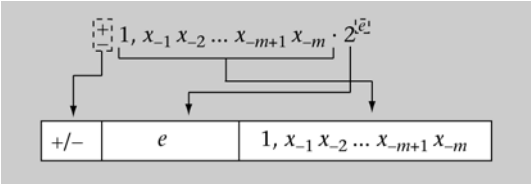
En las representaciones en coma flotante con la coma de la mantisa fijada a la izquierda del primer dígito no nulo, los números son de la forma:

$$\pm 0,1x_{-2}\cdots x_{-k}\cdot 2^{e}.$$

La codificación en coma flotante tiene que incorporar la información de signo (habitualmente 0 para los positivos y 1 para los negativos), el valor de la mantisa y el del exponente. En cambio, no se guarda la base de numeración dado que se asume que siempre es 2. La figura siguiente muestra el orden en el que usualmente se disponen estos valores:

El bit de signo

Como norma, el bit de signo es 0 para números positivos y 1 para números negativos.



Signo-exponente-mantisa

El orden de precedencia signo-exponente-mantisa no es el único posible, pero sí el más ampliamente utilizado.

En coma flotante, el exponente suele estar restringido a los enteros. Para codificarlo, lo más habitual es usar exceso a M, donde M toma frecuentemente los valores 2^{q-1} o $2^{q-1} - 1$, donde q es el número de bits del exponente.

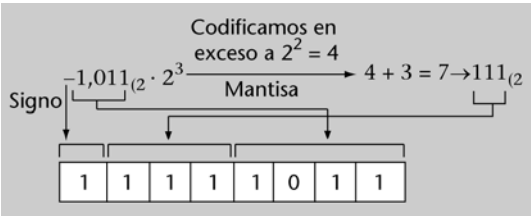
Representación del exponente

Por el exponente, el más extendido es el uso de exceso a M, pero se podría usar otro tipo de codificación, como por ejemplo Ca2.

En coma flotante de 8 bits, mantisa normalizada de 4 bits y exponente en exceso a M, donde M toma el valor 2^{q-1} y q es el número de bits disponibles para la representación del exponente, la codificación del número $-10{,}11_{(2}\cdot 2^{2}$ es la que se muestra a la figura siguiente:

Nota

De los 8 bits, 4 son por la mantisa y 1 por el signo. Restan 3 para el exponente. Si M es 2^{q-1}, $M = 2^{3-1} = 2^{2} = 4$.
El exponente se codifica en exceso a 4.



La tabla siguiente muestra algunos números representados en este formato:

Números	*S*	*e*			*R*			
$-1{,}101_{(2} \cdot 2^{-1}$	1	0	1	1	1	1	0	1
$1{,}101_{(2} \cdot 2^{-1}$	0	0	1	1	1	1	0	1
$1{,}0_{(2} \cdot 2^{-3}$	0	0	0	1	1	0	0	0
$-1{,}10_{(2} \cdot 2^{+1}$	1	1	0	1	1	1	0	0

En la tabla precedente, podemos observar que el primer bit de la mantisa (columna *R*) siempre es 1, porque se codifican de la forma $1{,}x_{-1}x_{-2}\cdots$. Tienen una parte fija (1,) y una parte variable ($x_{-1}x_{-2}\cdots$). Para optimizar recursos, podemos almacenar sólo la parte variable, puesto que la parte fija es conocida y común a todos los números. Esto permitirá almacenar 1 bit más para la mantisa, aumentando así su precisión, o bien utilizar un bit menos en la representación. Las mantisas almacenadas aplicando esta técnica reciben el nombre de **mantisas con bit implícito**.

La técnica del **bit implícito** consiste en almacenar sólo la parte variable de las mantisas normalizadas y asumir la parte fija como conocida y definida en el formato de la representación.

El uso del bit implícito permite almacenar mantisas 1 bit más grandes o reducir en 1 bit el número de bits necesarios para la representación de la mantisa.

Codificación de un número decimal en coma flotante normalizada y binaria

Para codificar el número $+104_{(10}$ en un formato de coma flotante normalizada de 8 bits, de los cuales 3 bits se destinan a la mantisa con bit implícito y exponente en exceso a *M*, seguiremos el proceso siguiente:

1) Codificar el número $+104_{(10}$ en base 2.

Si aplicamos el método de cambio de base basado en divisiones sucesivas, obtenemos que $104_{(10} = 1101000_{(2}$.

2) Normalizar la mantisa de la forma $1{,}x_{-1}x_{-2}x_{-3}\ldots$: $1101000_{(2} = 1{,}101 \cdot 2^6$

3) Identificar el signo, el exponente y la mantisa.

a) El número es positivo; por lo tanto, el bit de signo será 0: $S = 0$.
b) La mantisa de este número es 1,101. El formato indica que trabajamos con una mantisa de 3 bits y bit implícito. Por lo tanto, guardaremos los 3 bits a la derecha de la coma: 101.
c) El exponente toma el valor 6.

4) Codificar en exceso a *M* el exponente. De los 8 bits del formato, 3 se usan para la mantisa y 1 para el signo. Restan 4 para el exponente. Por lo tanto, el valor del exceso es $2^{4-1} = 2^3 = 8$. El $6_{(10}$ codificado en exceso a 8 es $6_{(10} + 8_{(10} = 14_{(10}$. Si aplicamos otra vez el método de cambio de base basado en divisiones sucesivas, encontramos que el $14_{(10}$ en base 2 es el $1110_{(2}$. Por lo tanto, $e = 1110$.

5) Unir las codificaciones de signo, exponente y mantisa en la orden de precedencia correcto ($S - e - R$) para obtener la representación final:

S	Exponente				Mantisa		
0	1	1	1	0	1	0	1

Cambio de base

Para cambiar a base 2 el $104_{(10}$ hacemos divisiones sucesivas:

$104 = 52 \cdot 2 + 0$
$52 = 26 \cdot 2 + 0$
$26 = 13 \cdot 2 + 0$
$13 = 6 \cdot 2 + 1$
$6 = 3 \cdot 2 + 0$
$3 = 1 \cdot 2 + 1$
$1 = 0 \cdot 2 + 1$

$104_{(10} = 1101000_{(2}$

Cambio de base

Para cambiar a base 2 el $14_{(10}$ hacemos divisiones sucesivas:

$14 = 7 \cdot 2 + 0$
$7 = 3 \cdot 2 + 1$
$3 = 1 \cdot 2 + 1$
$1 = 0 \cdot 2 + 1$

$14_{(10} = 1110_{(2}$

Por lo tanto, la codificación del número $+104_{(10}$ en el formato binario de coma flotante especificado es 0<u>1110</u>101*.

*** El subrayado destaca la posición del exponente dentro de una cadena de bits que codifica un número en coma flotante.**

Descodificación de un número en coma flotante normalizada binaria

Para hallar el valor decimal del número 01010101, que está en un formato de coma flotante normalizada de 8 bits con 4 bits de mantisa con bit implícito y exponente en exceso, seguiremos el proceso siguiente:

1) Identificar el signo.
Si asumimos que el formato mantiene signo, exponente y mantisa en este orden, el bit del extremo izquierdo codifica el signo. En el 01010101 el primer bit es 0; por lo tanto, el signo es positivo.

2) Identificar la mantisa.
Dado que la mantisa ocupa las 4 posiciones más bajas, se trata de los bits 0101. Ahora bien, el formato indica la existencia de bit implícito. Por lo tanto, la mantisa es en realidad $1{,}0101_{(2}$.

3) Identificar el exponente.
El exponente está determinado por los 3 bits que quedan:

S	Exponente			Mantisa			
0	**1**	**0**	**1**	0	1	0	1

M toma el valor 2^{q-1}, por lo cual $M = 2^{3-1} = 2^2 = 4$. Así, el exponente codificado es $101_{(2} - 4_{(10} = 5_{(10} - 4_{(10} = 1_{(10}$. El exponente $e = 1_{(10}$.

4) Unir signo, exponente y mantisa.
Signo positivo, mantisa $1{,}0101_{(2}$ y exponente $1_{(10}$: el número representado es el $+1{,}0101 \cdot 2^1$.

5) Hacer un cambio de base con objeto de hallar el valor decimal.
Si aplicamos el TFN, tenemos que:

$$+1{,}0101_{(2} \cdot 2^1 = (1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}) \cdot 2^1 = 2^1 + 2^{-2} \cdot 2^1 + 2^{-4} \cdot 2^1 = +2{,}625_{(10}$$

Por lo tanto, el número 0<u>101</u>0101$_{(2}$ codifica en el formato de coma flotante especificado el valor decimal $+2{,}625_{(10}$.

Para hallar el valor decimal del número $10010001_{(2}$, que está en un formato de coma flotante normalizada de 8 bits con 5 bits de mantisa con bit implícito y exponente en exceso, seguiremos el proceso siguiente:

1) Identificar el signo.
Si asumimos que el formato mantiene signo, exponente y mantisa en este orden, el bit del extremo izquierdo codifica el signo. En el 10010001 el primer bit es 1; por lo tanto, el signo es negativo.

2) Identificar la mantisa.
Dado que la mantisa ocupa las 5 posiciones más bajas, se trata de los bits 10001. El formato indica la existencia de bit implícito. Por lo tanto, la mantisa es, en realidad, $1{,}10001_{(2}$.

3) Identificar el exponente.
El exponente está determinado por los 2 bits que quedan:

S	e		Mantisa				
1	**0**	**0**	1	0	0	0	1

M toma el valor 2^{q-1}, por lo cual $M = 2^{2-1} = 2^1 = 2$. Así, el exponente codificado es $00_{(2} - 2_{(10} = 0_{(10} - 2_{(10} = -2_{(10}$. El exponente $e = -2_{(10}$.

4) Unir signo, exponente y mantisa.
Signo negativo, mantisa $1{,}10001_{(2}$ y exponente $-2_{(10}$: el número representado es el $-1{,}10001_{(2} \cdot 2^{-2}$.

5) Hacer un cambio de base para encontrar el valor decimal.

Si aplicamos el TFN, tenemos que:

$$-1{,}10001_{(2}\cdot 2^{-2} = -(1 \cdot 2^{0} + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5}) \cdot 2^{-2} =$$
$$= -(2^{-2} + 2^{-1}\cdot 2^{-2} + 2^{-5}\cdot 2^{-2}) = -0{,}3828125_{(10}$$

Por lo tanto, el número $1\underline{00}10001_{(2}$ codifica en el formato de coma flotante especificado el valor decimal $-0{,}3828125_{(10}$.

3.3.3. Representación BCD

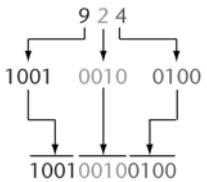
Una estrategia alternativa para la representación de números es la codificación directa de los dígitos decimales, sin hacer un cambio de base. Como en el caso de la codificación de la información alfanumérica, tenemos que asignar un código binario a cada dígito decimal. La tabla siguiente muestra la codificación binaria de los dígitos decimales en BCD:

Dígito decimal	Codificación binaria
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD

BCD son las siglas de *binary coded decimal,* es decir, decimal codificado en binario.

Los códigos BCD de los dígitos decimales son de 4 bits. La figura siguiente muestra la forma en que podemos representar un número decimal si codificamos cada dígito individualmente, lo que se denomina **representación BCD**:



La **representación BCD** (*binary coded decimal*) consiste en codificar los números decimales dígito a dígito. Cada dígito decimal se sustituye por 4 bits que corresponden a la codificación en binario del dígito decimal.

La codificación binaria dígito a dígito de los números decimales aprovecha parcialmente la capacidad de representación. El código 1100, por ejemplo, no

se usa. De las 16 combinaciones posibles que se pueden hacer con 4 bits, sólo se usan 10. Por consiguiente, la representación de un número en BCD necesita más bits que en binario.

Este tipo de representación es habitual en dispositivos de salida para visualizar datos.

Que los dígitos estén codificados en binario individualmente no cambia el hecho de que se trata de números decimales. Las operaciones de suma y resta se desarrollarán como en el caso de base 10. De hecho, lo único que se está haciendo en esta forma de representación es cambiar el símbolo que utilizamos para designar un dígito decimal por un código binario que tiene la misma función. Se puede entender como un cambio de la simbología para representar los dígitos.

Actividades

36. Codificad en BCD el número $125_{(10}$.

37. Codificad en BCD el número $637_{(10}$.

38. Indicad qué número codifica la representación BCD siguiente 00010011100.

39. Codificad el número $427_{(10}$ en BCD y en binario. Comparad el número de bits necesario en los dos casos.

40. Hallad el valor decimal que codifican las cadenas de bits siguientes, interpretando que se trata de números en un formato de coma flotante de 8 bits con mantisa normalizada de la forma 1,*X* y con bit implícito:
a) 11110010, donde la mantisa es de 4 bits.
b) 01010011, donde la mantisa es de 3 bits.

41. Haced las codificaciones siguientes:
a) El número $-1{,}335_{(10}$ en coma flotante de 8 bits, mantisa de 3 bits normalizada de la forma 1,*X* y con bit implícito empleando una aproximación por truncamiento.
b) Repetid el apartado anterior, pero con una aproximación por redondeo.
c) El número $10{,}0327_{(10}$ en coma flotante de 9 bits, mantisa de 3 bits normalizada de la forma 1,*X* y con bit implícito empleando una aproximación por truncamiento.

42. Determinad si el número $\mathbf{2{,}89_{(10} \cdot 10^{10}}$ es representable en un formato de coma flotante de 16 bits, con mantisa normalizada de la forma 1,*X*, bit implícito y 5 bits para el exponente.

43. Determinad si el número $\mathbf{-1256_{(10} \cdot 10^{-2}}$ es representable en un formato de coma flotante de 10 bits, con mantisa normalizada de la forma 1,*X*, bit implícito y 6 bits para el exponente.

Resumen

En este módulo se presenta un análisis de los sistemas de numeración posicionales y se exponen las formas de representar valores numéricos que es habitual utilizar dentro de los computadores. Los puntos principales que se tratan en este módulo son:

- El TFN y el algoritmo de divisiones sucesivas que permiten cambiar entre bases diferentes la representación de un valor numérico.

- La representación de números naturales mediante representaciones posicionales empleando base 2 (binario), base 16 (hexadecimal) y base 10 (decimal), así como las operaciones de suma y resta de números naturales.

- Las limitaciones derivadas de los condicionamientos físicos de los computadores y las características que presentan los diferentes formatos de representación (rango y precisión), así como el fenómeno del desbordamiento y las técnicas de aproximación.

- La codificación de los números enteros empleando las representaciones en complemento a 2 y signo y magnitud, y las operaciones de suma y resta en cada una de estas codificaciones.

- La codificación de números fraccionarios con y sin signo en coma fija.

- El empaquetamiento de información en hexadecimal y la codificación en BCD.

Ejercicios de autoevaluación

1. Codificad en complemento a 2 y signo y magnitud el número $-10_{(10}$ empleando 8 bits.

2. Determinad el valor decimal que codifica la cadena de bits 00100100 en los supuestos siguientes:
a) Si se trata de un número codificado en complemento a 2.
b) Si se trata de un número codificado en signo y magnitud.

3. Sumad en binario los números $111010101100_{(2}$ y $11100010010_{(2}$. Analizad el resultado obtenido.

4. Codificad en un formato de coma fija de 8 bits donde 3 son fraccionarios y en signo y magnitud, el número $+12{,}346_{(10}$. Emplead una aproximación por truncamiento en caso de que sea necesario.

5. Codificad en un formato de 8 bits y complemento a 2 el número $-45_{(10}$.

6. Determinad el número mínimo de bits enteros y fraccionarios necesarios en coma fija y signo y magnitud para codificar el número $-35{,}25_{(10}$.

7. Codificad los números $+12{,}25_{(10}$ y el $+32{,}5_{(10}$ en un formato de coma fija y signo y magnitud de 9 bits donde 2 son fraccionarios y sumadlos.

8. Codificad en la representación BCD el número $178_{(10}$.

9. Codificad en un formato de coma flotante de 8 bits con mantisa de 3 bits normalizada de la forma $1{,}X$ con bit implícito, y exponente en exceso a M con $M = 2^{q-1}$ (donde q es el número de bits del exponente), el número $+12{,}346_{(10}$. Emplead una aproximación por truncamiento en caso de que sea necesario.

10. Determinad el valor decimal que representa el código $378_{(16}$, sabiendo que se trata de un número en coma flotante de 12 bits con mantisa de 4 bits normalizada de la forma $1{,}X$ y bit implícito, exponente en exceso a M con $M = 2^{q-1}$ (donde q es el número de bits del exponente) y empaquetado en hexadecimal.

Solucionario

Actividades

1. Convertid a base 10 los valores siguientes:

a) $10011101_{(2} = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$
$= 128 + 16 + 8 + 4 + 1 = 157_{(10}$

b) $3AD_{(16} = 3 \cdot 16^2 + A \cdot 16^1 + D \cdot 16^0 = 3 \cdot 16^2 + \underset{\uparrow}{10} \cdot 16^1 + \underset{\uparrow}{13} \cdot 16^0 = 941_{(10}$

Correspondencia del dígito A en base 10 Dígito D en base 10

c) $333_{(4} = 3 \cdot 4^2 + 3 \cdot 4^1 + 3 \cdot 4^0 = 48 + 12 + 3 = 63_{(10}$

d) $333_{(8} = 3 \cdot 8^2 + 3 \cdot 8^1 + 3 \cdot 8^0 = 192 + 24 + 3 = 219_{(10}$

e) $B2{,}3_{(16} = B \cdot 16^1 + 2 \cdot 16^0 + 3 \cdot 16^{-1} = 11 \cdot 16^1 + 2 \cdot 16^0 + 3 \cdot 16^{-1} =$
$= 176 + 2 + 0{,}1875 = 178{,}1875_{(10}$

f) $3245_{(8} = 3 \cdot 8^3 + 2 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0 = 1536 + 128 + 32 + 5 = 1701_{(10}$

g) $AC3C_{(16} = A \cdot 16^3 + C \cdot 16^2 + 3 \cdot 16^1 + C \cdot 16^0 = 10 \cdot 16^3 + 12 \cdot 16^2 + 3 \cdot 16^1 + 12 \cdot 16^0 =$
$= 40960 + 3072 + 48 + 12 = 44092_{(10}$

h) $1010{,}11_{(8} = 1 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 + 0 \cdot 8^0 + 1 \cdot 8^{-1} + 1 \cdot 8^{-2} =$
$= 512 + 8 + 0{,}125 + 0{,}015625 = 520{,}140625_{(10}$

i) $110011{,}11_{(4} = 1 \cdot 4^5 + 1 \cdot 4^4 + 0 \cdot 4^3 + 0 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 + 1 \cdot 4^{-1} + 1 \cdot 4^{-2} =$
$= 1024 + 256 + 4 + 1 + 0{,}25 + 0{,}0625 = 1285{,}3125_{(10}$

j) $10011001{,}1101_{(2} = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 +$
$+ 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} =$
$= 128 + 16 + 8 + 1 + 0{,}5 + 0{,}25 + 0{,}0625 = 153{,}8125_{(10}$

k) $1110100{,}01101_{(2} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} +$
$+ 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} =$
$= 64 + 32 + 16 + 4 + 0{,}25 + 0{,}125 + 0{,}03125 = 116{,}40625_{(10}$

2. Convertid a base 2 los valores siguientes:

a) $425_{(10}$

Dividimos 425 por 2 sucesivamente, y registramos los restos de las divisiones enteras. Estos restos son los dígitos binarios:

425	1
212	0
106	0
53	1
26	0
13	1
6	0
3	1
1	

Por lo tanto: $425_{(10} = 110101001_{(2}$

b) $344_{(10}$

344	0
172	0
86	0
43	1
21	1
10	0
5	1
2	0
1	

Por lo tanto: $344_{(10} = 101011000_{(2}$

c) $31{,}125_{(10}$

- Parte fraccionaria

0,125 · 2	=	0,25	=	0	+ 0,25
0,25 · 2	=	0,50	=	0	+ 0,50
0,50 · 2	=	1,00	=	1	+ 0,00

- Parte entera

31	1
15	1
7	1
3	1
1	

Por lo tanto, si $0{,}125_{(10} = 0{,}001_{(2}$ i $31_{(10} = 11111_{(2}$, entonces $31{,}125_{(10} = 11111{,}001_{(2}$

d) $4365{,}14_{(10}$

- Parte fraccionaria

0,14 · 2	=	0,28	=	0	+ 0,28
0,28 · 2	=	0,56	=	0	+ 0,56
0,56 · 2	=	1,12	=	1	+ 0,12
0,12 · 2	=	0,24	=	0	+ 0,24
0,24 · 2	=	0,48	=	0	+ 0,48
0,48 · 2	=	0,96	=	0	+ 0,96
0,96 · 2	=	1,92	=	1	+ 0,92
0,92 · 2	=	1,84	=	1	+ 0,84
0,84 · 2	=	1,68	=	1	+ 0,68
0,68 · 2	=	1,36	=	1	+ 0,36
0,36 · 2	=	0,72	=	0	+ 0,72
0,72 · 2	=	1,44	=	1	+ 0,44
...					

- Parte entera

4365	1
2182	0
1091	1
545	1
272	0
136	0
68	0
34	0
17	1
8	0
4	0
2	0
1	

Por lo tanto, si $0{,}14_{(10} = 0{,}001000111101...{}_{(2}$ i $4365_{(10} = 1000100001101_{(2}$, entonces

$$4365{,}14_{(10} = 4365_{(10} + 0{,}14_{(10} = 1000100001101_{(2} + 0{,}001000111101...{}_{(2} = 1000100001101{,}001000111101...{}_{(2}$$

3. Convertid a hexadecimal los números siguientes:

a) $111010011{,}1110100111_{(2}$

Base 2	**0001**	1101	0011,	1110	1001	**1100**
Base 16	1	D	3,	E	9	C

Por lo tanto: $111010011{,}1110100111_{(2} = 1D3{,}E9C_{(16}$

b) $0{,}1101101_{(2}$

Base 2	0,	1101	**1010**
Base 16	0,	D	A

Por lo tanto: $0{,}1101101_{(2} = 0{,}DA_{(16}$

c) $45367_{(10}$

Dividiremos el valor numérico 45367 por 16 sucesivamente, y registraremos los restos de las divisiones enteras realizadas. Estos restos constituyen los dígitos hexadecimales:

45367	7
2835	3
177	1
11	

Por lo tanto: $45367_{(10} = B137_{(16}$

d) $111011{,}1010010101_{(2}$

Base 2	**00**11	1011,	1010	0101	01**00**
Base 16	3	B,	A	5	4

Por lo tanto: $111011,\ 1010010101_{(2} = 3B{,}A54_{(16}$

4. Convertid los números hexadecimales siguientes a base 2, base 4 y base 8:

Podemos aprovechar la propiedad $16 = 2^4$ y $16 = 4^2$ para tratar el paso de base 16 a base 2 y a base 4 dígito a dígito. Esto es, cada dígito hexadecimal se transformará en un conjunto de cuatro dígitos binarios, mientras que cada dígito hexadecimal se puede transformar en dos dígitos en base 4.

El paso a base 8 no se puede hacer directamente desde base 16, dado que 16 no es potencia de 8. Aprovecharemos la base 2 como base intermedia. 8 es potencia de 2 ($8 = 2^3$) y, por lo tanto, tenemos una correspondencia directa: cada agrupación de tres dígitos binarios se corresponderá con un dígito octal.

a) $ABCD_{(16}$

Base 16	A	B	C	D
Base 2	1010	1011	1100	1101
Base 4	22	23	30	31

Base 2	**00**1	010	101	111	001	101
Base 8	1	2	5	7	1	5

Por lo tanto: $ABCD_{(16} = 1010101111001101_{(2} = 22233031_{(4} = 125715_{(8}$

b) $45{,}45_{(16}$

Base 16	4	5,	4	5
Base 2	0100	0101,	0100	0101
Base 4	10	11,	10	11

Base 2	**0**01	000	101,	010	001	01**0**
Base 8	1	0	5,	2	1	2

Por lo tanto: $45{,}45_{(16} = 1000101{,}01000101_{(2} = 1011{,}1011_{(4} = 105{,}212_{(8}$

c) $96FF{,}FF_{(16}$

Base 16	9	6	F	F,	F	F
Base 2	1001	0110	1111	1111,	1111	1111
Base 4	21	12	33	33,	33	33

Base 2	**00**1	001	011	011	111	111,	111	111	11**0**
Base 8	1	1	3	3	7	7,	7	7	6

Por lo tanto: $96FF{,}FF_{(16} = 1001011011111111{,}11111111_{(2} = 21123333{,}3333_{(4} = 113377{,}776_{(8}$

5. Rellenad la tabla siguiente:

Como se puede ver, el valor numérico que en base 10 se representa por 74,3, con una representación en base 2, 8 y 16 tiene un número infinito de dígitos fraccionarios. En todos estos casos obtenemos una parte fraccionaria periódica.

Binario	Octal	Hexadecimal	Decimal
1101100,110	154,6	6C,C	108,75

Binario	Octal	Hexadecimal	Decimal
11110010,010011	**362,23**	F2,4C	242,296875
10100001,00000011	241,006	**A1,03**	161,01171875
1001010,0100110011...	112,2314631463...	4A,4CCCCC...	**74,3**

6. Empaquetad en hexadecimal la cadena de bits 10110001.

Para empaquetar sólo hay que agrupar los bits de 4 en 4 y hacer el cambio de base a hexadecimal. En este caso tenemos los grupos 1011 | 0001:

$1011_{(2} = B_{(16}$
$0001_{(2} = 1_{(16}$

Ahora, agrupamos todos estos dígitos en una única cadena y obtenemos $10110001_{(2} \rightarrow$ **B1h**.

7. Empaquetad en hexadecimal el número $0100000111,111010_{(2}$, que está en un formato de coma fija de 16 bits, de los cuales 6 son fraccionarios.

Se agrupan los bits de 4 en 4: 0100 | 0001 | 1111 | 1010 y se hace el cambio de base de cada grupo:

$0100_{(2} = 4_{(16}$
$0001_{(2} = 1_{(16}$
$1111_{(2} = F_{(16}$
$1010_{(2} = A_{(16}$

Por lo tanto, la cadena 0100000111111010 se empaqueta en hexadecimal por **41FAh**.

Observad que no hemos hecho un cambio de base del número fraccionario representado por la cadena de bits original, sino que hemos empaquetado los bits sin tener en cuenta su sentido.

8. Desempaquetad la cadena de bits A83h.

Obtenemos la codificación binaria de cada dígito:

$A_{(16} = 1010_{(2}$
$8_{(16} = 1000_{(2}$
$3_{(16} = 0011_{(2}$

Teniendo en cuenta esta correspondencia, A83h codifica la cadena de bits 101010000011.

a) Encontrad el valor decimal si se trata de un número natural.

Interpretado así, se trata del número binario $101010000011_{(2}$. Encontraremos el valor decimal aplicando el TFN:

$101010000011 = 2^{11} + 2^9 + 2^7 + 2^1 + 2^0 = 2048 + 512 + 128 + 2 + 1 = \mathbf{2691}_{(10}$.

b) Encontrad el valor decimal si se trata de un número en coma fija sin signo de 12 bits, donde 4 son fraccionarios.

Con esta interpretación, se trata del número binario $10101000,0011_{(2}$. Encontraremos el valor decimal aplicando el TFN:

$10101000,0011 = 2^7 + 2^5 + 2^3 + 2^{-3} + 2^{-4} = 128 + 32 + 8 + 0,125 + 0,06125 = \mathbf{168,1865}_{(10}$.

9. Consideremos el número $1010,101_{(2}$.

a) Haced el cambio a base 16.

Para hacer el cambio a base 16 tenemos que hacer agrupaciones de 4 bits a partir de la coma fraccionaria:

Base 2	1010,	101**0**
Base 16	A,	A

Por lo tanto, el número $1010,101_{(2}$ en hexadecimal es A,A (16.

b) Haced el empaquetamiento hexadecimal.

Para hacer el empaquetamiento, no tenemos que tener en cuenta la posición de la coma fraccionaria:

Base 2	**0**101	0,101
Base 16	5	5

Por lo tanto, el número $1010{,}101_{(2}$ se empaqueta en hexadecimal como 55h.

10. Calculad las operaciones siguientes en la base especificada:

a.

$$\begin{array}{rcccccccccl} & 1 & 1 & 1 & 1 & 1 & & & & & \\ & & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & (2 \\ + & & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & (2 \\ \hline & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & (2 \end{array}$$

b.

$$\begin{array}{rcccl} & 2 & 3 & 4 & 5 & (8 \\ + & & 3 & 2 & 1 & (8 \\ \hline & 2 & 6 & 6 & 6 & (8 \end{array}$$

c.

$$\begin{array}{rcccl} & & 1 & & \\ & A & 2 & 3 & F & (16 \\ + & 5 & 4 & A & 3 & (16 \\ \hline & F & 6 & E & 2 & (16 \end{array}$$

d.

$$\begin{array}{rcccccccccl} & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & (2 \\ & & 1 & & & 1 & & & & & \\ - & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & (2 \\ \hline & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & (2 \end{array}$$

e.

$$\begin{array}{rcccl} & 2 & 3 & 4 & 5 & (8 \\ - & & 3 & 2 & 1 & (8 \\ \hline & 2 & 0 & 2 & 4 & (8 \end{array}$$

f.

$$\begin{array}{rcccl} & A & 2 & 3 & F & (16 \\ & 1 & 1 & & & \\ - & 5 & 4 & A & 3 & (16 \\ \hline & 4 & D & 9 & C & (16 \end{array}$$

11. Calculad las operaciones siguientes en la base especificada:

a.

$$\begin{array}{rcccccl} & & 1 & 1 & & & \\ & 6 & 2, & 4 & 8 & & (16 \\ + & 3 & 5, & D & F & & (16 \\ \hline & 9 & 8, & 2 & 7 & & (16 \end{array}$$

b.

$$\begin{array}{rcccccccccccccccccl} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & & & & & & & \\ & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1, & 1 & 1 & 0 & 1 & 1 & & & (2 \\ + & & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0, & 1 & 1 & 1 & & & & & (2 \\ \hline & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0, & 1 & 0 & 1 & 1 & 1 & & & (2 \end{array}$$

c.

	6	2,	4	8	$_{(16}$
	1	1	1		
–	3	5,	D	F	$_{(16}$
	2	C,	6	9	$_{(16}$

d.

	1	1	1	1	0	1	1	0	1,	1	1	0	1	1	$_{(2}$
			1	1					1	1	1				
–	1	0	0	1	1	0	1	0	0,	1	1	1			$_{(2}$
	0	1	0	1	1	1	0	0	0,	1	1	1	1	1	$_{(2}$

12. Calculad las multiplicaciones siguientes:

La multiplicación de un número por b^k, donde b es la base de numeración, equivale a desplazar la coma fraccionaria k posiciones a la derecha.

a) $128{,}7_{(10} \cdot 10^4 = 128{,}7_{(10} \cdot 10000_{(10} = 1287000_{(10}$
b) $AFD_{(16} \cdot 16^2 = AFD_{(16} \cdot 100_{(16} = AFD00_{(16}$
c) $1101{,}01_{(2} \cdot 2^2 = 1101{,}01_{(2} \cdot 100_{(2} = 110101_{(2}$

13. Calculad el cociente y el resto de las divisiones enteras siguientes:

La división de un número por b^k donde b es la base de numeración, equivale a desplazar la coma fraccionaria k posiciones a la izquierda.

a) $52978_{(10} / 10^3 = 52978_{(10} / 1000_{(10} = 52{,}978_{(10}$

El cociente de la división entera es $52_{(10}$. El resto es $978_{(10}$.

b) $3456_{(16} / 16^2 = 3456_{(16} / 100_{(16} = 34{,}56_{(16}$.

El cociente de la división entera es $34_{(16}$. El resto es $56_{(16}$.

c) $100101001001_{(2} / 2^8 = 100101001001_{(2} / 100000000_{(2} = 1001{,}01001001_{(2}$

El cociente de la división entera es $1001_{(2}$. El resto es $01001001_{(2}$.

14. Determinad el rango y la precisión de los formatos de coma fija sin signo $x_1x_0{,}x_{-1}x_{-2}x_{-3}$ y , $x_2x_1x_0{,}x_{-1}x_{-2}$ donde x_i es un dígito decimal.

Dado que la base de numeración es 10, el rango de la representación del formato $x_1x_0{,}x_{-1}x_{-2}x_{-3}$ es $[0, 99{,}999_{(10}]$. La precisión de este formato es $0{,}001_{(10}$ porque esta es la distancia entre dos números consecutivos representables en este formato, como por ejemplo el $12{,}121_{(10}$ y el $12{,}122_{(10}$.

De forma similar, el rango de representación del formato $x_2x_1x_0{,}x_{-1}x_{-2}$ es $[0, 999{,}99_{(10}]$, y su precisión es $0{,}01_{(10}$, la distancia entre dos números consecutivos representables en el formato, como por ejemplo el $45{,}77_{(10}$ y el $45{,}78_{(10}$.

15. Determinad si el número 925,4 se puede representar en los formatos indicados en la actividad 14.

El número $925{,}4_{(10}$ no se puede representar en el formato $x_1x_0{,}x_{-1}x_{-2}x_{-3}$, puesto que este número está fuera del rango de representación. En cambio, se puede representar en el formato $x_2x_1x_0{,}x_{-1}x_{-2}$, dado que se encuentra dentro del rango $[0, 999{,}99_{(10}]$. Además, se puede representar de manera exacta, porque en el formato hay disponibles dígitos fraccionarios suficientes.

16. Representad en el formato de coma fija y sin signo $x_1x_0{,}x_{-1}x_{-2}$, donde x_i es un dígito decimal, los números siguientes:

Para escribir estos números en el formato indicado hay que escribirlos con dos dígitos enteros y dos dígitos fraccionarios:

a) $10_{(10}$	se escribe	10,00
b) $10{,}02_{(10}$	se escribe	10,02
c) $03{,}1_{(10}$	se escribe	03,10
d) $03{,}2_{(10}$	se escribe	03,20

17. Determinad la cantidad de números que se pueden representar en el formato $x_2x_1x_0{,}x_{-1}x_{-2}x_{-3}$, donde x_i es un dígito decimal.

La máxima cantidad de números que se pueden representar en un formato es b^k, donde k es el número de dígitos disponibles en el formato y b la base de numeración. Dado que se trata de un formato decimal, cada dígito puede tomar 10 valores diferentes (0 - 9), y como el formato dispone de 6 dígitos para la representación, podemos representar un total de 10^6 números. Fijémonos que la cantidad de números que se pueden representar no depende de la posición de la coma.

18. Calculad el error de representación que se comete cuando representamos en el formato $x_2x_1x_0,x_{-1}x_{-2}$, donde x_i es un dígito decimal, los números siguientes:

a) $223{,}45_{(10}$
El número $223{,}45_{(10}$ está directamente representado en el formato $x_2x_1x_0,x_{-1}x_{-2}$. Por lo tanto, se trata de un número representable y el error cometido es cero.

b) $45{,}89_{(10}$
El número $45{,}89_{(10}$, es representable directamente en el formato $x_2x_1x_0,x_{-1}x_{-2}$. La representación es $045{,}89_{(10}$ y es exacta. Por lo tanto, el error de representación es cero.

c) $55{,}6356_{(10}$
El número $55{,}6356_{(10}$ no se puede representar directamente en el formato $x_2x_1x_0,x_{-1}x_{-2}$, dado que tiene 4 dígitos fraccionarios. Tendremos que hacer una aproximación, lo que comporta un cierto error de aproximación.

Con una aproximación por truncamiento, la representación será $55{,}635_{(10}$. El error de representación que se comete es $|55{,}6356_{(10} - 55{,}635_{(10}| = 0{,}0006_{(10}$.

Para encontrar la representación una aproximación por redondeo, tenemos que sumar la mitad de la precisión y truncar el resultado a 3 dígitos fraccionarios: $55{,}6356_{(10} + 0{,}001_{(10} = 55{,}6366_{(10}$ que con el truncamiento a 3 dígitos fraccionarios queda $55{,}636_{(10}$. El error de representación cometido es, en este caso, $|55{,}6356_{(10} - 55{,}636_{(10}| = 0{,}0004_{(10}$.

d) $23{,}56_{(10}$
El número $23{,}56_{(10}$, es representable directamente en el formato $x_2x_1x_0,x_{-1}x_{-2}$. La representación es $023{,}56_{(10}$ y es exacta. Por lo tanto, el error de representación es cero.

19. Escoged el formato hexadecimal que use el mínimo número de dígitos y que permita representar el número $16{,}25_{(10}$ de manera exacta. ¿Cuál es el rango y la precisión del formato?

Para representar este número en hexadecimal, primero lo pasaremos a binario. La representación binaria de $16{,}25_{(10}$ es $10000{,}01_{(2}$.

Para la representación hexadecimal de este número hay que añadir ceros a los extremos hasta disponer de grupos de 4 bits completos a ambos lados de la coma decimal. Entonces tenemos:

$$00010000{,}0100_{(2} = 10{,}4_{(16}.$$

- El rango de esta representación es:[0 ($00{,}0_{(16}$), $255{,}9375_{(10}$ ($FF{,}F_{(16}$)].
- Su precisión es: $0{,}0625_{(10} = 00{,}1_{(16} - 00{,}0_{(16}$.

20. ¿Cuál es el número más pequeño que hay que sumar a $8341_{(10}$ para que se produzca desbordamiento en una representación decimal (base 10) de cuatro dígitos?

Para que en una representación decimal de 4 dígitos sin signo se produzca desbordamiento, tenemos que sobrepasar el número $9999_{(10}$; por lo tanto, tendremos desbordamiento cuando la suma de dos números sea $10000_{(10}$. Entonces, el número más pequeño que tenemos que sumar a $8341_{(10}$ es $10000_{(10} - 8341_{(10} = 1659_{(10}$.

21. Convertid los valores decimales siguientes a binarios en los sistemas de representación de signo y magnitud y complemento a 2, con un formato entero de 8 bits:

Pasamos los valores numéricos a binario:

a) 53

53	1
26	0
13	1
6	0
3	1
1	

b) –25

25	1
12	0
6	0
3	1
1	

c) 93

93	1
46	0
23	1
11	1
5	1
2	0
1	

$+53_{(10} = +110101_{(2}$ $\quad -25_{(10} = -11001_{(2}$ $\quad +93_{(10} = +1011101_{(2}$

d) –1

1	1
0	

$-1_{(10} = -1_{(2}$

e) –127

127	1
63	1
31	1
15	1
7	1
3	1
1	

$-127_{(10} = -1111111_{(2}$

f) –64

64	0
32	0
16	0
8	0
4	0
2	0
1	

$-64_{(10} = -1000000_{(2}$

Para obtener la representación en signo y magnitud, tan sólo tenemos que poner el bit de signo y añadir la magnitud expresada en 7 bits:

Base 10	Base 2	Signo y magnitud
$+53_{(10}$	$+110101_{(2}$	$00110101_{(SM2}$
$-25_{(10}$	$-11001_{(2}$	$10011001_{(SM2}$
$+93_{(10}$	$+1011101_{(2}$	$01011101_{(SM2}$
$-1_{(10}$	$-1_{(2}$	$10000001_{(SM2}$
$-127_{(10}$	$-1111111_{(2}$	$11111111_{(SM2}$
$-64_{(10}$	$-1000000_{(2}$	$11000000_{(SM2}$

La representación en Ca2 de las magnitudes positivas coincide con la representación en signo y magnitud. La representación en Ca2 de las magnitudes negativas se puede obtener de varias maneras:

- Se puede hacer la operación $2^8 - |X|$ en base 10, y pasar posteriormente el resultado a binario.
- Podemos hacer la operación $2^8 - |X|$ directamente en base 2.
- Se aplica un cambio de signo a la magnitud positiva en Ca2.

a) El $+53_{(10} = +110101_{(2}$ se representa por $00110101_{(Ca2}$ en Ca2.

b) Podemos obtener la representación en Ca2 del $-25_{(10}$ de la manera siguiente:

- $2^8 - 25 = 256_{(10} - 25_{(10} = 231_{(10} = 11100111_{(Ca2}$, o bien,
- $2^8 - 25 = 100000000_{(2} - 11001_{(2} = 11100111_{(Ca2}$, o bien,
- $+25_{(10} = +11001_{(2} \rightarrow$ Representación de la magnitud positiva $\rightarrow 00011001_{(Ca2} \rightarrow$
  $\rightarrow$ Cambio de signo $\rightarrow 11100111_{(Ca2}$

c) El $+93_{(10} = +1011101_{(2}$ se representa por $01011101_{(Ca2}$ en Ca2.

d) Obtenemos la representación en Ca2 del $-1_{(10}$:

- $2^8 - 1 = 256_{(10} - 1_{(10} = 255_{(10} = 11111111_{(Ca2}$, o bien,
- $2^8 - 1 = 100000000_{(2} - 1_{(2} = 11111111_{(Ca2}$, o bien,
- $+1_{(10} = +1_{(2} \rightarrow$ Representación de la magnitud positiva $\rightarrow 00000001_{(Ca2} \rightarrow$
  $\rightarrow$ Cambio de signo $\rightarrow 11111111_{(Ca2}$

e) La representación en Ca2 del $-127_{(10}$ se puede obtener:

- $2^8 - 127 = 256_{(10} - 127_{(10} = 129_{(10} = 10000001_{(Ca2}$, o bien,
- $2^8 - 127 = 100000000_{(2} - 1111111_{(2} = 10000001_{(Ca2}$, o bien,
- $+127_{(10} = +1111111_{(2} \rightarrow$ Representación de la magnitud positiva $\rightarrow 01111111_{(Ca2} \rightarrow$
  $\rightarrow$ Cambio de signo $\rightarrow 10000001_{(Ca2}$

f) La representación en Ca2 del $-64_{(10}$ se obtiene:

- $2^8 - 64 = 256_{(10} - 64_{(10} = 192_{(10} = 11000000_{(Ca2}$, o bien,
- $2^8 - 64 = 100000000_{(2} - 1000000_{(2} = 11000000_{(Ca2}$, o bien,
- $+64_{(10} = +1000000_{(2} \rightarrow$ Representación de la magnitud positiva $\rightarrow 01000000_{(Ca2} \rightarrow$
  $\rightarrow$ Cambio de signo $\rightarrow 11000000_{(Ca2}$

Decimal	Binario	Complemento a 2	Signo y magnitud
$+53_{(10}$	$+110101_{(2}$	$00110101_{(Ca2}$	$00110101_{(SM2}$
$-25_{(10}$	$-11001_{(2}$	$11100111_{(Ca2}$	$10011001_{(SM2}$
$+93_{(10}$	$+1011101_{(2}$	$01011101_{(Ca2}$	$01011101_{(SM2}$
$-1_{(10}$	$-1_{(2}$	$11111111_{(Ca2}$	$10000001_{(SM2}$
$-127_{(10}$	$-1111111_{(2}$	$10000001_{(Ca2}$	$11111111_{(SM2}$
$-64_{(10}$	$-1000000_{(2}$	$11000000_{(Ca2}$	$11000000_{(SM2}$

22. Si tenemos los números binarios 00110110, 11011010, 01110110, 11111111 y 11100100, ¿cuáles son los equivalentes decimales considerando que son valores binarios representados en signo y magnitud?

Si consideramos que son valores en signo y magnitud, tenemos:

$00110110_{(SM2} = +54_{(10}$
$11011010_{(SM2} = -90_{(10}$
$01110110_{(SM2} = +118_{(10}$
$11111111_{(SM2} = -127_{(10}$
$11100100_{(SM2} = -100_{(10}$

23. Repetid el ejercicio anterior considerando que las cadenas de bits son números en complemento a 2.

Si consideramos que son valores en complemento a 2:

$00110110_{(Ca2} = +54_{(10}$
$11011010_{(Ca2} = -2^7 + 90_{(10} = -38_{(10}$
$01110110_{(Ca2} = +118_{(10}$
$11111111_{(Ca2} = -2^7 + 127_{(10} = -1_{(10}$
$11100100_{(Ca2} = -2^7 + 100_{(10} = -28_{(10}$

24. Si tenemos las cadenas de bits siguientes $A = 1100100111$, $B = 1000011101$ y $C = 0101011011$, haced las operaciones que proponemos a continuación considerando que son números binarios en formato de signo y magnitud: $A + B$, $A - B$, $A + C$, $A - C$, $B - C$, $B + C$.

Para hacer las operaciones de suma y resta en signo y magnitud, tenemos que tener en cuenta el signo de las magnitudes, y actuar consecuentemente. Para lograr más claridad, junto a la operación en binario se ha hecho el equivalente en base 10. Los valores A , B y C en base 10 corresponden a:

$A = 1100100111 = -295_{(10}$
$B = 1000011101 = -29_{(10}$
$C = 0101011011 = +347_{(10}$

- **Suma $A + B$**

A *es* negativo y B también. Por lo tanto, haremos la suma de las magnitudes (sin signo) y si no hay desbordamiento añadiremos un 1 como bit de signo al resultado obtenido, para obtener el resultado en la representación en signo y magnitud:

Suma $A + B$

			1	1	1	1	1	1			1	1		
	1	0	0	1	0	0	1	1	1	(SM2	2	9	5	(10
+	0	0	0	0	1	1	1	0	1	(SM2	+	2	9	(10
	1	0	1	0	0	0	1	0	0	(SM2	3	2	4	(10

$A + B = 1101000100_{(2} = -324_{(10}$

- **Resta *A* – *B***

A es negativo y *B* también. Por lo tanto, restaremos la magnitud pequeña (*B*) de la magnitud grande (*A*) y aplicaremos el signo de la magnitud grande (*A*) al resultado (no se puede producir desbordamiento):

Resta *A* – *B*

	1	0	0	1	0	0	1	1	1	(SM2	2	9	5	(10
				1	1							1		
–	0	0	0	0	1	1	1	0	1	(SM2	–	2	9	(10
	1	0	0	0	0	1	0	1	0	(SM2	2	6	6	(10

$A - B = 1100001010_{(2} = -266_{(10}$

- **Suma *A* + *C***

A es negativo y *C* es positivo. Por lo tanto, restaremos la magnitud pequeña (*A)* de la magnitud grande (*C*) y aplicaremos el signo de la magnitud grande (*C*) al resultado (no se puede producir desbordamiento):

Suma *A* + *C*

	1	0	1	0	1	1	0	1	1	(SM2	3	4	7	(10
			1			1					1			
–	1	0	0	1	0	0	1	1	1	(SM2	– 2	9	5	(10
	0	0	0	1	1	0	1	0	0	(SM2	0	5	2	(10

$A + C = 0000110100_{(2} = +52_{(10}$

- **Resta *A* – *C***

A es negativo y *C* es positivo. Por lo tanto, haremos la suma de las magnitudes (sin signo) y, si no hay desbordamiento, añadiremos un 1 como bit de signo al resultado obtenido, para obtener el resultado en la representación en signo y magnitud:

Resta *A* – *C*

Desbordamiento

↓

1		1	1	1	1	1	1	1			1	1		
	1	0	0	1	0	0	1	1	1	(SM2	2	9	5	(10
+	1	0	1	0	1	1	0	1	1	(SM2	+ 3	4	7	(10
1	0	1	0	0	0	0	0	1	0	(SM2	6	4	2	(10

$A - C = -642_{(10}$
No se puede representar con 10 bits
en el formato de signo y magnitud.

- **Resta *B* – *C***

B es negativo y *C* es positivo. Por lo tanto, haremos la suma de las magnitudes (sin signo) y, si no hay desbordamiento, añadiremos un 1 como bit de signo al resultado obtenido, para obtener el resultado en la representación en signo y magnitud:

Resta *B* – *C*

				1	1	1	1	1				1			
	0	0	0	0	1	1	1	0	1	(SM2		2	9	(10
+	1	0	1	0	1	1	0	1	1	(SM2	+	3	4	7	(10
	1	0	1	1	1	1	0	0	0	(SM2		3	7	6	(10

$B - C = 1101111000_{(2} = -376_{(10}$

- **Suma *B* + *C***

B es negativo y *C* es positivo. Por lo tanto, restaremos la magnitud pequeña (*B*) de la magnitud grande (*C*) y aplicaremos el signo de la magnitud más grande (*C*) al resultado (no se puede producir desbordamiento):

Suma *B* + *C*

	1	0	1	0	1	1	0	1	1	(SM2		3	4	7	(10
			1	1	1	1							1		
–	0	0	0	0	1	1	1	0	1	(SM2	–		2	9	(10
	1	0	0	1	1	1	1	1	0	(SM2		3	1	8	(10

$B + C = 0100111110_{(2} = +318_{(10}$

25. Repetid la actividad anterior considerando que las cadenas representan números en complemento a 2.
Para hacer las operaciones de suma en Ca2, operaremos directamente sobre las representaciones. El resultado será correcto siempre que no se produzca desbordamiento. Para hacer las operaciones de resta, aplicaremos un cambio de signo al sustraendo y haremos una operación de suma:

$$A = 1100100111 = -217_{(10}$$
$$B = 1000011101 = -483_{(10}$$
$$C = 0101011011 = +347_{(10}$$

- **Suma *A* + *B***

Suma *A* + *B*

Desbordamiento
↓

	1				1	1	1	1	1	1			1	1		
		1	1	0	0	1	0	0	1	1	1	(Ca2		– 2	1	7	(10
+		1	0	0	0	0	1	1	1	0	1	(Ca2	+	– 4	8	3	(10
	1	0	1	0	1	0	0	0	1	0	0	(Ca2		– 7	0	0	(10

$A + B = -700_{(10}$

Se produce desbordamiento, dado que al sumar dos números negativos obtenemos uno positivo. El resultado no cabe en el formato de salida. Recordemos que el rango de representación de enteros con 10 bits con el formato de complemento a 2 es $[-2^{10-1}, 2^{10-1} - 1] = [-512, 511]$

- **Resta *A* – *B***

Aplicaremos un cambio de signo a *B* y haremos una operación de suma:

```
1000011101
          ^ Se mantienen los bits hasta aquí (el primer 1 que encontramos, incluyendo éste)
          1
         ^ Se complementan los bits a partir de este punto
011110001
```

Al aplicar un cambio de signo al 1000011101 obtenemos el 0111100011, que será el valor que utilizaremos en la suma:

Suma *A* + (–*B*)

Transporte que se ignora

↓

	1	1	1	1	1			1	1	1					1		
		1	1	0	0	1	0	0	1	1	1	(Ca2		– 2	1	7	(10
+		0	1	1	1	1	0	0	0	1	1	(Ca2	+	+ 4	8	3	(10
	1	0	1	0	0	0	0	1	0	1	0	(Ca2		+ 2	6	6	(10

$A - B = 0100001010 = +266_{(10}$

- **Suma *A* + *C***

A es negativo y *C* es positivo. No se puede producir desbordamiento:

Suma *A* + *C*

Transporte que se ignora

↓

	1	1		1	1	1	1	1	1	1							
		1	1	0	0	1	0	0	1	1	1	(Ca2		– 2	1	7	(10
+		0	1	0	1	0	1	1	0	1	1	(Ca2	+	+ 3	4	7	(10
	1	0	0	1	0	0	0	0	0	1	0	(Ca2		+ 1	3	0	(10

$A + C = 0010000010 = +130_{(10}$

- **Resta *A* – *C***

Aplicaremos un cambio de signo a *C* y haremos una operación de suma:

```
0101011011      ←    Valor numérico inicial
1010100100      ←    Complemento bit a bit de la expresión inicial
+          1    ←    Sumamos 1 al bit menos significativo del formato
1010100101
```

Al aplicar un cambio de signo al 0101011011 obtenemos el 1010100101, que será el valor que utilizaremos en la suma:

Suma $A + (-C)$

Desbordamiento

↓

	1				1			1	1	1					1		
		1	1	0	0	1	0	0	1	1	1	(Ca2		−2	1	7	(10
+		1	0	1	0	1	0	0	1	0	1	(Ca2	+	−3	4	7	(10
	1	0	1	1	1	0	0	1	1	0	0	(Ca2		−5	6	4	(10

$A - C = -564_{(10}$

Se produce desbordamiento, dado que al sumar dos números negativos, obtenemos uno positivo. El resultado no cabe en el formato de salida. Recordemos que el rango de representación de enteros con 10 bits con el formato de complemento a 2 es $[-2^{10-1}, 2^{10-1} - 1] = [-512, 511]$.

- **Resta $B - C$**

Aplicaremos un cambio de signo a C y haremos una operación de suma. Al aplicar un cambio de signo al 0101011011 obtenemos el 1010100101, que será el valor que utilizaremos en la suma:

Suma $B + (-C)$

Desbordamiento

↓

	1				1	1	1	1		1				1	1		
		1	0	0	0	0	1	1	1	0	1	(Ca2		−4	8	3	(10
+		1	0	1	0	1	0	0	1	0	1	(Ca2	+	−3	4	7	(10
	1	0	0	1	1	0	0	0	0	1	0	(Ca2		−8	3	0	(10

$B - C = -830_{(10}$

Se produce desbordamiento, dado que al sumar dos números negativos, obtenemos uno positivo. El resultado no cabe en el formato de salida.

- **Suma $B + C$**

B es negativo y C es positivo. No se puede producir desbordamiento:

Suma $B + C$

					1	1	1	1	1					1			
	1	0	0	0	0	1	1	1	0	1	(Ca2		−4	8	3	(10
+	0	1	0	1	0	1	1	0	1	1	(Ca2	+	+3	4	7	(10
	1	1	0	1	1	1	1	0	0	0	(Ca2		−1	3	6	(10

$B + C = 1101111000 = -136_{(10}$

26. Si tenemos la cadena de bits 10110101, haced las conversiones siguientes:

a) Considerando que representa un número en Ca2, representad el mismo número en signo y magnitud y 16 bits:

Si la cadena de bits 10110101 representa un número en Ca2, se trata de un número negativo, puesto que el primer bit es 1. Podemos hacer un cambio de signo para obtener la magnitud positiva:

	1 0 1 1 0 1 0 1	← Valor numérico inicial
	0 1 0 0 1 0 1 0	← Complemento bit a bit de la expresión inicial
+	1	← Sumamos 1 al bit menos significativo del formato
	0 1 0 0 1 0 1 1	

Para codificar el valor inicial en signo y magnitud, sólo tenemos que cambiar el bit de signo del valor conseguido con el cambio de signo. Por lo tanto, la codificación en signo y magnitud del valor 10110101 que está en Ca2 es $\mathbf{11001011}_{\mathbf{(SM2}}$.

La codificación en 16 bits la podemos obtener haciendo la extensión del formato, que en este caso se consigue añadiendo los ceros necesarios a la derecha del signo:

$$11001011_{(SM2} = \mathbf{1000000001001011}_{\mathbf{(SM2}}$$

b) Considerando que representa un número en signo y magnitud, representad el mismo número en Ca2 y 16 bits:

Si se trata de un número codificado en signo y magnitud, es un número negativo, puesto que el primer dígito corresponde al signo y es 1. El resto de dígitos codifican la magnitud en binario. Podemos obtener la magnitud positiva cambiando el bit de signo: $00110101_{(2}$.

La representación de los valores positivos coincide en Ca2 y en signo y magnitud. Por lo tanto, podemos interpretar que tenemos la magnitud positiva en Ca2 y que podemos conseguir la magnitud negativa en Ca2 aplicando un cambio de signo:

	0 0 1 1 0 1 0 1	← Valor numérico inicial
	1 1 0 0 1 0 1 0	← Complemento bit a bit de la expresión inicial
+	1	← Sumamos 1 al bit menos significativo del formato
	1 1 0 0 1 0 1 1	

Por lo tanto, la codificación en Ca2 del valor $10110101_{(SM2}$ que está en signo y magnitud es $\mathbf{11001011}_{\mathbf{(Ca2}}$.

La codificación en 16 bits la podemos obtener haciendo la extensión del formato, que en este caso se consigue copiando el bit de signo a la izquierda de la codificación tantas veces como sea necesario:

$$11001011_{(Ca2} = \mathbf{1111111111001011}_{\mathbf{(Ca2}}$$

27. Determinad qué valor decimal codifica la cadena de bits 1010010 en los supuestos siguientes:

a) Si se trata de un número en coma fija sin signo de 7 bits donde 4 son fraccionarios.

Con este formato, el número binario que representa esta tira de bits es $101{,}0010_{(2}$ y, al aplicar el TFN, se obtiene el número decimal que representa:

$$\begin{aligned} 101{,}0010_{(2} &= 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} \\ &= 2^2 + 2^0 + 2^{-3} \\ &= 4 + 1 + 0{,}125 \\ &= \mathbf{5{,}125}_{\mathbf{(10}} \end{aligned}$$

b) Si se trata de un número en coma fija sin signo de 7 bits donde 1 es fraccionario.

En este caso, el número binario que representa esta tira de bits es $101001{,}0_{(2}$ y, al aplicar el TFN, se obtiene el número decimal que representa:

$$\begin{aligned} 101001{,}0_{(2} &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} \\ &= 2^5 + 2^3 + 2^0 \\ &= 32 + 8 + 1 \\ &= \mathbf{41}_{(10} \end{aligned}$$

28. Codificad los números $+12{,}85_{(10}$, $+0{,}7578125_{(10}$ y $-11{,}025_{(10}$ en una representación fraccionaria binaria en signo y magnitud de 8 bits donde 3 son fraccionarios. Usad una aproximación por redondeo en caso de que sea necesario.

Codificamos el número $+12{,}85_{(10}$ en el formato especificado. Como se trata de un número positivo, el bit de signo es **0**. En cuanto a la magnitud $12{,}85_{(10}$, primero pasamos a binario la parte entera y, posteriormente, la parte fraccionaria. Para la parte entera usamos el algoritmo de divisiones sucesivas por la base de llegada (2):

$$\begin{array}{rcll} 12 & = & 6 \cdot 2 & + \mathbf{0} \\ 6 & = & 3 \cdot 2 & + \mathbf{0} \\ 3 & = & 1 \cdot 2 & + \mathbf{1} \\ 1 & = & 0 \cdot 2 & + \mathbf{1} \end{array} \uparrow$$

Así pues, $12_{(10} = 1100_{(2}$. En cuanto a la parte fraccionaria, hacemos multiplicaciones sucesivas por la base de llegada (2):

$$\begin{array}{lclcl} 0{,}85 \cdot 2 & = & 1{,}70 & = & \mathbf{1} + 0{,}70 \\ 0{,}70 \cdot 2 & = & 1{,}40 & = & \mathbf{1} + 0{,}40 \\ 0{,}40 \cdot 2 & = & 0{,}80 & = & \mathbf{0} + 0{,}80 \\ 0{,}80 \cdot 2 & = & 1{,}60 & = & \mathbf{1} + 0{,}60 \\ 0{,}60 \cdot 2 & = & 1{,}20 & = & \mathbf{1} + 0{,}20 \end{array} \downarrow$$

Como el formato especificado sólo tiene 3 bits para la parte fraccionaria, ya tenemos más bits de los necesarios y podemos parar el proceso aquí. Por lo tanto, $0{,}85_{(10} = 0{,}11011\ldots_{(2}$. Para aproximar el valor con 3 bits y redondeo, procedemos como sigue:

$$0{,}11011_{(2} + 0{,}0001_{(2} = 0{,}11101_{(2} \rightarrow \text{truncamos 3 bits} \rightarrow 0{,}111_{(2}$$

A continuación, juntamos las partes entera y fraccionaria y obtenemos: $12{,}85_{(10} \approx 1100{,}111_{(2}$. Finalmente, para obtener la representación en el formato indicado, hay que añadir el bit de signo, de forma que la tira de bits que representa este número en el formato dado es: $\mathbf{0}1100{,}111_{(2}$. Hay que recordar que la tira de bits que se almacenaría en un computador no contiene la coma ni la especificación de la base: **01100111**.

En los otros dos casos, procedemos de manera totalmente análoga:

$+0{,}7578125_{(10}$:

- El bit de signo es 0, porque el número es positivo.
- La parte entera es $0_{(10} = 0_{(2}$. Como tenemos 4 bits para la parte entera, escribiremos $0000_{(2}$.
- En cuanto a la parte fraccionaria:

$$\begin{array}{lclcl} 0{,}7578125 \cdot 2 & = & 1{,}515625 & = & \mathbf{1} + 0{,}515625 \\ 0{,}515625 \cdot 2 & = & 1{,}03125 & = & \mathbf{1} + 0{,}03125 \\ 0{,}03125 \cdot 2 & = & 0{,}0625 & = & \mathbf{0} + 0{,}0625 \\ 0{,}0625 \cdot 2 & = & 0{,}125 & = & \mathbf{0} + 0{,}125 \\ 0{,}1250 \cdot 2 & = & 0{,}25 & = & \mathbf{0} + 0{,}25 \end{array} \downarrow$$

Así, $0{,}7578125_{(10} = 0{,}11000\ldots_{(2}$ que, redondeando con 3 bits:

$$0{,}11000_{(2} + 0{,}0001_{(2} = 0{,}11010_{(2} \rightarrow \text{truncamos 3 bits} \rightarrow 0{,}110_{(2}$$

Finalmente, se juntan todas las partes y obtenemos $0{,}7578125_{(10} \approx \mathbf{0}0000{,}110_{(2}$. Por lo tanto, la tira de bits que se almacenaría en un computador es **00000110**.

$-11{,}025_{(10}$:

- El bit de signo es 1 porque el número es negativo.

- La parte entera es $11_{(10}$:

$$\begin{array}{rcll} 11 & = & 5 \cdot 2 & + \mathbf{1} \\ 5 & = & 2 \cdot 2 & + \mathbf{1} \\ 2 & = & 1 \cdot 2 & + \mathbf{0} \\ 1 & = & 0 \cdot 2 & + \mathbf{1} \end{array} \uparrow$$

Es decir$11_{(10} = 1011_{(2}$.

- En cuanto a la parte fraccionaria:

$$\begin{array}{lclclcl} 0{,}025 \cdot 2 & = & 0{,}05 & = & \mathbf{0} + 0{,}05 \\ 0{,}05 \cdot 2 & = & 0{,}1 & = & \mathbf{0} + 0{,}1 \\ 0{,}1 \cdot 2 & = & 0{,}2 & = & \mathbf{0} + 0{,}2 \\ 0{,}2 \cdot 2 & = & 0{,}4 & = & \mathbf{0} + 0{,}4 \\ 0{,}4 \cdot 2 & = & 0{,}8 & = & \mathbf{0} + 0{,}8 \end{array} \downarrow$$

Así, $0{,}025_{(10} = 0{,}00000\ldots_{(2}$ que, redondeando con 3 bits:

$$0{,}00000_{(2} + 0{,}0001_{(2} = 0{,}00010_{(2} \rightarrow \text{truncamos 3 bits} \rightarrow 0{,}000_{(2}$$

Finalmente, se juntan todas las partes y obtenemos $-11{,}025_{(10} \approx \mathbf{1}1011{,}000_{(2}$. Por lo tanto, la tira de bits que se almacenaría en un computador es **11011000**.

29. Si tenemos una representación en coma fija en signo y magnitud binaria de 8 bits, donde 3 bits son fraccionarios, determinad los números codificados por las cadenas de bits 01001111, 11001111, 01010100, 00000000 y 10000000.

Para obtener la representación decimal de estas cadenas de bits, sólo hay que aplicar el TFN considerando que los tres últimos bits constituyen la parte fraccionaria y que el primer bit representa el signo:

$01001111 \rightarrow +1001{,}111_{(2}$

$$\begin{aligned} +1001{,}111_{(2} &= 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ &= 2^3 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} \\ &= 8 + 1 + 0{,}5 + 0{,}25 + 0{,}125 \\ &= \mathbf{+9{,}875_{(10}} \end{aligned}$$

$11001111 \rightarrow -1001{,}111_{(2}$

$$\begin{aligned} -1001{,}111_{(2} &= -(1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}) \\ &= -(2^3 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3}) \\ &= -(8 + 1 + 0{,}5 + 0{,}25 + 0{,}125) \\ &= \mathbf{-9{,}875_{(10}} \end{aligned}$$

$01010100 \rightarrow +1010{,}100_{(2}$

$$\begin{aligned} +1010{,}100_{(2} &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} \\ &= 2^3 + 2^1 + 2^{-1} \\ &= 8 + 2 + 0{,}5 \\ &= \mathbf{+10{,}5_{(10}} \end{aligned}$$

$00000000 \rightarrow +0000{,}000_{(2}$

$$\begin{aligned} +0000{,}000_{(2} &= 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} \\ &= \mathbf{+0_{(10}} \end{aligned}$$

$10000000 \rightarrow -0000{,}000_{(2}$

$$\begin{aligned} -0000{,}000_{(2} &= -(0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3}) \\ &= \mathbf{-0_{(10}} \end{aligned}$$

30. Si las cadenas de bits 00101010, 11010010 y 10100010 representan números en coma fija sin signo de 8 bits, donde 3 son fraccionarios, representadlos en un formato de coma fija sin signo de 12 bits donde 4 son fraccionarios.

Para extender el rango de estas tiras de bits hay que añadir ceros tanto a la izquierda como a la derecha de la magnitud, dado que se trata de magnitudes sin signo.

La representación original tiene 5 bits para la parte entera y la representación de llegada tiene 8. Por lo tanto, hay que añadir 8 – 5 = 3 bits a la izquierda de la magnitud. Del mismo modo, la representación original tiene 3 bits para la parte fraccionaria, mientras que la representación de llegada tiene 4. Así pues, hay que añadir 4 – 3 = 1 bits a la derecha de la magnitud:

00101010 → 00101,010 → **000**00101,010**0** → 000001010100
11010010 → 11010,010 → **000**11010,010**0** → 000110100100
10100010 → 10100,010 → **000**10100,010**0** → 000101000100

31. Repetid la actividad anterior considerando que se trata de números en signo y magnitud.

En este caso, hay que considerar que el primer bit representa el signo. La extensión de la parte entera implica replicar el bit de signo e insertar tantos ceros como se convenga. En este caso pasamos de 4 a 7 bits para la parte entera, por lo tanto, habrá que añadir 7 – 4 = 3 bits a la izquierda de la parte entera. En cuanto a la parte fraccionaria, la situación es idéntica al ejercicio anterior. Así pues, habrá que añadir 4 – 3 = 1 bit a la derecha de la magnitud:

00101010 → +0101,010 → **+000**0101,010**0** → 000001010100
11010010 → –1010,010 → **–000**1010,010**0** → 100010100100
10100010 → –0100,010 → **–000**0100,010**0** → 100001000100

32. Determinad el rango de representación y la precisión en los formatos siguientes:

a) Coma fija en signo y magnitud con 8 bits, donde 3 son fraccionarios.

El rango de una representación fraccionaria con signo y magnitud es:

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

donde n es el número de bits empleado en la representación y m el número dígitos fraccionarios. Por lo tanto, el rango en este caso se obtiene haciendo $n = 8$ y $m = 3$, es decir:

$$[-2^{8-3-1} + 2^{-3}, +2^{8-3-1} - 2^{-3}] = [-2^4 + 2^{-3}, +2^4 - 2^{-3}] = \mathbf{[-15{,}875_{(10}, +15{,}875_{(10}]}$$

En cuanto a la precisión, ésta viene dada por el bit de menor peso de la magnitud, concretamente es igual a $2^{-m} = 2^{-3} = \mathbf{0{,}125_{(10}}$.

b) Coma fija en signo y magnitud con 8 bits, donde 4 son fraccionarios.

En este caso, la situación es la misma que en el apartado anterior pero con $n = 8$ y $m = 4$. El rango es el siguiente:

$$[-2^{8-4-1} + 2^{-4}, +2^{8-4-1} - 2^{-4}] = [-2^3 + 2^{-4}, +2^3 - 2^{-4}] = \mathbf{[-7{,}9375_{(10}, +7{,}9375_{(10}]}$$

Análogamente, la precisión es igual a $2^{-m} = 2^{-4} = \mathbf{0{,}0625_{(10}}$.

c) Coma fija sin signo con 8 bits, donde 3 son fraccionarios.

En coma fija sin signo, el rango de representación viene dado por:

$$[0, +2^{n-m} - 2^{-m}]$$

donde n es el número de bits empleado en la representación y m el número dígitos fraccionarios. Por lo tanto, el rango en este caso se obtiene haciendo $n = 8$ y $m = 3$, es decir:

$$[0, +2^{8-3} - 2^{-3}] = [0, +2^5 - 2^{-3}] = \mathbf{[0, +31{,}875_{(10}]}$$

La precisión es la misma que en los casos anteriores, porque es independiente de si la representación es sin signo o con signo y magnitud. Por lo tanto, la precisión es $2^{-m} = 2^{-3} = 0{,}125$.

d) Coma fija sin signo con 8 bits, donde 4 son fraccionarios.

En este caso, la situación es la misma que en el apartado anterior pero con $n = 8$ y $m = 4$. El rango es el siguiente:

$$[0, +2^{8-4} - 2^{-4}] = [0, +2^4 - 2^{-4}] = \mathbf{[0, +15{,}9375_{(10}]}$$

Análogamente, la precisión es $2^{-m} = 2^{-4} = \mathbf{0{,}0625_{(10}}$.

33. Determinad la precisión necesaria para poder representar el número $+0{,}1875_{(10}$ de forma exacta (sin error de representación) con un formato de coma fija en base 2.

Para determinar la precisión necesaria para representar el número $+0{,}1875_{(10}$ de forma exacta, primero codificaremos en binario este número haciendo multiplicaciones sucesivas por la base de llegada:

$$\begin{array}{lclcl} 0{,}1875 \cdot 2 & = & 0{,}375 & = & \mathbf{0} + 0{,}375 \\ 0{,}375 \cdot 2 & = & 0{,}75 & = & \mathbf{0} + 0{,}75 \\ 0{,}75 \cdot 2 & = & 1{,}5 & = & \mathbf{1} + 0{,}5 \\ 0{,}5 \cdot 2 & = & 1 & = & \mathbf{1} + 0 \end{array} \quad \downarrow$$

Por lo tanto, $0{,}1875_{(10}=0{,}0011_{(2}$. Así pues, para poder representar este número de forma exacta, es necesario que la representación disponga, como mínimo, de 4 dígitos fraccionarios ($m = 4$) y, por lo tanto, una precisión de la forma $2^{-m} = 2^{-4} = \mathbf{0{,}0625_{(10}}$.

34. Determinad las características de rango y precisión, así como el número de dígitos enteros y fraccionarios necesarios en un formato de coma fija en signo y magnitud, para poder representar de forma exacta los números $+31{,}875_{(10}$ y $-16{,}21875_{(10}$.

En primer lugar, obtendremos la codificación binaria de estos dos números.

Por el $+31{,}875_{(10}$:

- El bit de signo es 0, porque el número es positivo.
- La parte entera es $31_{(10}$:

$$\begin{array}{lclcl} 31 & = & 15 \cdot 2 & + & \mathbf{1} \\ 15 & = & 7 \cdot 2 & + & \mathbf{1} \\ 7 & = & 3 \cdot 2 & + & \mathbf{1} \\ 3 & = & 1 \cdot 2 & + & \mathbf{1} \\ 1 & = & 0 \cdot 2 & + & \mathbf{1} \end{array} \quad \uparrow$$

Es decir $31_{(10} = 11111_{(2}$.

- En cuanto a la parte fraccionaria:

$$\begin{array}{lclcl} 0{,}875 \cdot 2 & = & 1{,}75 & = & \mathbf{1} + 0{,}75 \\ 0{,}75 \cdot 2 & = & 1{,}5 & = & \mathbf{1} + 0{,}5 \\ 0{,}5 \cdot 2 & = & 1 & = & \mathbf{1} + 0 \end{array} \quad \downarrow$$

Así pues, $0{,}875_{(10} = 0{,}111_{(2}$

- Finalmente, se juntan todas las partes y obtenemos $+31{,}875_{(10} = \mathbf{0}11111{,}111_{(SM2}$.

Por el $-16{,}21875_{(10}$:

- El bit de signo es 1, porque el número es negativo.
- La parte entera es $16_{(10}$:

$$\begin{array}{lclcl} 16 & = & 8 \cdot 2 & + & \mathbf{0} \\ 8 & = & 4 \cdot 2 & + & \mathbf{0} \\ 4 & = & 2 \cdot 2 & + & \mathbf{0} \\ 2 & = & 1 \cdot 2 & + & \mathbf{0} \\ 1 & = & 0 \cdot 2 & + & \mathbf{1} \end{array} \quad \uparrow$$

Es decir $16_{(10} = 10000_{(2}$.

- En cuanto a la parte fraccionaria:

$$\begin{array}{lclcl} 0{,}21875 \cdot 2 & = & 0{,}4375 & = & \mathbf{0} + 0{,}4375 \\ 0{,}4375 \cdot 2 & = & 0{,}875 & = & \mathbf{0} + 0{,}875 \\ 0{,}875 \cdot 2 & = & 1{,}75 & = & \mathbf{1} + 0{,}75 \\ 0{,}75 \cdot 2 & = & 1{,}5 & = & \mathbf{1} + 0{,}5 \\ 0{,}5 \cdot 2 & = & 1 & = & \mathbf{1} + 0 \end{array} \quad \downarrow$$

Así pues, $0{,}21875_{(10} = 0{,}00111_{(2}$

- Finalmente, se juntan todas las partes y obtenemos $-16{,}21875_{(10} = \mathbf{1}10000{,}00111_{(SM2}$.

Para representar exactamente **los dos** números hace falta un formato que tenga:

- Un bit de signo.

- 5 bits para la parte entera (con 5 bits se pueden representar exactamente tanto $16_{(10}$ como $31_{(10)}$).
- 5 bits para la parte fraccionaria (con 5 bits se pueden representar exactamente tanto $0{,}875_{(10}$ como $0{,}21875_{(10)}$).

Tenemos, pues, un formato de signo y magnitud con $n = 11$ (bit de signo, 5 bits de parte entera y 5 bits de parte fraccionaria) y m = 5. Con estos datos, el rango es el siguiente: $[-2^{n-m-1} + 2^{-m}, +2^{n-m-1} - 2^{-m}] = [-2^{11-5-1} + 2^{-5}, +2^{11-5-1} - 2^{-5}] = [-2^5 + 2^{-5}, +2^5 - 2^{-5}] =$ $=$**$[-31{,}96875_{(10}, +31{,}96875_{(10}]$**. La precisión es $2^{-m} = 2^{-5} =$ **$0{,}03125_{(10}$**.

35. Calculad la suma y la resta de los pares de números siguientes, asumiendo que están en coma fija en signo y magnitud con 8 bits, donde 3 son fraccionarios. Verificad si el resultado es correcto:

a) $00111000_{(2}$ y $10100000_{(2}$

$00111000_{(2} \rightarrow +0111{,}000_{(2} = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = +7_{(10}$
$10100000_{(2} \rightarrow -0100{,}000_{(2} = -1 \cdot 2^2 = -4_{(10}$

Para calcular la suma de estos dos números, hace falta, primero, darse cuenta de que se trata de números de diferente signo. Por lo tanto, la suma se obtendrá restando la magnitud más grande ($0111{,}000_{(2}$) de la más pequeña ($0100{,}000_{(2}$) y copiando el signo de la magnitud más grande (positivo):

$$
\begin{array}{rcccccccccl}
 & 0 & 1 & 1 & 1 & , & 0 & 0 & 0 & {}_{(2} & \\
 & & {\scriptstyle 0} & {\scriptstyle 0} & {\scriptstyle 0} & & {\scriptstyle 0} & {\scriptstyle 0} & & & \leftarrow \text{acarreo} \\
- & 0 & 1 & 0 & 0 & , & 0 & 0 & 0 & {}_{(2} & \\
\hline
 & 0 & 0 & 1 & 1 & , & 0 & 0 & 0 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
$$

Por lo tanto, $00111000_{(2} + 10100000_{(2} \rightarrow +0111{,}000_{(2} + (-0100{,}000_{(2}) = +0011{,}000_{(2} \rightarrow$ **$00011000_{(2}$**. Si convertimos el resultado a base 10, tenemos que $+0011{,}000_{(2} = 1 \cdot 2^1 + 1 \cdot 2^0 = 2 + 1 = 3_{(10}$, cosa que demuestra que el resultado es correcto, puesto que $+7_{(10} + (-4_{(10}) = 3_{(10}$.

Para calcular la resta habrá que hacer la suma de las magnitudes, porque el número $10100000_{(2}$ es negativo. El resultado de la resta será positivo (porque a un número positivo le restamos uno negativo):

$$
\begin{array}{rcccccccccl}
 & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} & {\scriptstyle 0} & & {\scriptstyle 0} & {\scriptstyle 0} & & & \leftarrow \text{acarreo} \\
 & 0 & 1 & 1 & 1 & , & 0 & 0 & 0 & {}_{(2} & \\
+ & 0 & 1 & 0 & 0 & , & 0 & 0 & 0 & {}_{(2} & \\
\hline
 & 1 & 0 & 1 & 1 & , & 0 & 0 & 0 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
$$

Por lo tanto, $00111000_{(2} - 10100000_{(2} \rightarrow +0111{,}000_{(2} - (-0100{,}000_{(2}) = +1011{,}000_{(2} \rightarrow$ **$01011000_{(2}$**. Nuevamente, podemos comprobar que el resultado obtenido es correcto si lo convertimos a base 10: $+1011{,}000_{(2} = 2^3 + 2^1 + 2^0 = 8 + 2 + 1 = +11_{(10} = +7_{(10} - (-4_{(10})$.

b) $10111010_{(2}$ y $11101100_{(2}$

$10111010_{(2} \rightarrow -0111{,}010_{(2} = -(2^2 + 2^1 + 2^0 + 2^{-2}) = -(4 + 2 + 1 + 0{,}25) = -7{,}25_{(10}$
$11101100_{(2} \rightarrow -1101{,}100_{(2} = -(2^3 + 2^2 + 2^0 + 2^{-1}) = -(8 + 4 + 1 + 0{,}5) = -13{,}5_{(10}$

En este caso se trata de dos números negativos, dado que el primer dígito, que indica el signo, es un 1 en ambos casos. Para hacer la suma, habrá que hacer la suma de las magnitudes y copiar el bit signo:

$$
\begin{array}{rcccccccccl}
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 0} & & {\scriptstyle 0} & {\scriptstyle 0} & & & \leftarrow \text{acarreo} \\
 & 0 & 1 & 1 & 1 & , & 0 & 1 & 0 & {}_{(2} & \\
+ & 1 & 1 & 0 & 1 & , & 1 & 0 & 0 & {}_{(2} & \\
\hline
\mathbf{1} & 0 & 1 & 0 & 0 & , & 1 & 1 & 0 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
$$

Notad que se ha producido acarreo en el último bit y, por lo tanto, como estamos sumando sólo las magnitudes, hay desbordamiento. Esto quiere decir que el resultado de la suma no es representable con 8 bits, de los cuales 3 son fraccionarios.

Si dispusiéramos de un bit más para la parte entera (un formato de 9 bits con 3 de ellos fraccionarios) el resultado sí que sería representable: tendríamos $-10100{,}110_{(2} \rightarrow 110100110$. Efectivamente, comprobamos que
$-10100{,}110_{(2} = -(2^4 + 2^2 + 2^{-1} + 2^{-2}) = -(16 + 4 + 0{,}5 + 0{,}25) = -20{,}75_{(10} = -7{,}25_{(10} + (-13{,}5_{(10})$.

En cuanto a la resta, hay que darse cuenta de que el sustraendo ($-1101{,}100_{(2}$) es más grande en magnitud que el minuendo ($-0111{,}010_{(2}$). Así pues, haremos la resta de las magnitudes como $1101{,}100_{(2} - 0111{,}010_{(2}$ y el resultado tiene que ser positivo:

$$
\begin{array}{rcccccccccl}
 & 1 & 1 & 0 & 1 & , & 1 & 0 & 0 & {}_{(2} & \\
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 0} & & {\scriptstyle 1} & {\scriptstyle 0} & & & \leftarrow \text{acarreo} \\
- & 0 & 1 & 1 & 1 & , & 0 & 1 & 0 & {}_{(2} & \\
\hline
 & 0 & 1 & 1 & 0 & , & 0 & 1 & 0 & {}_{(2} & \leftarrow \text{resultado}
\end{array}
$$

Por lo tanto, $10111{,}010_{(2} - 11101100_{(2} \rightarrow -0111{,}010_{(2} - (-1101100_{(2}) = +0110{,}010_{(2} \rightarrow$ $\mathbf{0110010_{(2}}$. Nuevamente, podemos comprobar que el resultado obtenido es correcto si lo convertimos a base 10:
$+0110{,}010_{(2} = 2^2 + 2^1 + 2^{-2} = 4 + 2 + 0{,}25 = +6{,}25_{(10} = -7{,}25_{(10} - (-13{,}5_{(10}) = -7{,}25_{(10} + 13{,}5_{(10}$

36. Codificad en BCD el número $125_{(10}$

En primer lugar codificamos en binario con 4 bits cada uno de los dígitos del número:

$1_{(10} = 0001_{(2}$
$2_{(10} = 0010_{(2}$
$5_{(10} = 0101_{(2}$

Y, finalmente, formamos una única tira con todos los bits $\mathbf{000100100101_{(2}}$.

37. Codificad en BCD el número $637_{(10}$

Como en el apartado anterior, primeramente codificamos en binario con 4 bits cada uno de los dígitos del número:

$6_{(10} = 0110_{(2}$
$3_{(10} = 0011_{(2}$
$7_{(10} = 0111_{(2}$

Y, finalmente, formamos una única tira con todos los bits $\mathbf{011000110111_{(2}}$.

38. Indicad qué número codifica la representación BCD siguiente 00010011100.

Separamos la tira de bits en grupos de 4 y comprobamos qué número decimal representan. Tenemos $000100111000_{(2} \rightarrow 0001 \mid 0011 \mid 1000$, por lo tanto:

$0001_{(2} = 1_{(10}$
$0011_{(2} = 3_{(10}$
$1000_{(2} = 8_{(10}$

Y, finalmente, construimos el número decimal juntando los dígitos decimales: $\mathbf{138_{(10}}$.

39. Codificad el número $427_{(10}$ en BCD y en binario. Comparad el número de bits necesario en los dos casos.

Para codificar $427_{(10}$ en BCD, primeramente codificamos en binario con 4 bits cada uno de los dígitos del número:

$4_{(10} = 0100_{(2}$
$2_{(10} = 0010_{(2}$
$7_{(10} = 0111_{(2}$

Y, finalmente, formamos una única tira con todos los bits $\mathbf{010000100111_{(2}}$.

Para codificar el mismo número en binario, hay que aplicar el método de las divisiones sucesivas por la base de llegada (2):

$$\begin{array}{rcrcrcr} 427 & = & 213 & \cdot & 2 & + & \mathbf{1} \\ 213 & = & 106 & \cdot & 2 & + & \mathbf{1} \\ 106 & = & 53 & \cdot & 2 & + & \mathbf{0} \\ 53 & = & 26 & \cdot & 2 & + & \mathbf{1} \\ 26 & = & 13 & \cdot & 2 & + & \mathbf{0} \\ 13 & = & 6 & \cdot & 2 & + & \mathbf{1} \\ 6 & = & 3 & \cdot & 2 & + & \mathbf{0} \\ 3 & = & 1 & \cdot & 2 & + & \mathbf{1} \\ 1 & = & 0 & \cdot & 2 & + & \mathbf{1} \end{array} \uparrow$$

Por lo tanto, $427_{(10} = \mathbf{110101011}_{(2}$.

En el caso de la codificación BCD, hacen falta **12 bits** para representar el número $427_{(10}$, en cambio, la codificación binaria sólo necesita **9**.

40. Encontrad el valor decimal que codifican las cadenas de bits siguientes, interpretando que se trata de números en un formato de coma flotante de 8 bits con mantisa normalizada de la forma 1,*X* y con bit implícito:

a) 11110010, donde la mantisa es de 4 bits.

En primer lugar hay que identificar los diferentes elementos que componen este formato:

Signo (*S*)	Exponente (*e*)	Mantisa
1	111	0010

- Bit de signo $s = 1$, por lo tanto, se trata de un número negativo.
- Exponente: como la mantisa es de 4 bits, quedan 3 bits ($111_{(2}$) para el exponente ($q = 3$). Por lo tanto, el exceso será $M = 2^{q-1} = 2^2 = 4_{(10}$. Entonces, el exponente vale $e = 111_{(2} - 4_{(10} = (2^2 + 2^1 + 2^0) - 4 = (4 + 2 + 1) - 4 = 7 - 4 = 3_{(10}$.
- Finalmente, los cuatro bits que quedan forman la mantisa que, como está normalizada en la forma 1,*X* con bit implícito toma el valor $R = 1{,}0010_{(2}$.

Ahora ya podemos calcular el número representado en base 10 aplicando la fórmula $\pm R \cdot 2^e$ y el TFN:

$$-1{,}0010_{(2} \cdot 2^3 = -(2^0 + 2^{-3}) \cdot 2^3 = -(2^3 + 2^0) = -(8 + 1) = \mathbf{-9}_{(10}.$$

b) 01010011, donde la mantisa es de 3 bits.

En este caso, procedemos de manera totalmente análoga al apartado anterior:

Signo (*S*)	Exponente (*e*)	Mantissa
0	1010	011

- Bit de signo $s = 0$, por lo tanto, se trata de un número positivo.
- Exponente: como la mantisa es de 3 bits, quedan 4 bits ($1010_{(2}$) para el exponente ($q = 4$). Por lo tanto, el exceso será $M = 2^{q-1} = 2^3 = 8_{(10}$. Entonces, el exponente vale $e = 1010_{(2} - 8_{(10} = (2^3 + 2^1) - 8 = (8 + 2) - 8 = 10 - 8 = 2_{(10}$.
- Finalmente, los cuatro bits que quedan forman la mantisa que, como está normalizada en la forma 1,*X* con bit implícito toma el valor $R = 1{,}011_{(2}$.

Ahora ya podemos calcular el número representado en base 10 aplicando la fórmula $\pm R \cdot 2^e$ y el TFN:

$$+1{,}011_{(2} \cdot 2^2 = +(2^0 + 2^{-2} + 2^{-3}) \cdot 2^2 = +(2^2 + 2^0 + 2^{-1}) = +(4 + 1 + 0{,}5) = \mathbf{+5{,}5}_{(10}.$$

41. Haced las codificaciones siguientes:

a) El número $-1{,}335_{(10}$ en coma flotante de 8 bits, mantisa de 3 bits normalizada de la forma 1,*X* y con bit implícito empleando una aproximación por truncamiento.

El formato especificado tiene la forma dada por la tabla siguiente:

Signo (*S*)	Exponente (*e*)	Mantisa
1 bit	4 bits	3 bits

En primer lugar, codificaremos la magnitud $1{,}335_{(10}$ en base 2:

- La parte entera es $1_{(10}$:

$$1 = 0 \cdot 2 + \mathbf{1}$$

Es decir $1_{(10} = 1_{(2}$.

- En cuanto a la parte fraccionaria:

$$\begin{array}{lclcl} 0{,}335 & \cdot\ 2 = & 0{,}67 & = & \mathbf{0} + 0{,}67 \\ 0{,}67 & \cdot\ 2 = & 1{,}34 & = & \mathbf{1} + 0{,}34 \\ 0{,}34 & \cdot\ 2 = & 0{,}68 & = & \mathbf{0} + 0{,}68 \\ 0{,}68 & \cdot\ 2 = & 1{,}36 & = & \mathbf{1} + 0{,}36 \\ 0{,}36 & \cdot\ 2 = & 0{,}72 & = & \mathbf{0} + 0{,}72 \\ 0{,}72 & \cdot\ 2 = & 1{,}44 & = & \mathbf{1} + 0{,}44 \end{array} \downarrow$$

Así pues, $0{,}355_{(10} = 0{,}010101...{}_{(2}$ y paramos porque ya tenemos bastantes más bits de los que permite representar este formato.

- Ahora, se juntan las dos partes de la magnitud y obtenemos $1{,}335_{(10} \approx 1{,}010101_{(2}$.

- A continuación normalizamos la expresión obtenida (añadiendo la información del signo) la forma $\pm R \cdot 2^e$: $+1{,}335_{(10} \approx +1{,}010101_{(2} \cdot 2^0$.

- Finalmente, obtenemos los diferentes elementos que definen la representación:

– **Signo**: se trata de un número negativo, por lo tanto, el bit de signo será un **1**.

– **Exponente**: El exponente es $0_{(10}$ que hay que codificar en exceso a $M = 2^{q-1}$. Como disponemos de 4 bits, el exceso es $M = 2^3 = 8$. Por lo tanto hay que representar $0_{(10} + 8_{(10} = 8_{(10}$ con 4 bits:

$$\begin{array}{lclcl} 8 & = & 4 \cdot 2 & + & \mathbf{0} \\ 4 & = & 2 \cdot 2 & + & \mathbf{0} \\ 2 & = & 1 \cdot 2 & + & \mathbf{0} \\ 1 & = & 0 \cdot 2 & + & \mathbf{1} \end{array} \uparrow$$

Es decir $e = 8_{(10} = \mathbf{1000}_{(2}$ (en exceso a 8).

– **Mantisa**: $R = 1{,}010101_{(2}$. Como hay bit implícito $(1{,}X)$, sólo hay que representar la parte de la derecha de la coma y disponemos de 3 bits (con truncamiento). Así pues la mantisa serán los tres bits a partir de la coma: **010**.

– Ahora ya podemos juntar todas las partes:

Signo (*S*)	Exponente (*e*)	Mantisa
1	**1000**	**010**

Es decir, la representación es **1<u>1000</u>010**.

b) Repetid el apartado anterior, pero con una aproximación por redondeo.

La única diferencia con el apartado anterior es el cálculo de la mantisa, porque ahora hay que redondear en vez de truncar. Tal y como se indica más arriba, tenemos $R = 1{,}010101_{(2}$, para redondearlo con tres bits, hay que sumar la mitad de la precisión y truncar el resultado:

$$1{,}010101_{(2} + 0{,}0001_{(2} = 1{,}011001_{(2} \rightarrow \text{truncamos 3 bits} \rightarrow 1{,}011_{(2}$$

Por lo tanto, la representación será la siguiente:

Signo (*S*)	Exponente (*e*)	Mantisa
1	**1000**	**011**

Es decir, la secuencia de bits de la representación es **1<u>1000</u>011**.

c) El número $10{,}0327_{(10}$ en coma flotante de 9 bits, mantisa de 3 bits normalizada de la forma $1{,}X$ y con bit implícito empleando una aproximación por truncamiento.

El formato especificado tiene la forma dada por la tabla siguiente:

Signe (*S*)	Exponente (*e*)	Mantisa
1 bit	5 bits	3 bits

En primer lugar, codificaremos la magnitud $10{,}0327_{(10}$ en base 2:

- La parte entera es $10_{(10}$:

$$\begin{array}{rcll} 10 & = & 5 \cdot 2 & + \ \mathbf{0} \\ 5 & = & 2 \cdot 2 & + \ \mathbf{1} \\ 2 & = & 1 \cdot 2 & + \ \mathbf{0} \\ 1 & = & 0 \cdot 2 & + \ \mathbf{1} \end{array} \uparrow$$

Es decir $10_{(10} = 1010_{(2}$.

- En cuanto a la parte fraccionaria:

$$\begin{array}{lclcl} 0{,}0327 & \cdot\ 2 & = & 0{,}0654 & = & \mathbf{0} + 0{,}0654 \\ 0{,}0654 & \cdot\ 2 & = & 0{,}1308 & = & \mathbf{0} + 0{,}1308 \\ 0{,}1308 & \cdot\ 2 & = & 0{,}2616 & = & \mathbf{0} + 0{,}2616 \\ 0{,}2616 & \cdot\ 2 & = & 0{,}5232 & = & \mathbf{0} + 0{,}5232 \\ 0{,}5232 & \cdot\ 2 & = & 1{,}0464 & = & \mathbf{1} + 0{,}0464 \\ 0{,}0464 & \cdot\ 2 & = & 0{,}0928 & = & \mathbf{0} + 0{,}0928 \end{array} \downarrow$$

Así pues, $0{,}0327_{(10} = 0{,}000010...{}_{(2}$ y paramos porque ya tenemos bastantes más bits de los que permite representar este formato.

- Ahora, se juntan las dos partes de la magnitud y obtenemos $10{,}0327_{(10} \approx 1010{,}000010_{(2}$.

- A continuación normalizamos la expresión obtenida (añadiendo la información del signo) en la forma $\pm R \cdot 2^e$: $+10{,}0327_{(10} \approx +1{,}010000010_{(2} \cdot 2^3$.

- Finalmente, obtenemos los diferentes elementos que definen la representación:

– **Signo**: se trata de un número positivo, por lo tanto, el bit de signo será un **0**.

– **Exponente**: El exponente es $3_{(10}$ que hay que codificar en exceso a $M = 2^{q-1}$. Como disponemos de 5 bits el exceso es $M = 2^4 = 16$. Por lo tanto hay que representar $3_{(10} + 16_{(10} = 19_{(10}$ con 4 bits:

$$\begin{array}{rcll} 19 & = & 9 \cdot 2 & + \ \mathbf{1} \\ 9 & = & 4 \cdot 2 & + \ \mathbf{1} \\ 4 & = & 2 \cdot 2 & + \ \mathbf{0} \\ 2 & = & 1 \cdot 2 & + \ \mathbf{0} \\ 1 & = & 0 \cdot 2 & + \ \mathbf{1} \end{array} \uparrow$$

Es decir $e = 19_{(10} = \mathbf{10011_{(2}}$ (en exceso a 16).

– **Mantisa**: $R = 1{,}010000010_{(2}$. Como hay bit implícito (1,*X*), sólo hay que representar la parte de la derecha de la coma y disponemos de 3 bits (con truncamiento). Así pues la mantisa serán los tres bits a partir de la coma: **010**.

– Ahora ya podemos juntar todas las partes:

Signo (*S*)	Exponente (*e*)	Mantisa
0	**10011**	**010**

Es decir, la representación es **0<u>10011</u>010**.

42. Determinad si el número $\mathbf{2{,}89_{(10} \cdot 10^{10}}$ es representable en un formato de coma flotante de 16 bits, con mantisa normalizada de la forma 1,*X*, bit implícito y 5 bits para el exponente.

El formato especificado tiene la forma dada por la tabla siguiente:

Signo (*s*)	Exponente (*e*)	Mantisa
1 bit	5 bits	10 bits

El número positivo de magnitud mayor que podemos representar en este formato tiene un cero en el bit de signo y un 1 en el resto de bits: 0<u>11111</u>1111111111.

Buscamos el número decimal que representa:
- Bit de signo $s = 0$, por lo tanto, se trata de un número positivo.
- Exponente: $11111_{(2}$, es decir $q = 5$. Por tanto, el exceso será $M = 2^{q-1} = 2^4 = 16_{(10}$. Entonces, el exponente vale $e = 11111_{(2} - 16_{(10} = 31 - 16 = 15_{(10}$.
- Finalmente, los bits que quedan forman la mantisa, que, como está normalizada en la forma $1,X$ con bit implícito, toma el valor $R = 1,1111111111_{(2}$.

Ahora ya podemos calcular el número representado en base 10 aplicando la fórmula $\pm R \cdot 2^e$ y el TFN:

$+1,1111111111_{(2} \cdot 2^{15} = +11111111111_{(2} \cdot 2^5 = +\ 2047 \cdot\ 2^5 = +65504_{(10}$.

Como el número $2,89_{(10} \cdot 10^{10}$ es mayor que el $65504_{(10}$, no se puede representar en este formato.

43. Determinad si el número $\mathbf{-1256_{(10} \cdot 10^{-2}}$ es representable en un formato de coma flotante de 10 bits, con mantisa normalizada de la forma $1,X$, bit implícito y 6 bits para el exponente.

El formato especificado tiene la forma dada por la tabla siguiente:

Signo (s)	Exponente (e)	Mantisa
1 bit	6 bits	3 bits

El número negativo de magnitud mayor que podemos representar en este formato tiene todos los bits a 1: 1<u>111111</u>111.

Buscamos el número decimal que representa:
- Bit de signo $s = 1$, por lo tanto, se trata de un número negativo.
- Exponente: $111111_{(2}$, es decir $q = 6$. Por tanto, el exceso será $M = 2^{q-1} = 2^5 = 32_{(10}$. Entonces, el exponente vale $e = 111111_{(2} - 32_{(10} = 63 - 32 = 31_{(10}$.
- Finalmente, los bits que quedan forman la mantisa, que, como está normalizada en la forma $1,X$ con bit implícito, toma el valor $R = 1,111_{(2}$.

Ahora ya podemos calcular el número representado en base 10 aplicando la fórmula $\pm R \cdot 2^e$ y el TFN:
$-1,111_{(2} \cdot 2^{31} = -1111_{(2} \cdot 2^{28} = -15 \cdot\ 2^{28}{}_{(10}$.

Como la magnitud del número $-1256_{(10} \cdot 10^{-2}$ (o lo que es lo mismo, del $-12,56_{(10}$) es más pequeña que $-15 \cdot\ 2^{28}{}_{(10}$, se puede representar en este formato.

Ejercicios de autoevaluación

1. Podemos obtener la codificación en Ca2 y 8 bits del número $-10_{(10}$ de las siguientes formas :

- $2^8 - 10 = 256_{(10} - 10_{(10} = 246_{(10} = \mathbf{11110110}_{(Ca2}$, o bien,
- $2^8 - 10 = 100000000_{(2} - 1010_{(2} = \mathbf{11110110}_{(Ca2}$, o bien,
- $+10_{(10} = +1010_{(2} \rightarrow$ Representación de la magnitud positiva $\rightarrow 00001010_{(Ca2} \rightarrow$
  $\rightarrow$ Cambio de signo $\rightarrow \mathbf{11110110}_{(Ca2}$

Para codificar el número en signo y magnitud, hemos de añadir a la representación de la magnitud en base 2 la información del signo. Como es un número negativo, el bit de signo será 1. Por lo tanto, la codificación será $\mathbf{10001010}_{(SM2}$

2.
a) Si se trata de un número codificado en complemento a 2.
En este caso se trata de un número positivo porque el bit del extremo izquierdo es 0. Por lo tanto, el resto de los bits codifican la magnitud como en el caso de signo y magnitud. Si aplicamos el TFN a la magnitud, obtenemos:

$0100100_{(2} = 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 36_{(10}$

En este caso codifica el número decimal +36.

b) Si se trata de un número codificado en signo y magnitud.
Como se trata de un número positivo, y la codificación en signo y magnitud de un número positivo coincide con la representación en Ca2, la cadena codifica el mismo número, el $+36_{(10}$.

3.

	1	1	1											
		1	1	1	0	1	0	1	0	1	1	0	0	(2
+			1	1	1	0	0	0	1	0	0	1	0	(2
	1	0	1	0	1	1	0	1	1	1	1	1	0	(2

Lo que podemos observar es que necesitamos un bit más para poder representar el resultado de la operación.

4. Como se trata de un número positivo, el bit de signo es **0**. Con respecto a la magnitud $12{,}346_{(10}$, primero pasamos a binario la parte entera y, posteriormente, la parte fraccionaria. Para la parte entera utilizamos el algoritmo de divisiones sucesivas por la base de llegada (2):

$$\begin{array}{rcll} 12 & = & 6 \cdot 2 & + \mathbf{0} \\ 6 & = & 3 \cdot 2 & + \mathbf{0} \\ 3 & = & 1 \cdot 2 & + \mathbf{1} \\ 1 & = & 0 \cdot 2 & + \mathbf{1} \end{array} \uparrow$$

Así pues, $12_{(10} = 1100_{(2}$. Con respecto a la parte fraccionaria, hacemos multiplicaciones sucesivas por la base de llegada (2):

$$\begin{array}{lclcll} 0{,}346 & \cdot 2 & = & 0{,}692 & = & \mathbf{0} + 0{,}692 \\ 0{,}692 & \cdot 2 & = & 1{,}384 & = & \mathbf{1} + 0{,}384 \\ 0{,}384 & \cdot 2 & = & 0{,}768 & = & \mathbf{0} + 0{,}768 \\ 0{,}768 & \cdot 2 & = & 1{,}536 & = & \mathbf{1} + 0{,}536 \\ 0{,}536 & \cdot 2 & = & 1{,}072 & = & \mathbf{1} + 0{,}072 \end{array} \downarrow$$

Como el formato especificado sólo tiene 3 bits para la parte fraccionaria, ya tenemos más bits de los necesarios y podemos detener el proceso aquí. Por lo tanto, $0{,}346_{(10} = 0{,}01011...{}_{(2}$. Para aproximar el valor con 3 bits por truncamiento eliminamos todos los bits a partir del tercero: $0{,}346_{(10} = 0{,}010_{(2}$

A continuación, unimos las partes entera y fraccionaria y obtenemos: $12{,}346_{(10} \approx 1100{,}010_{(2}$. Finalmente, para obtener la representación en el formato indicado, hay que añadir el bit de signo, de manera que la tira de bits que representa este número en el formato dado es: $01100{,}010_{(2}$. Hay que recordar que la tira de bits que se almacenaría en un computador no contiene la coma ni la especificación de la base: **01100010**.

5. Podemos obtener la representación en Ca2 y 8 bits del $-45_{(10}$ de cualquiera de las siguientes formas:

- $2^8 - 45 = 256_{(10} - 45_{(10} = 211_{(10} = \mathbf{11010011}_{(Ca2}$, o bien,
- $2^8 - 45 = 100000000_{(2} - 101101_{(2} = \mathbf{11010011}_{(Ca2}$, o bien,
- $+45_{(10} = +101101_{(2}$ → Representación de la magnitud positiva → $00101101_{(Ca2}$ → → cambio de signo → $\mathbf{11010011}_{(Ca2}$

6. En primer lugar, obtendremos la codificación binaria del número $-35{,}25_{(10}$:

- El bit de signo es 1 porque el número es negativo.
- La parte entera es $35_{(10}$:

$$\begin{array}{rcll} 35 & = & 17 \cdot 2 & + 1 \\ 17 & = & 8 \cdot 2 & + 1 \\ 8 & = & 4 \cdot 2 & + 0 \\ 4 & = & 2 \cdot 2 & + 0 \\ 2 & = & 1 \cdot 2 & + 0 \\ 1 & = & 0 \cdot 2 & + 1 \end{array} \uparrow$$

Es decir, $35_{(10} = 100011_{(2}$.

- Con respecto a la parte fraccionaria:

$$\begin{array}{lclcll} 0{,}25 & \cdot 2 & = & 0{,}5 & = & \mathbf{0} + 0{,}5 \\ 0{,}5 & \cdot 2 & = & 1 & = & \mathbf{1} + 0 \end{array} \downarrow$$

Así pues, $0{,}25_{(10} = 0{,}01_{(2}$

- Finalmente, se unen todas las partes y obtenemos $-35{,}25_{(10} = \mathbf{1}100011{,}01_{(SM2}$.

Con el fin de representar exactamente este número, es necesario un formato que tenga un total de 9 bits:
- Un bit de signo
- 6 bits para la parte entera
- 2 bits para la parte fraccionaria

7. La equivalencia binaria de estos números decimales es la siguiente:
$+12{,}25_{(10} = +1100{,}01_{(2}$
$+32{,}5_{(10} = +100000{,}1_{(2}$

En signo y magnitud y 9 bits donde 2 son fraccionarios, la codificación es la siguiente:
$+1100{,}01_{(2} = 0001100{,}01_{(SM2}$
$+100000{,}1_{(2} = 0100000{,}10_{(SM2}$
Para hacer la suma examinamos los bits de signo. Se trata de dos números positivos, por lo tanto, se deben sumar las magnitudes y añadir al resultado un bit de signo positivo:

$$\begin{array}{rcccccccl} & 0 & 0 & 1 & 1 & 0 & 0, & 0 & 1 & _{(2} \\ + & 1 & 0 & 0 & 0 & 0 & 0, & 1 & 0 & _{(2} \\ \hline & 1 & 0 & 1 & 1 & 0 & 0, & 1 & 1 & _{(2} \end{array}$$

Al hacer la suma no se produce desbordamiento porque no hay acarreo en la última etapa, que es lo que determina si hay desbordamiento en este formato. Al resultado obtenido se debe añadir el bit de signo para tener la codificación en signo y magnitud correcta: $\mathbf{0101100{,}11_{(SM2}}$

8. Para codificar en BCD tan sólo hemos de codificar cada dígito decimal con 4 bits:
$1 = 0001_{(2}$
$7 = 0111_{(2}$
$8 = 1000_{(2}$

Finalmente, unimos los códigos BCD para obtener la cadena final: **000101111000**

9. El formato especificado tiene la forma dada por la tabla siguiente:

Signo (s)	Exponente (e)	Mantisa
1 bit	4 bits	3 bits

- En primer lugar, hemos de codificar la magnitud $12{,}346_{(10}$ en base 2. En el ejercicio de autoevaluación 4 hemos obtenido que $12{,}346_{(10} \approx 1100{,}010_{(2}$
- Seguidamente normalizamos la expresión obtenida en la forma $\pm R \cdot 2^e$:
$+12{,}346_{(10} \approx +1{,}100010_{(2} \cdot 2^3$.

- Finalmente, obtenemos los diferentes elementos que definen la representación:
  - **Signo**: se trata de un número positivo, por tanto, el bit de signo será **0**.
  - **Exponente**: el exponente es $3_{(10}$, que hay que codificar en exceso en $M = 2^{q-1}$. Como disponemos de 4 bits, el exceso es $M = 2^3 = 8$. Por lo tanto, hay que representar $3_{(10} + 8_{(10} = 11_{(10}$ con 4 bits:

$$\begin{array}{rcccc} 11 & = & 5 \cdot 2 & + & 1 \\ 5 & = & 2 \cdot 2 & + & 1 \\ 2 & = & 1 \cdot 2 & + & 0 \\ 1 & = & 0 \cdot 2 & + & 1 \end{array} \uparrow$$

Es decir $e = 11_{(10} = \mathbf{1011_{(2}}$ (en exceso a 8).
  - **Mantisa**: $R = 1{,}100010_{(2}$. Como hay bit implícito $(1,X)$, sólo hay que representar la parte de la derecha de la coma y disponemos de 3 bits (con truncamiento). Así pues, la mantisa serán los tres bits a partir de la coma: **100**.
  - Ahora ya podemos juntar todas las partes:

Signo (s)	Exponente (e)	Mantisa
0	**1011**	**100**

Es decir, la representación es **0<u>1011</u>100**.

10. En primer lugar, desempaquetamos la cadena de bits. Cada dígito hexadecimal corresponde a 4 bits:
- 3 = 0011
- 7 = 0111
- 8 = 1000

Por lo tanto, $378_{(16}$ empaqueta la cadena de bits 001101111000. Esta cadena codifica un número en coma flotante con 4 bits de mantisa: 0<u>01101111</u>000. Buscamos el número decimal que representa:
- Bit de signo $s = 0$, por lo tanto, se trata de un número positivo.
- Exponente: $0110111_{(2}$, es decir $q = 7$. Por lo tanto, el exceso será $M = 2^{q-1} = 2^6 = 64_{(10}$. Entonces, el exponente vale $e = 0110111_{(2} - 64_{(10} = 55 - 64 = -9_{(10}$.
- Finalmente, los bits que quedan forman la mantisa, que, como está normalizada en la forma $1,X$ con bit implícito, toma el valor $R = 1,1000_{(2}$.

Ahora ya podemos calcular el número representado en base 10 aplicando la fórmula $\pm R \cdot 2^e$ y el TFN:

$$-1,1000_{(2} \cdot 2^{-9} = -11_{(2} \cdot 2^{-10} = -3_{(10} \cdot 2^{-10}{}_{(10} = -\frac{3}{1024}$$

Glosario

acarreo *m* Bit que indica que se ha superado el valor *b* de la base de numeración al sumar dos dígitos. La suma de dos valores numéricos se lleva a cabo dígito a dígito, progresando de derecha a izquierda. Un suma y sigue al sumar dos dígitos indica que hay que añadir una unidad al hacer la suma de los dos dígitos siguientes.
en carry

base *f* Número de dígitos diferentes en un sistema de numeración.

binary coded decimal **(BCD)** Véase **decimal codificado en binario**.

binario *m* Sistema de numeración en base 2.

bit *m* Abreviación de *BInary DigiT* ('dígito binario'). Corresponde a la unidad de información más pequeña posible. Se define como la cantidad de información asociada a la respuesta de una pregunta formulada de una manera no ambigua, donde sólo son posibles dos alternativas de respuesta, y que, además, tienen la misma probabilidad de ser escogidas.

bit menos significativo *m* Bit menos significativo (más a la derecha) de una representación numérica posicional.
sigla: LSB.

bit más significativo *loc* Bit más significativo (más a la izquierda) de una representación numérica posicional.
sigla: MSB.

borrow *m* Suma y sigue empleado en la operación de resta.

byte Véase **octeto**.

cadena *f* Conjunto de elementos de un mismo tipo dispuestos uno a continuación del otro.

carry Véase **acarreo**.

coma fija *f* Sistema de representación numérica que emplea un número fijo de dígitos enteros y fraccionarios.

coma flotante *f* Sistemas de representación numérica que codifican un número variable de dígitos enteros y fraccionarios. La cantidad de dígitos enteros y fraccionarios depende del formato y del valor numérico que se representa.

decimal *m* Sistema de numeración en base 10.

decimal codificado en binario *m* Designa la codificación de los dígitos decimales (0,1,2...,9) sobre un conjunto de 4 dígitos binarios.
sigla: BCD

desbordamiento *m* Indicación de que el resultado de una operación está fuera del rango de representación disponible.
en overflow

dígito *m* Elemento de información que puede coger un número finito de valores diferentes. La cantidad de valores diferentes es dada por la base de numeración.

digital *m* Sistema que trabaja con datos discretos.

empaquetamiento *m* Transformación que permite representar la información binaria de forma más compacta.

formato *m* Descripción estructural de una secuencia de datos, donde se especifica el tipo, la longitud y la disposición de cada elemento.

fraccionario *m* Menor que la unidad.

hexadecimal *m* Sistema de numeración en base 16.

LSB Véase **bit menos significativo**

palabra *f* Unidad de información procesada por un computador de un solo golpe (en paralelo). *en* word

MSB Véase **bit más significativo**.

octal *m* Sistema de numeración en base 8.

octeto *m* Cadenas de 8 bits.

overflow Véase desbordamiento.

precisión *f* Diferencia entre dos valores consecutivos de una representación.

raíz *f* Base de un sistema de numeración.

rango *m* Conjunto de valores que indican los márgenes entre los cuales se encuentran los valores posibles de un formato de representación.

representación Véase formato.

word Véase **palabra**.

Bibliografía

De Miguel, P. (1996). *Fundamentos de los computadores* (5.ª ed., cap. 2). Madrid: Paraninfo.

Patterson, D.; Henessy, J. L. (1995). *Organización y diseño de computadores: la interfaz hardware/software*. Madrid: McGraw-Hill.

Stallings, W. (1996). *Organización y arquitectura de computadores: diseño para optimizar prestaciones*. Madrid: Prentice Hall.