# SISTEMAS DE PROCESAMIENTO DE DATOS

## Unidad 2 – Sistemas de representación de la información

Universidad Tecnológica Nacional – Facultad Regional Avellaneda
Técnico Superior en Programación

# Contenido

# Unidad 2. Sistemas de representación de la información

Desde los comienzos de la informática, los ingenieros y todas aquellas personas dedicadas al desarrollo de sistemas informáticos, debieron ponerse de acuerdo en la forma en la que cada uno representaba la información. Lo mismo sucede a grandes rasgos con el ser humano, que debió ponerse de acuerdo, por ejemplo, para representar cantidades con el sistema numérico decimal. Como ese, podríamos estar horas hablando de distintos ejemplos. Volviendo al caso de la informática, podemos poner como ejemplo el caso de la primera computadora, la ENIAC. Este dispositivo primitivo representaba toda la información y los datos, utilizando el sistema decimal. Esto quería decir que cada persona que quisiera desarrollar algo allí, debía utilizar ese sistema. A partir de la EDVAC en adelante, el sistema utilizado universalmente en los dispositivos electrónicos fue el sistema de numeración binario. Esto fue así ya que representaba muchas ventajas para poder manipular la información y representarla electrónicamente. De todas maneras, existen otros sistemas de numeración como el octal, y el hexadecimal, también utilizados en algunos casos para representar información.

**Por qué en el interior de la computadora se opera con 2 estados eléctricos correspondientes a 2 estados lógicos, representados por el 0 y 1 binarios**:

Los millones de transistores que componen los circuitos de una computadora funcionan como las llaves de dos estados “si-no” usadas para la electricidad hogareña; pero los transistores son dispositivos electrónicos que pueden cambiar de un estado al otro, millones de veces por segundo. Para simplificar la idea fundamental sin entrar en detalles electrónicos podemos decir que cada transistor deja pasar la corriente eléctrica o no, determinando un 1 o un 0 respectivamente.

Operar con 2 estados es más práctico, más simple y más confiable que hacerlo con 10 valores eléctricos distintos y en ese caso tener que representarlos en decimal, debido entre otras cosas, a características propias de los componentes electrónicos que determinan la estabilidad de las señales y afectarían a dichos valores.

Con el avance de la tecnología y el uso de las computadoras en distintos campos, fue necesario empezar a desarrollar mecanismos para representar algunos tipos de datos particulares, como números negativos o números con decimales. De allí surgieron, por ejemplo, los complementos a 1 y a 2, los números con punto fijo y los números con punto flotante.

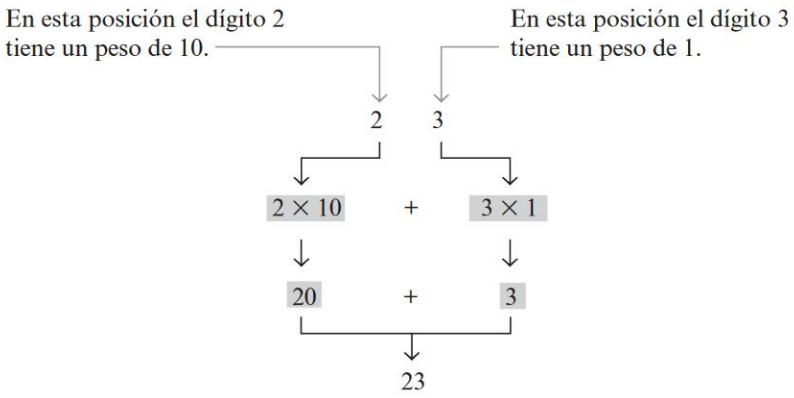

*Fig. 1. Representación del proceso de transformación de los datos en información.*

Antes de ver en detalle cada uno de los sistemas de representación antes mencionados, es necesario dejar en claro dos conceptos fundamentales que ya fueron mencionados en párrafos anteriores: Dato e información.

Llamaremos “dato” a todo aquello que es procesador por la computadora, y que luego de ese proceso da como resultado, una salida significativa para el usuario. Esta salida, es lo que llamaremos “información”. Para llevarlo a un caso práctico, podemos decir que queremos sumar 2 + 4. Los “datos” será aquello que queremos procesar, en este caso el número 2 y el número 4. El proceso será la suma de estos dos datos, y el resultado, el número 6, es la información resultante de aquel proceso.

# Sistema de numeración decimal

En el sistema de numeración decimal cada uno de los diez dígitos, de 0 a 9, representa una determinada cantidad. Como ya sabe, los diez símbolos (dígitos) no se limitan a expresar solamente diez cantidades diferentes, ya que usamos varios dígitos en las posiciones adecuadas dentro de un número para indicar la magnitud de la cantidad. Es posible especificar cantidades hasta nueve antes de quedarse sin dígitos; si se desea especificar una cantidad mayor que nueve, se emplean dos o más dígitos y la posición de cada dígito dentro del número indica la magnitud que representa. Por ejemplo, si deseamos expresar la cantidad veintitrés, usaremos (en sus respectivas posiciones dentro del número) el dígito 2 para representar la cantidad de veinte y el dígito 3 para representar la cantidad de 3, como se ilustra a continuación:




La posición de cada dígito en un número decimal indica la magnitud de la cantidad representada y se le puede asignar un peso. Los pesos para los números enteros son las potencias positivas de diez, que aumentan de derecha a izquierda, comenzando por 100 = 1.

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$$

Para números fraccionarios, los pesos son las potencias negativas de diez que decrecen de izquierda a derecha comenzando por 10−1.

$$10^2\ 10^1\ 10^0, 10^{-1}\ 10^{-2}\ 10^{-3} \ldots$$

Coma decimal

El valor de un número decimal es la suma de los dígitos después de haber multiplicado cada dígito por su peso.

# Sistema de numeración binario

Para aprender a contar en el sistema binario, en primer lugar, es preciso observar cómo se cuenta en el sistema decimal. Comenzamos en cero y continuamos hasta el nueve antes de quedarnos sin dígitos. Luego, comenzamos con otra posición de dígito (a la izquierda) y continuamos contando desde 10 hasta 99. En este punto, se terminan todas las combinaciones con dos dígitos, por lo que es necesaria una tercera posición de dígito para poder contar desde 100 hasta 999. Cuando contamos en binario se produce una situación similar, excepto en que sólo disponemos de dos dígitos, denominados bits. Empezamos a contar: 0, 1. En este punto, ya hemos utilizado los dos dígitos, por lo que incluimos otra posición de dígito y continuamos: 10, 11. Ahora, hemos agotado todas las combinaciones de dos dígitos, por lo que es necesaria una tercera posición. Con tres posiciones de dígito podemos continuar contando: 100, 101, 110 y 111. Ahora necesitamos una cuarta posición de dígito para continuar, y así sucesivamente. En general, con n bits se puede contar hasta un número igual a 2n −1.

| Número decimal | Número binario | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 |

*Fig. 2. Tabla binaria y su equivalencia con los números decimales desde el 0 hasta el 15*

Un número binario es un número con peso. El bit más a la derecha es el LSB (Least Significant Bit, bit menos significativo) en un número binario entero y tiene un peso de 20 = 1. El bit más a la izquierda es el MSB (Most Significant Bit, bit más significativo); su peso depende del tamaño del número binario.

Los números fraccionarios también pueden representarse en el sistema binario colocando bits a la derecha de la coma binaria, del mismo modo que los números decimales fraccionarios se colocan a la derecha de la coma decimal. En un número binario con parte fraccionaria, el bit más a la izquierda es el

| Potencias positivas de dos (números enteros) | | | | | | | | | Potencias negativas de dos (números fraccionarios) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ |
| 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 | 1/64 |
| | | | | | | | | | 0,5 | 0,25 | 0,125 | 0,0625 | 0,03125 | 0,015625 |

*Fig. 3. Observe que el peso se duplica para cada potencia positiva de dos y que se reduce a la mitad para cada potencia negativa de dos.*

MSB y tiene un peso de 2-1 = 0,5. Los pesos fraccionarios de los respectivos bits decrecen de izquierda a

derecha según las potencias negativas de dos para cada bit. La estructura de pesos de un número binario es:

donde n es el número de bits a partir de la coma binaria. Por tanto, todos los bits a la izquierda de la coma binaria tienen pesos que son potencias positivas de dos, como previamente se ha visto para los números enteros. Todos los bits situados a la derecha de la coma binaria tienen pesos que son potencias negativas de dos, o pesos fraccionales.

El valor decimal de cualquier número binario puede hallarse sumando los pesos de todos los bits que están a 1 y descartando los pesos de todos los bits que son 0. Ejemplo;

Se determina el peso de cada bit que está a 1, y luego se obtiene la suma de los pesos para obtener el número decimal. Por ejemplo:

Peso: $2^6 2^5 2^4 2^3 2^2 2^1 2^0$

$$2^{n-1} \ldots 2^3\, 2^2\, 2^1\, 2^0 , 2^{-1}\, 2^{-2} \ldots 2^{-n}$$

Coma binaria

Número binario: 1 1 0 1 1 0 1

$1101101 = 2_6 + 2_5 + 2_3 + 2_2 + 2_0$

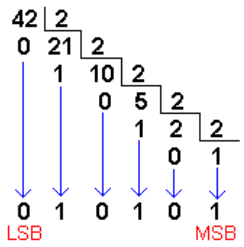
$= 64 + 32 + 8 + 4 + 1 =$ **109**

## Conversión de decimal a binario

Existen varios métodos para convertir números decimales a binarios. Aquí estudiaremos dos: Método de la suma de pesos y método de la división sucesiva de 2.

El método de la suma de pesos consiste en determinar el conjunto de pesos binarios cuya suma es igual al número decimal. Una forma fácil de recordar los pesos binarios es que el peso más bajo es 1, es decir 20, y que, duplicando cualquier peso, se obtiene el siguiente peso superior; por tanto, la lista de los siete primeros pesos binarios será: 1, 2, 4, 8, 16, 32, 64, como verá en una sección posterior. Por ejemplo, el número decimal 9 puede expresarse como la suma de pesos binarios siguiente: 9 = 8 +1 o 9 = $2^3 + 2^0$. Colocando los 1s en las posiciones de pesos apropiadas, $2^3$ y $2^0$, y los 0s en las posiciones $2^2$ y $2^1$ se determina el número binario correspondiente al decimal 9: 1 0 0 1.

El método de la división sucesiva de 2, es un método sistemático para convertir a binario, números enteros. Por ejemplo, para convertir el número decimal 42 a binario, comenzamos dividiendo 42 entre 2. A continuación, cada cociente resultante se divide entre dos hasta obtener un cociente cuya parte entera sea igual a 0. Los restos generados en cada división forman el número binario. El primer resto es el bit menos significativo (LSB) del número binario y el último resto es el bit más significativo (MSB).

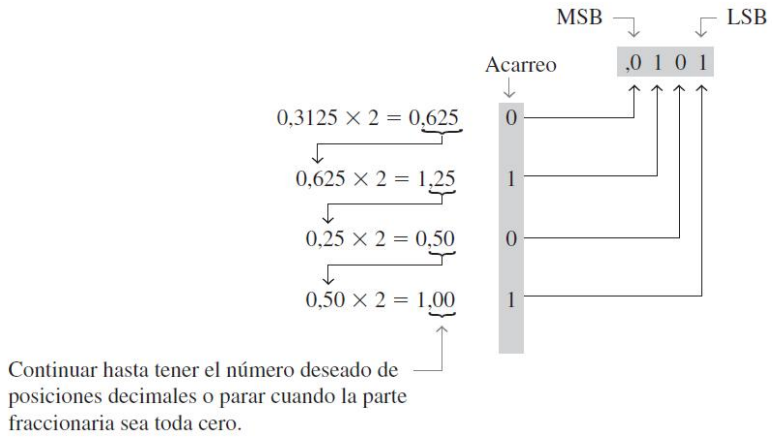


## Conversión de decimales fraccionarios a binario

Para realizar la conversión de decimales fraccionarios a binario existen dos métodos: método de suma de pesos (similar al utilizado en la conversión de decimales enteros a binario) y la multiplicación sucesiva de 2.

El método de la suma de pesos se puede aplicar a los números decimales fraccionarios, como se muestra en el siguiente ejemplo: 0,625 = 0,5 + 0,125 = $2^{-1} + 2^{-3}$ = 0,101. Esto indica que hay un 1 en la posición $2^{-1}$, un 0 en la posición $2^{-2}$ y un 1 en la posición $2^{-3}$.

Como hemos visto, los números decimales enteros pueden convertirse a binario dividiendo sucesivamente entre dos. Los números decimales fraccionarios pueden convertirse en números binarios multiplicando sucesivamente por 2. Por ejemplo, para convertir a binario el número decimal fraccionario 0,3125, comenzamos multiplicando 0,3125 por 2 y después se multiplica cada parte fraccional resultante del producto por 2 hasta que el producto fraccionario sea cero o hasta que se alcance el número deseado de posiciones decimales. Los dígitos acarreados o, acarreos, generados por las multiplicaciones dan lugar al número binario. El primer acarreo que se obtiene es el MSB y el último acarreo es el LSB. Este procedimiento se ilustra como sigue:




## Suma de números binarios

La suma de números binarios puede resumirse de la siguiente manera:

- 0 + 0 = 0 (Suma 0 con acarreo 0)
- 0 + 1 = 1 (Suma 1 con acarreo 0)
- 1 + 0 = 1 (Suma 1 con acarreo 0)
- 1 + 1 = 10 (Suma 0 con acarreo 1)

Observe que las tres primeras reglas dan lugar a un resultado de un solo bit y la cuarta regla, la suma de dos 1s, da lugar a 2 en binario (10). Cuando se suman números binarios, teniendo en cuenta la última regla se obtiene en la columna dada la suma de 0 y un acarreo de 1 que pasa a la siguiente columna de la izquierda.

## Resta de números binarios

La resta de números binarios puede resumirse de la siguiente manera:

- 0 − 0 = 0
- 1 − 1 = 0
- 1 − 0 = 1
- 10 − 1 = 1 (0 − 1 con acarreo negativo de 1)

Cuando se restan números, algunas veces se genera un acarreo negativo que pasa a la siguiente columna de la izquierda. En binario, sólo se produce un acarreo negativo cuando se intenta restar 1 de 0. En este caso, cuando se acarrea un 1 a la siguiente columna de la izquierda, en la columna que se está restando se genera un 10, y entonces debe aplicarse la última de las cuatro reglas enumeradas.

## Multiplicación de números binarios

La multiplicación de números binarios puede resumirse de la siguiente manera:

- 0 × 0 = 0
- 0 × 1 = 0
- 1 × 0 = 0
- 1 × 1 = 1

La multiplicación con números binarios se realiza de la misma forma que con números decimales. Se realizan los productos parciales, desplazando cada producto parcial sucesivo una posición hacia la izquierda, y sumando luego todos los productos parciales.

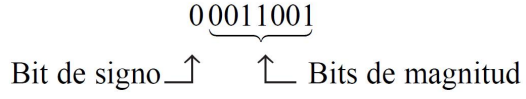
# Representación de números negativos en binario

Hasta ahora se mostró cómo es posible representar números decimales en el sistema de numeración binario, pero solo números decimales positivos. Como todos los sistemas digitales utilizan este sistema de numeración para representar la información, resulta fundamental que sea posible la representación de números decimales negativos. Existen para ello existen tres formatos: formato de signo-magnitud, formato de complemento a 1 y formato de complemento a 2.

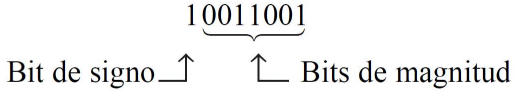
En cualquiera de los formatos, el bit más a la izquierda será utilizado para representar el signo binario. Un 0 indica que el número es positivo, mientras que un 1 indica que el número es negativo.

## Formato de signo-magnitud

Cuando un número binario con signo se representa en formato signo-magnitud, el bit más a la izquierda es el bit de signo y los restantes bits son los bits de magnitud. Los bits de magnitud son el número binario real (no complementado) tanto para los números positivos como para los negativos. Por ejemplo, el número decimal +25 se expresa utilizando un número binario con signo de 8 bits en el formato de signo-magnitud como:



Mientras que el número -25 se representa como:




## Formato de complemento a 1

Para poder estudiar cómo se representan los números negativos en este formato, es necesario saber que el complemento a 1 de un número binario se halla cambiando todos los 1s por 0s y todos los 0s por 1s. Por ejemplo, dado el número binario 01100101 (sin bit de signo), el complemento a 1 del mismo sería 10011010. Nótese cómo casa bit cambió por su valor opuesto.

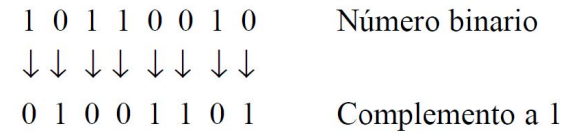



*Fig. 4. Ejemplo de cálculo del complemento a 1 de un número binario.*

En el formato de complemento a 1 para representar negativos, los números positivos en el formato de complemento a 1 se representan de la misma forma que los números positivos en el formato signo-magnitud. Sin embargo, los números negativos son el complemento a 1 del correspondiente número positivo. Por ejemplo, con ocho bits, el número decimal −25 se expresa como el complemento a 1 de +25 (00011001), es decir 11100110.

## Formato de complemento a 2

En este formato, los números negativos se representan calculando el complemento a 2 de número positivo.

El complemento a 2 de un número se obtiene calculando primero el complemento a 1 y luego sumándole 1. Por ejemplo, para calcular el complemento a 2 del número binario 10110010 (sin signo), primero tenemos que calcular el complemento a 1 del mismo. Esto da como resultado 01001101. A este resultado le sumamos 1, lo que da como resultado 01001110. Este último número binario será el complemento a 2 del número inicial.

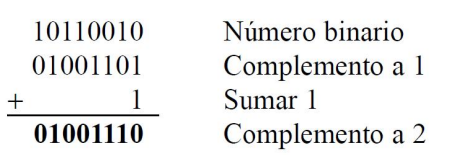



*Fig. 5. Ejemplo de cálculo del complemento a 2 de un número binario.*

Los números positivos en el formato de complemento a 2 se representan de la misma forma que en el formato signo-magnitud y de complemento a 1. Los números negativos son el complemento a 2 del correspondiente número positivo. De nuevo, utilizando ocho bits, tomamos −25 y lo expresamos como el complemento a 2 de +25 (00011001): 11100111.

# El valor decimal de los números con signo

Para poder realizar el pasaje de un número binario a decimal, si el número es positivo, podemos aplicar la técnica vista anteriormente. Pero en el caso de los números negativos, para poder calcular el número decimal asociado, será necesario primero conocer el formato en el que están representados.

## Valor decimal en formato signo-magnitud

Los valores decimales de los números positivos y negativos en el formato signo-magnitud se determinan sumando los pesos de todas las posiciones de los bits de magnitud cuando son 1 e ignorando aquellas posiciones en las que haya ceros. El signo se determina examinando el bit de signo.

### EJEMPLO

Determinar el valor decimal del número binario con signo expresado como signo-magnitud: 10010101.

Los siete bits de magnitud y sus pesos potencias de dos son los siguientes:

| $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |

Sumando los pesos de las posiciones donde hay 1s, tenemos

$$16 + 4 + 1 = 21$$

El bit de signo es 1; por tanto, el número decimal es **–21.**

## Valor decimal en formato de complemento a 1

Los valores decimales de los números positivos en el formato de complemento a 1 se determinan sumando los pesos de todas las posiciones de bit donde haya 1 y se ignoran aquellas posiciones donde haya ceros. Los valores decimales de los números negativos se determinan asignando el valor negativo al peso del bit de signo, y sumando todos los pesos donde haya 1s y sumando 1 al resultado.

### EJEMPLO

Determinar los valores decimales de los números binarios con signo expresados en complemento a 1:
**(a)** 00010111 **(b)** 11101000

***Solución***

**(a)** Los bits y sus pesos según las potencias de dos para el número positivo son:

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

Sumando los pesos donde hay 1s,

$$16 + 4 + 2 + 1 = +\mathbf{23}$$

**(b)** Los bits y sus pesos según las potencia de dos para el número negativo son los siguientes. Observe que el bit de signo negativo tiene un peso de $-2^7$, es decir, –128.

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

Sumando los pesos de las posiciones donde hay 1s,

$$-128 + 64 + 32 + 8 = -24$$

Sumando 1 al resultado, el número decimal final es:

$$-24 + 1 = \mathbf{-23}$$

## Valor decimal en formato de complemento a 2

Los valores decimales de los números positivos en el formato de complemento a 2 se determinan sumando los pesos de todas las posiciones de bit donde haya 1 y se ignoran aquellas posiciones donde haya ceros. El peso del bit de signo en un número negativo viene dado por su valor negativo.

**EJEMPLO**

Determinar los valores decimales de los siguientes números binarios con signo expresados en complemento a 2 :
**(a)** 01010110 **(b)** 10101010

***Solución*** **(a)** Los bits y sus pesos según las potencias de dos para el número positivo son:

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

Sumando los pesos donde hay 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

**(b)** Los bits y sus pesos según las potencias de dos para el número negativo son los siguientes. Observe que el bit de signo negativo tiene un peso de $-2^7 = -128$.

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

Sumando los pesos donde hay 1s,

$$-128 + 32 + 8 + 2 = \mathbf{+86}$$

# Representación de números con punto flotante

Para representar números enteros muy grandes, son necesarios muchos bits. También surge un problema cuando se necesitan representar números con parte entera y parte fraccionaria, tal como 23, 5618. El sistema de numeración en coma flotante, basado en la notación científica, permite representar números muy grandes y números muy pequeños sin aumentar el número de bits, y también sirve para representar números con parte fraccionaria y parte entera. Un número en coma flotante (también conocido como número real) tiene dos partes más un signo. La mantisa es la parte del número en coma flotante que representa la magnitud del número. El exponente es la parte de un número en coma flotante que representa el número de lugares que se va a desplazar el punto decimal (o punto binario). Un ejemplo de número decimal le será útil para comprender el concepto básico de los números en coma flotante. Consideremos un número decimal que, en formato entero, es 241.506.800. La mantisa es .2415068 y el exponente es 9. Cuando el entero se expresa como un número en coma flotante, se normaliza desplazando el punto decimal a la izquierda de todos los dígitos, de modo que la mantisa es un número fraccionario y el exponente es una potencia de 10. Este número en coma flotante se escribe: $0{,}2415068 \times 10^9$

Para los números en coma flotante binarios, el formato puede tomar tres formas: simple precisión, doble precisión y precisión ampliada. Todos ellos utilizan el mismo formato básico excepto por el número de bits. Los números en coma flotante de simple precisión tienen 32 bits, los de doble precisión

tienen 64 bits y los de precisión ampliada tienen 80 bits. Vamos a restringir nuestra exposición al formato de los números en coma flotante de simple precisión.

## Números con punto flotante simple precisión

En el formato estándar para un número binario de simple precisión, el bit de signo (S) es el que se encuentra más a la izquierda, el exponente (E) incluye los siguientes 8 bits y la mantisa o parte fraccionaria (F) incluye los restantes 23 bits.

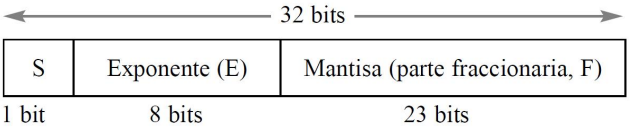



*Fig. 6. Esquema representativo de la organización del signo, exponente y mantisa en un número codificado con punto flotante simple precisión.*

En la mantisa o parte fraccionaria, se entiende que el punto binario estará a la izquierda de los 23 bits. Realmente, la mantisa consta de 24 bits, ya que, en cualquier número binario, el bit más a la izquierda (más significativo) es siempre un 1. Por tanto, este 1 se entiende que estará allí, aunque no ocupe una posición de bit real.

Los 8 bits de los que consta el exponente representan un exponente desplazado que se ha obtenido mediante la adición de 127 al exponente real. El propósito de este desplazamiento es poder definir números muy grandes o muy pequeños sin necesidad de emplear un bit de signo diferente para el exponente. El exponente desplazado permite emplear un rango de valores para los exponentes comprendido entre −126 y +128. Para ilustrar cómo un número binario se expresa en formato de coma flotante, utilizaremos como ejemplo el número binario 1011010010001. En primer lugar, podemos expresarlo como 1 más un número binario fraccionario, desplazando el punto binario doce posiciones a la izquierda y multiplicándolo después por la apropiada potencia de 2.

$1011010010001 = 1{,}011010010001 \times 2^{12}$

Suponiendo que se trate de un número positivo, el bit de signo (S) es 0. El exponente, 12, se expresa como un exponente desplazado añadiéndole 127 (12 + 127 = 139). El exponente desplazado (E) se expresa como el número binario 10001011. La mantisa es la parte fraccionaria (F) del número binario, 011010010001. Dado que siempre existe un 1 a la izquierda de la coma binaria en la expresión de la potencia de dos, no se incluye en la mantisa. El número en coma flotante completo es:

| S | E | F |
|---|---|---|
| 0 | 10001011 | 01101001000100000000000 |

A continuación, veamos cómo evaluar un número binario que ya está en formato de coma flotante. El método general para determinar el valor de un número en coma flotante se expresa mediante la siguiente fórmula:

$\text{Número} = (-1)^S(1 + F)(^{2E-127})$

Para ilustrar este método, consideremos el siguiente número binario en coma flotante:

| S | E | F |
|---|---|---|
| 1 | 10010001 | 10001110001000000000000 |

El bit de signo es 1. El exponente desplazado es 10010001 = 145. Aplicando la fórmula, obtenemos:

Número = $(-1)^1(1{,}10001110001)(2^{145-127})$

= $(-1)(1{,}10001110001)(2^{18}) = -1100011100010000000$

Este número binario en coma flotante es equivalente a −407.688 en decimal. Dado que el exponente puede ser cualquier número comprendido entre −126 y +128, pueden expresarse números extremadamente grandes y pequeños. Un número en coma flotante de 32 bits puede reemplazar a un número entero binario utilizando 129 bits. Dado que el exponente determina la posición del punto binario, se pueden representar números que contengan tanto parte entera como parte fraccionaria. Existen dos excepciones para el formato de los números en coma flotante: el número 0,0 se representa utilizando sólo ceros e infinito se representa utilizando sólo unos en el exponente y ceros en la mantisa.

# Sistema de numeración hexadecimal

El sistema hexadecimal es un sistema en base dieciséis, es decir, está formado por 16 caracteres numéricos y alfabéticos. La mayoría de los sistemas digitales procesan grupos de datos binarios que son múltiplos de cuatro bits, lo que hace al número hexadecimal muy adecuado, ya que cada dígito hexadecimal se representa mediante un número binario de 4 bits.

Diez dígitos numéricos y seis caracteres alfabéticos forman el sistema de numeración hexadecimal. El uso de las letras A, B, C, D, E y F para representar números puede parecer extraño al principio, pero tenga en mente que cualquier sistema de numeración es sólo un conjunto de símbolos secuenciales. Si comprende qué cantidades representan estos símbolos, entonces la forma de los símbolos en sí tiene poca importancia, una vez que se haya acostumbrado a utilizarlos. Utilizaremos el subíndice 16 para designar a los números hexadecimales y evitar así cualquier confusión con los números decimales. En ocasiones, puede ver la letra “h” detrás de un número hexadecimal.

Para seguir contando luego de la letra F, simplemente se inicia otra columna y se continúa contando así: 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31...

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

*Fig. 7. Tabla de equivalencias entre el sistema de numeración decimal, binario y hexadecimal.*

Con dos dígitos hexadecimales, se puede contar hasta FF16, que corresponde al decimal 255. Para continuar contando, se necesitan tres dígitos hexadecimales. Por ejemplo, 10016 es el decimal 256, 10116 es el decimal 257, y así sucesivamente. El número hexadecimal máximo con 3 dígitos es FFF16, es

decir el decimal 4.095. El máximo número hexadecimal con 4 dígitos es el FFFF16, que es el decimal 65.535.

## Conversión binario-hexadecimal

La conversión de un número binario en hexadecimal es un procedimiento muy sencillo. Simplemente se parte el número binario en grupos de 4 bits, comenzando por el bit más a la derecha, y se reemplaza cada grupo de 4 bits por su símbolo hexadecimal equivalente.

### EJEMPLO

Convertir a hexadecimal los siguientes números binarios:

**(a)** 1100101001010111 **(b)** 111111000101101001

**(a)** 1100 1010 0101 0111
C A 5 7 $= \mathbf{CA57}_{16}$

**(b)** 0011 1111 0001 0110 1001
3 F 1 6 9 $= \mathbf{3F169}_{16}$

En el apartado (b) se han añadido dos ceros para completar el grupo de 4 bits de la izquierda.

## Conversión hexadecimal-binario

Para convertir un número hexadecimal en un número binario se realiza el proceso inverso, reemplazando cada símbolo hexadecimal por el grupo de cuatro bits adecuado. Debería estar claro que es mucho más fácil tratar con un número hexadecimal que con el número binario equivalente. Puesto que la conversión también es fácil, el sistema hexadecimal se usa ampliamente para representar los números binarios en programación, salidas de impresora y displays.

### EJEMPLO

Determinar los números binarios correspondientes a los siguientes números hexadecimales:

**(a)** $10A4_{16}$ **(b)** $CF8E_{16}$ **(c)** $9742_{16}$

**(a)** 1 0 A 4
1 0000 1010 0100

**(b)** C F 8 E
1100 1111 1000 1110

**(c)** 9 7 4 2
1001 0111 0100 0010

En el apartado (a), el MSB se entiende que tiene tres ceros delante del 1 para formar un grupo de 4 bits.

## Conversión hexadecimal-decimal

Un método para encontrar el equivalente decimal de un número hexadecimal es, primero, convertir el número hexadecimal a binario, y después, el binario a decimal.

## EJEMPLO

Convertir los siguientes números hexadecimales a decimal:

**(a)** $1C_{16}$ **(b)** $A85_{16}$

***Solución*** Recuerde que primero se hace la conversión del número hexadecimal a binario y luego a decimal.

**(a)** $\overbrace{0001}^{1}\overbrace{1100}^{C} = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28}_{10}$

**(b)** $\overbrace{1010}^{A}\overbrace{1000}^{8}\overbrace{0101}^{5} = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = \mathbf{2693}_{10}$

Otro método para convertir un número hexadecimal a su equivalente decimal es multiplicar el valor decimal de cada dígito hexadecimal por su peso, y luego realizar la suma de estos productos. Los pesos de un número hexadecimal crecen según las potencias de 16 (de derecha a izquierda). Para un número hexadecimal de 4 dígitos, los pesos son:

| $16^3$ | $16^2$ | $16^1$ | $16^0$ |
|---|---|---|---|
| 4096 | 256 | 16 | 1 |

## EJEMPLO

Convertir los siguientes números hexadecimales a decimal:

**(a)** $E5_{16}$ **(b)** $B2F8_{16}$

***Solución*** En la Tabla 2.3 puede ver que las letras A hasta F representan los números decimales 10 hasta 15, respectivamente.

**(a)** $E5_{16} = (E \times 16) + (5 \times 1) = (14 \times 16) + (5 \times 1) + 224 + 5$

$= \mathbf{229}_{10}$

**(b)** $B2F8_{16} = (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1)$

$= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1)$

$= 45.056 + 512 + 240 + 8$

$= \mathbf{45.816}_{10}$

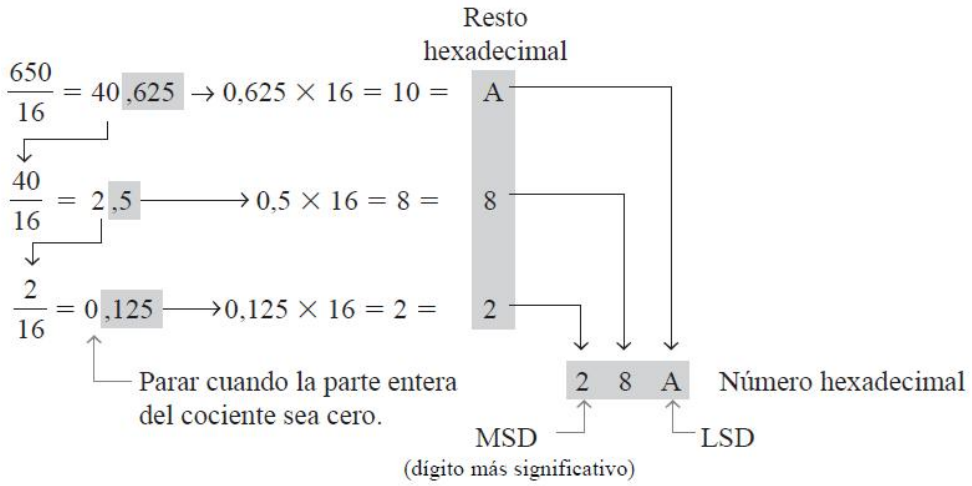
# Conversión decimal-hexadecimal

La división sucesiva por 16 de un número decimal generará el número hexadecimal equivalente formado por los restos de las divisiones. El primer resto que se genera es el dígito menos significativo (LSD). Cada división sucesiva por 16 dará un resto que será un dígito del número hexadecimal equivalente. Este procedimiento es similar a la división sucesiva por 2 para la conversión decimal–binario, que se ha visto en la Sección 2.3. El Ejemplo 2.28 ilustra el procedimiento. Observe que cuando un cociente tiene parte fraccionaria, ésta se multiplica por el divisor para obtener el resto.

**EJEMPLO**

Convertir el número decimal 650 en hexadecimal mediante el método del división sucesiva por 16.

***Solución***




# Sistema de numeración octal

Como el sistema hexadecimal, el sistema octal proporciona un método adecuado para expresar los códigos y números binarios. Sin embargo, se usa menos frecuentemente que el hexadecimal en las computadoras y microprocesadores para expresar magnitudes binarias con propósitos de entrada y salida.

El sistema de numeración octal está formado por ocho dígitos, que son: 0, 1, 2, 3, 4, 5, 6, 7. Para contar por encima de 7, añadimos otra columna y continuamos así: 10, 11, 12, 13, 14, 15, 16, 17, 20, 21. Contar en octal es parecido a contar en decimal, excepto que los dígitos 8 y 9 no se usan. Para distinguir los números octales de los números decimales y hexadecimales, utilizaremos el subíndice 8 para indicar un número octal. Por ejemplo, 158 es equivalente a 1310 en decimal y a D en hexadecimal. En ocasiones, puede ver una “o” o una “Q” detrás de un número octal.

## Conversión octal-decimal

Puesto que el sistema de numeración octal es un sistema en base ocho, cada posición sucesiva de dígito es una potencia superior de ocho, empezando por el dígito situado más a la derecha con 80. La evaluación de un número octal en términos de su equivalente decimal se consigue multiplicando cada dígito por su peso y sumando los productos, como se muestra a continuación para 23748:

Peso: $8^3\ 8^2\ 8^1\ 8^0$

Número octal: 2 3 7 4

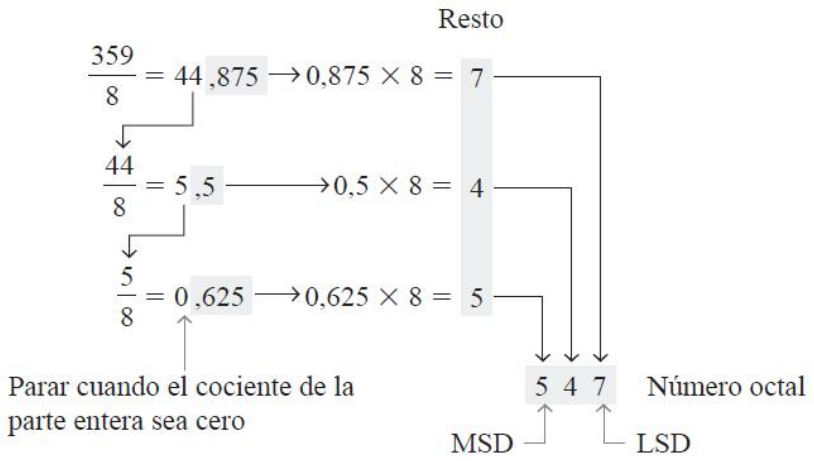
$2374_8 = (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0)$

$= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1)$

$= 1024 + 192 + 56 + 4 = 1276_{10}$

## Conversión decimal-octal

Un método para convertir un número decimal en un número octal es el método de la división sucesiva por 8, que es parecido al método utilizado en la conversión a binario o a hexadecimal de los números decimales. Para mostrar cómo se hace, convertimos a octal el número decimal 359. Cada división sucesiva por 8 da un resto que será un dígito del número octal equivalente. El primer resto que se genera es el dígito menos significativo (LSD).




## Conversión octal-binario

Puesto que cada dígito octal se puede representar mediante un número binario de 3 dígitos, es fácil convertir a binario un número octal. Cada dígito octal se representa mediante tres bits. Para convertir a binario un número octal basta con reemplazar cada dígito octal con los tres bits apropiados.

| **Dígito octal** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Binario** | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

*Fig. 8. Tabla de equivalencias entre los sistemas de numeración octal y binario.*

## Conversión binario-octal

La conversión de un número binario a un número octal es el inverso de la conversión de octal a binario. El procedimiento es el siguiente: se comienza por el grupo de tres bits más a la derecha y, moviéndose de derecha a izquierda, se convierte cada grupo de 3 bits en el dígito octal equivalente. Si para el grupo más a la izquierda no hay disponibles tres bits, se añaden uno o dos ceros para completar el grupo. Estos ceros no afectan al valor del número binario.

**EJEMPLO**

Convertir a octal cada uno de los siguientes números binarios:

**(a)** 110101 **(b)** 101111001 **(c)** 100110011010 **(d)** 11010000100

**(a)** $\underbrace{110}_{6}\underbrace{101}_{5} = 65_8$ **(b)** $\underbrace{101}_{5}\underbrace{111}_{7}\underbrace{001}_{1} = 571_8$

**(c)** $\underbrace{100}_{4}\underbrace{110}_{6}\underbrace{011}_{3}\underbrace{010}_{2} = 4632_8$ **(d)** $\underbrace{011}_{3}\underbrace{010}_{2}\underbrace{000}_{0}\underbrace{100}_{4} = 3204_8$

# Código decimal binario

El código decimal binario (BCD, Binary Coded Decimal) es una forma de expresar cada uno de los dígitos decimales con un código binario. Puesto que en el sistema BCD sólo existen diez grupos de código, es muy fácil convertir entre decimal y BCD. Como nosotros leemos y escribimos en decimal, el código BCD proporciona una excelente interfaz para los sistemas binarios. Ejemplos de estas interfaces son las entradas por teclado y las salidas digitales

El código 8421 es un tipo de código decimal binario (BCD). Código decimal binario significa que cada dígito decimal, de 0 hasta 9, se representa mediante un código binario de cuatro bits. La designación 8421 indica los pesos binarios de los cuatro bits ($2^3$, $2^2$, $2^1$, $2^0$). La facilidad de conversión entre los números en código 8421 y los familiares números decimales es la principal ventaja de este código. Todo lo que tiene que recordar sobre las diez combinaciones binarias que representan los diez dígitos decimales se muestra en la Tabla 2.5. El código 8421 es el código BCD más importante, y cuando hacemos referencia a BCD, siempre es al código 8421, a no ser que se indique otra cosa.

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

*Fig. 9. Tabla de equivalencias entre el sistema de numeración binario y el código decimal binario (BCD).*

# Códigos digitales

## Código de Gray

El código Gray es un código sin pesos y no aritmético; es decir, no existen pesos específicos asignados a las posiciones de los bits. La característica más importante del código Gray es que **sólo varía un bit de un código al siguiente.** Esta propiedad es importante en muchas aplicaciones, tales como los codificadores de eje de posición, en los que la susceptibilidad de error aumenta con el

| Decimal | Binario | Código Gray | Decimal | Binario | Código Gray |
|---|---|---|---|---|---|
| 0 | 0000 | 0000 | 8 | 1000 | 1100 |
| 1 | 0001 | 0001 | 9 | 1001 | 1101 |
| 2 | 0010 | 0011 | 10 | 1010 | 1111 |
| 3 | 0011 | 0010 | 11 | 1011 | 1110 |
| 4 | 0100 | 0110 | 12 | 1100 | 1010 |
| 5 | 0101 | 0111 | 13 | 1101 | 1011 |
| 6 | 0110 | 0101 | 14 | 1110 | 1001 |
| 7 | 0111 | 0100 | 15 | 1111 | 1000 |

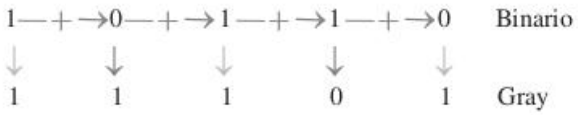
*Fig. 10. Tabla de equivalencias entre el código de gray, sistema decimal y binario.*

número de cambios de bit entre números adyacentes dentro de una secuencia.

Algunas veces, la conversión de código binario a código Gray resulta útil. Las siguientes reglas explican cómo convertir un número binario en un número en código Gray:

1. El bit más significativo (el que está más a la izquierda, MSB) en el código Gray es el mismo que el correspondiente MSB del número binario.
2. Yendo de izquierda a derecha, sumar cada par adyacente de los bits en código binario para obtener el siguiente bit en código Gray. Los acarreos deben descartarse.

Por ejemplo, la conversión del número binario 10110 a código Gray se hace del siguiente modo:




Para convertir de código Gray a binario, se utiliza un método similar, pero con algunas diferencias. Se aplican las siguientes reglas:

1. El bit más significativo (bit más a la izquierda) en el código binario es el mismo que el correspondiente bit en código Gray.
2. A cada bit del código binario generado se le suma el bit en código Gray de la siguiente posición adyacente. Los acarreos se descartan.

Por ejemplo, la conversión del número en código Gray 11011 a binario es como sigue:


## Código ASCII

El American Standard Code for Information Interchange (ASCII, código estándar americano para el intercambio de información) es un código alfanumérico universalmente aceptado, que se usa en la mayoría de las computadoras y otros equipos electrónicos. La mayor parte de los teclados de computadora se estandarizan de acuerdo con el código ASCII, y cuando se pulsa una letra, un número o un comando de control, es el código ASCII el que se introduce en la computadora.

El código ASCII dispone de 128 caracteres que se representan mediante un código binario de 7 bits. Realmente, el código ASCII puede considerarse como un código de 8 bits en el que el MSB siempre es 0. En hexadecimal, este código de 8 bits va de 00 hasta 7F. Los primeros 32 caracteres ASCII son comandos no gráficos, que nunca se imprimen o presentan en pantalla, y sólo se utilizan para propósitos de control. Ejemplos de caracteres de control son el carácter “nulo”, “avance de línea”, “inicio de texto” y “escape”. Los demás caracteres son símbolos gráficos que pueden imprimirse o mostrarse en pantalla, e incluyen las letras del alfabeto (mayúsculas y minúsculas), los diez dígitos decimales, los signos de puntuación y otros símbolos comúnmente utilizados.

Los primeros treinta y dos códigos de la tabla ASCII representan los caracteres de control. Estos se utilizan para permitir a dispositivos, tales como una computadora o una impresora, que se comuniquen entre sí cuando transfieren información y datos. La Tabla 2.8 enumera los caracteres de control y las funciones de las teclas de control que permiten introducir directamente el código ASCII a través del teclado, presionando la tecla control (CTRL) y el símbolo correspondiente. También se facilita una breve descripción de cada carácter de control.

Además de los 128 caracteres ASCII estándar, existen 128 caracteres adicionales que fueron adoptados por IBM para utilizar en sus computadoras personales (PC). Debido a la popularidad del PC, estos caracteres especiales del código ASCII extendido se usan también en otras aplicaciones distintas de los PC, por lo que se ha convertido en un estándar no oficial.

Los caracteres del código ASCII extendido se representan mediante una serie de códigos de 8 bits que van, en hexadecimal, del 80 hasta FF.

El código ASCII extendido está formado por caracteres que pertenecen a las siguientes categorías generales:

- Caracteres alfabéticos no ingleses.
- Símbolos de moneda no ingleses.
- Letras griegas
- Símbolos matemáticos
- Caracteres para gráficos
- Caracteres para gráficos de barras
- Caracteres sombreados

Aclaración: Ante la insuficiencia del ASCII para brindar un sistema de codificación que permita representar los distintos caracteres que se usan en todos los idiomas, aparece posteriormente el estándar UNICODE (del inglés «universal» y «code» - universal y código o sea código universal o unicódigo, del cual es parte UTF-8, entre otros) que constituye un estándar industrial cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático. Los primeros 128 caracteres de Unicode corresponden a los caracteres del ASCII.

# Detección de errores y códigos de corrección

Cuando se transmite información entre dispositivos electrónicos, por muchos motivos los datos que se transmiten pueden sufrir alteraciones. Estas alteraciones producen errores en el dato que será recibido incorrectamente por el receptor. Es por eso que existen distintos métodos para codificar los datos a enviar y para detectar posibles errores en la información recibida.

## Método de paridad para la detección de errores

Muchos sistemas emplean un bit de paridad como medio para la detección de errores de bit. Cualquier grupo de bits contiene un número par o impar de 1s. Un bit de paridad se añade al grupo de bits para hacer que el número total de 1s en el grupo sea siempre par o siempre impar. Un bit de paridad par hace que el número total de 1s sea par, y un bit de paridad impar hace que el número total de 1s del grupo sea impar.

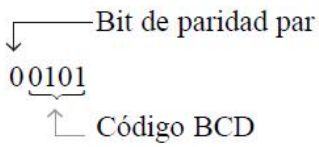
| Paridad par | | Paridad impar | |
|---|---|---|---|
| *P* | BCD | *P* | BCD |
| 0 | 0000 | 1 | 0000 |
| 1 | 0001 | 0 | 0001 |
| 1 | 0010 | 0 | 0010 |
| 0 | 0011 | 1 | 0011 |
| 1 | 0100 | 0 | 0100 |
| 0 | 0101 | 1 | 0101 |
| 0 | 0110 | 1 | 0110 |
| 1 | 0111 | 0 | 0111 |
| 1 | 1000 | 0 | 1000 |
| 0 | 1001 | 1 | 1001 |

*Fig. 11. En la tabla vemos los códigos de numeración BCD y el bit de paridad correspondiente a cada uno.*

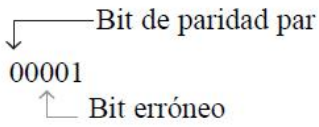
Un determinado sistema puede funcionar con paridad par o impar, pero no con ambas. Por ejemplo, si un sistema trabaja con paridad par, una comprobación que se realice en cada grupo de bits recibidos tiene que asegurar que el número total de 1s en ese grupo es par. Si hay un número impar de 1s, quiere decir que se ha producido un error.

El bit de paridad se puede añadir en cualquier lugar del código, dependiendo del diseño del sistema (generalmente se ubica al principio o al final). Observe que el número total de 1s, incluyendo el bit de paridad, siempre es par para paridad par, y siempre es impar para paridad impar.

Un bit de paridad facilita la detección de un único error de bit (o de cualquier número impar de errores, lo cual es muy improbable), pero no puede detectar dos errores dentro de un grupo. Por ejemplo, supongamos que deseamos transmitir el código BCD 0101 (el método de paridad puede usarse con cualquier número de bits, ahora usamos cuatro con propósitos de ilustración). El código total transmitido incluyendo el bit de paridad par es:




Supongamos ahora que se produce un error en el tercer bit de la izquierda (el 1 se transmite como 0).




Cuando se recibe este código, la circuitería de comprobación de paridad determina que sólo hay un 1 (impar), cuando debería haber un número par de 1s. Puesto que en el código recibido no aparece un número par de 1s, esto indica que se ha producido un error.

Un bit de paridad impar también facilita de forma similar la detección de un único error en un grupo de bits dado.

## Código de Hamming para detección y corrección de errores

Como hemos visto, un único bit de paridad permite detectar errores de un único bit en una palabra de código. Un único bit de paridad puede indicar si existe un error en un determinado grupo de bits. Para

corregir un error detectado, se necesita más información, ya que hay que identificar la posición del bit erróneo antes de poder corregirlo. Debe incluirse más de un bit de paridad en un grupo de bits para poder corregir el error detectado. En un código de 7 bits, existen siete posibles bits erróneos. En este caso, tres bits de paridad no sólo pueden detectar el error, sino que también pueden especificar la posición del bit erróneo. El código Hamming proporciona un método de corrección de un único bit erróneo. A continuación, se estudia la construcción de un código Hamming de 7 bits para corregir un único error.

Si el número de bits de datos se designa por d, entonces el número de bits de paridad, p, se determina mediante la siguiente relación $\mathbf{2^p \geq d + p + 1}$. Por ejemplo, si tenemos cuatro bits de datos, entonces *p* se calcula por el método de prueba y error usando la antes mencionada. Entonces,

$$\mathbf{2^p = 2^2 = 4} \text{ y}$$

y

$$\mathbf{d + p + 1 = 4 + 2 + 1 = 7}$$

Puesto que $\mathbf{2^p}$ tiene que ser igual o mayor que **d + p + 1**, la relación no se satisface. Probamos de nuevo, sea *p = 3*. Luego,

$$\mathbf{2^p = 2^3 = 8}$$

y

$$\mathbf{d + p + 1 = 4 + 3 + 1 = 8}$$

Este valor de p satisface la relación de la ecuación, por lo que se necesitan tres bits de paridad para poder corregir un único error en cuatro bits de datos. Debemos destacar que se proporciona la detección y corrección de errores para todos los bits, tanto de paridad como de datos, del grupo de códigos; es decir, los bits de paridad también se comprueban a sí mismos.

Ahora que ya sabemos cuál es el número necesario de bits de paridad en nuestro ejemplo, debemos colocar correctamente los bits dentro del código. Debe darse cuenta de que, en este ejemplo, el código está formado por cuatro bits de datos y tres bits de paridad. El bit más a la izquierda es el bit 1, el siguiente bit es el bit 2, y así sucesivamente: bit 1, bit 2, bit 3, bit 4, bit 5, bit 6, bit 7.

Los bits de paridad se sitúan en las posiciones que se han numerado haciéndolas corresponder con las potencias de dos en sentido ascendente (1, 2, 4, 8, ...), del modo siguiente:

$P_1$, $P_2$, $D_1$, $P_3$, $D_2$, $D_3$, $D_4$

El símbolo $P_n$ designa un determinado bit de paridad y $D_n$ designa cada uno de los bits de datos.

Para terminar, hay que asignar apropiadamente un valor de 1 o de 0 a cada uno de los bits de paridad. Puesto

| **Designación de bit** | $P_1$ | $P_2$ | $D_1$ | $P_3$ | $D_2$ | $D_3$ | $D_4$ |
|---|---|---|---|---|---|---|---|
| **Posición de bit** | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Número de posición en binario** | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Bits de datos ($D_n$) | | | | | | | |
| Bits de paridad ($P_n$) | | | | | | | |

*Fig. 12. Tabla de posiciones de bits para un código de corrección de errores de 7 bits.*

que cada bit de paridad proporciona una comprobación sobre los restantes bits del código total, tenemos que conocer el valor de dichos otros bits pares asignar el valor del bit de paridad. Para hallar los valores de los bits, primero expresamos en binario el número correspondiente a cada posición de bit; es decir, escribimos el número binario correspondiente a cada número decimal de posición (ver Fig. 23). A continuación, como se ilustra en la primera fila de la Fig. 23, indicamos las posiciones de los bits de paridad y de datos. Observe que el número binario de posición del bit de paridad $P_1$ tiene un 1 como su dígito más a la derecha. Este bit de paridad comprueba las posiciones de todos los bits, incluyéndose a sí mismo, que tienen 1s en la misma posición en el correspondiente número de posición en binario. Por tanto, el bit de paridad $P_1$ comprueba las posiciones de bits 1, 3, 5 y 7.

El número de posición en binario para el bit de paridad $P_2$ tiene un 1 en su posición intermedia. Este bit comprueba entonces todas las posiciones de bit, incluyéndose a sí mismo, que tienen un 1 en esa misma posición. Por tanto, el bit de paridad $P_2$ comprueba las posiciones de bit 2, 3, 6 y 7.

El número de posición en binario para el bit de paridad $P_3$ tiene un 1 como su bit más a la izquierda. Este bit comprueba entonces todas las posiciones de bit, incluyéndose a sí mismo, que tienen un 1 en esa misma posición. Por tanto, el bit de paridad $P_3$ comprueba las posiciones de bit 4, 5, 6 y 7.

En cada uno de los casos, se asigna un valor al bit de paridad de modo que la cantidad de 1s en el conjunto de bits que se desea comprobar sea impar o par, dependiendo de lo que se haya especificado. Los siguientes ejemplos clarificarán este procedimiento.

## EJEMPLO

Se recibe el código 101101010. Corregir los errores. Se emplean cuatro bits de paridad y el tipo de paridad impar.

*Solución* En primer lugar, construimos una tabla de posiciones de bits, como la mostrada en la Tabla 2.15.

| **Designación de bit** | $P_1$ | $P_2$ | $D_1$ | $P_3$ | $D_2$ | $D_3$ | $D_4$ | $P_4$ | $D_5$ |
|---|---|---|---|---|---|---|---|---|---|
| **Posición de bit** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Número de posición en binario** | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |
| Código recibido | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

*Primera comprobación de paridad*:
El bit $P_1$ comprueba las posiciones 1, 3, 5, 7 y 9.
En este grupo hay dos 1s.
La comprobación de paridad es incorrecta. ⟶ 1 (LSB)

*Segunda comprobación de paridad*:
El bit $P_2$ comprueba las posiciones 2, 3, 6 y 7.
En este grupo hay dos 1s.
La comprobación de paridad es incorrecta. ⟶ 1

*Tercera comprobación de paridad*:
El bit $P_3$ comprueba las posiciones 4, 5, 6 y 7.
En este grupo hay un 1.
La comprobación de paridad es incorrecta. ⟶ 1

*Cuarta comprobación de paridad*:
El bit $P_4$ comprueba las posiciones 8 y 9.
En este grupo hay un 1.
La comprobación de paridad es correcta. ⟶ 0 (MSB)

*Resultado:*
El código de posición de error es 0111 (siete en binario). Esto quiere decir que 1 bit situado en la posición 7 es erróneo. Por tanto, el código corregido es 101101110.