TEMA 2. Representación de la información.

Índice de contenido

Los datos. Tipos de datos y su representación.

Desde el principio, los ordenadores se construyeron pensando en manipular datos. La idea era crear máquinas que realizaran cálculos de una forma rápida y eficaz.

Desde aquel primer ordenador basado en relés que creó *George Stibitz* en *1937*, todos los que han venido después han representado la información a partir de elementos *biestables*, es decir, elementos que pueden tomar dos valores diferentes.

En realidad esta lógica se aplica a multitud de soportes con características completamente diferentes: si pasa o no corriente eléctrica por un determinado punto, si existe o no un campo magnético en un lugar concreto de la superficie de un *disco duro*, si un punto en particular de la superficie de un *DVD* refleja o no la luz láser, etc. Es decir, dentro de un ordenador, la información se guarda en función de dos posibles estados.

Por este motivo, desde un primer momento, el sistema de numeración que emplearon los ordenadores fue el *binario*.

A diferencia del sistema de numeración *decimal*, que utiliza diez símbolos diferentes para representar cualquier cantidad (0 a 9), el sistema *binario* utiliza sólo dos (el 0 y el 1).

La cantidad mínima de información que puede representar un ordenador es un *dígito binario* (en inglés, ***B**inary dig**it***), aunque para nombrarlo suele utilizarse el acrónimo *bit*.

Por lo tanto, la información almacenada en el interior de un ordenador estará representada por una secuencia de unos y ceros (más o menos larga, en función del dato almacenado), que estará codificada siguiendo unas normas particulares. A esto es a lo que llamamos *representación*.

Lógicamente la representación de la información dependerá del tipo al que ésta pertenezca. A continuación nombraremos los tipos de representación más frecuentes.

Sistemas de numeración y codificación de la información.

Como hemos dicho más arriba, los sistemas informáticos utilizan internamente el sistema de numeración *binario*, que está basado únicamente en dos dígitos (0 y 1). También hemos comentado que las personas utilizamos el sistema de numeración *decimal* que, como su nombre indica, se basa en el uso de diez dígitos (del 0 al 9)

En definitiva, un sistema de numeración es un conjunto de reglas y símbolos que nos permiten representar valores numéricos.

Esto significa que un determinado valor se representará de forma diferente según el sistema de numeración que estemos empleando. O al contrario, la misma representación puede equivaler a diferentes valores cuando usamos sistemas de numeración distintos.

Veamos un ejemplo:

Sistema de numeración	
Decimal	Binario
2	10
10	1010
220	11011100

Cuando mezclamos valores representados en diferentes sistemas de numeración, podemos indicar el sistema al que pertenece cada número añadiendo a su derecha un subíndice con el número de símbolos que utiliza dicho sistema. Por ejemplo:

$$220_{(10} = 11011100_{(2}$$

Tanto el sistema de numeración *decimal* como el *binario* (y el resto de los que usaremos en este capítulo) son posicionales. Esto significa que cada dígito representa un determinado valor que depende del propio dígito y de la posición que éste ocupa en el número. Para explicar esta idea, tomemos este número: $327_{(10}$

El dígito 7 representa 7 unidades porque se encuentra en la primera posición de la derecha. Sin embargo, el dígito 2 representa 2 decenas, es decir, 20 unidades, porque lo encontramos en la segunda posición, comenzando por la derecha. Del mismo modo, el dígito 3 representa 3 centenas (300 unidades), porque aparece en la tercera posición, comenzando por la derecha.

También existen sistemas de numeración antiguos que no eran posicionales, como el romano o el egipcio.

Todos los sistemas de numeración posicionales se rigen por el *Teorema fundamental de la numeración* que establece que el valor de un número viene dado por la suma de cada uno de sus dígitos multiplicados por su base elevada a la posición que ocupa dicho dígito, teniendo en cuenta que las unidades ocupan la posición 0.

$$N = \sum_{i=-d}^{i=n} dígito_i \times base^i$$

Siguiendo esta idea, podríamos representar el número anterior, usando su *expresión posicional*, de la siguiente forma:

$$327_{(10} = 3\times10^2 + 2\times10^1 + 7\times10^0$$

Convertir valores *binarios* a *decimal*

En Informática el *Teorema fundamental de la numeración* tiene aprovechamiento inmediato, ya que nos permite convertir un número representado en *binario* a *decimal* de una forma muy sencilla.

El truco está en escribir en *decimal* la expresión posicional del número en *binario*. De esta forma, al resolverla, obtendremos el resultado en *decimal*.

Veamos un ejemplo:

$$11011100_{(2} = 1\times2^7+1\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0$$
$$= 128 + 64 + 16 + 8 + 4 = 220_{(10}$$

Convertir valores *decimales* a *binarios*

Cuando necesitemos realizar la conversión en sentido contrario, podemos aplicar el método de las divisiones sucesivas.

Básicamente, este método consiste en realizar una división entera (es decir, sin obtener decimales) del número en *decimal* entre dos (la base a la que vamos a convertir). Una vez concluida la división, tomamos el cociente y volvemos a dividirlo entre dos. Así continuamos hasta que obtengamos un cociente igual a cero. En este punto, tomamos los restos de todas las divisiones en orden inverso a como han ido apareciendo (es decir, el último resto será el primero).

Como ejemplo, veamos la misma conversión del punto anterior, pero en el sentido contrario:

Dividendo	Divisor	Cociente	Resto
220	2	110	0
110	2	55	0
55	2	27	1
27	2	13	1
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

Los sistemas de numeración *octal* y *hexadecimal*

Una de las dificultades que plantea el trabajo con números *binarios* es la cantidad de dígitos que empleamos para representar cantidades relativamente pequeñas (en el ejemplo anterior hemos necesitado 8 dígitos *binarios* para representar el mismo valor que en *decimal* representamos con tres).

Por este motivo, en lugar de trabajar directamente en *binario*, es frecuente utilizar

otros sistema de numeración que utilicen un mayor número de símbolos.

Podríamos pensar que el sistema de numeración perfecto para este trabajo es el *decimal*, porque es al que estamos más acostumbrados. Sin embargo, los sistemas de numeración cuya base es una potencia de 2 tienen una ventaja añadida: la conversión entre ellos es automática.

Este es el motivo por el que suelen utilizarse los sistemas de numeración *octal* y *hexadecimal*.

El sistema de numeración *octal*

Como indica su nombre, el sistema de numeración *octal* utiliza ocho dígitos (0 a 7) para representar valores.

Como ya hemos dicho más arriba, la ventaja que representa *octal* sobre *decimal* es que utiliza una base que es potencia de 2 ($2^3 = 8$). En resumidas cuentas, esta característica nos va a permitir que cada tres caracteres en *binario* se puedan representar, directamente, con un carácter en *octal*.

Para comprenderlo, observemos la siguiente tabla que nos muestra valores en *octal* y en *binario*:

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

De esta forma, para convertir un número escrito en *binario* al sistema *octal*, bastaría con tomar los bits agrupados de tres en tres, comenzando por la derecha y completando con los ceros necesarios a la izquierda:

A continuación, buscamos cada grupo de tres dígitos en la tabla anterior y los sustituimos por su dígito correspondiente en la tabla:

Por lo tanto, el número en *binario* $11011100_{(2}$ corresponde con el número $334_{(8}$ en *octal*.

Resulta evidente que el paso de *octal* a *binario* es igual de sencillo.

Convertir valores de *octal* a *decimal*

La forma más sencilla de convertir un valor en *octal* a *decimal* es aplicar el *Teorema fundamental de la numeración* como ya hemos hecho con el *binario*. Como la expresión posicional del número en *octal* la escribimos en *decimal*, al resolverla, obtendremos el resultado en *decimal*.

Veamos un ejemplo:

$$334_{(8} = 3\times8^2+3\times8^1+4\times8^0$$
$$= 192 + 24 + 4 = 220_{(10}$$

Convertir valores de *decimal* a *octal*

Para realizar la conversión de un valor *decimal* a *octal*, podemos aplicar el método de las divisiones sucesivas que ya usamos con el *sistema binario*. Es decir, realizaremos la división entera (sin obtener decimales) del número en *decimal* entre ocho (la base a la que vamos a convertir). Una vez concluida la división, tomamos el cociente y volvemos a dividirlo entre ocho. Este proceso continuará hasta que obtengamos un cociente igual a cero.

En este punto, tomamos los restos de todas las divisiones en orden inverso a como han ido apareciendo (es decir, el último resto será el primero).

Como ejemplo, veamos la misma conversión del punto anterior, pero en el sentido contrario:

Dividendo	Divisor	Cociente	Resto
220	8	27	4
27	8	3	3
3	8	0	3

334

El sistema de numeración *hexadecimal*

El sistema de numeración *hexadecimal* utiliza dieciséis dígitos para representar valores. Como nosotros sólo utilizamos los dígitos del 0 al 9, y no son suficientes, en este caso se añaden las letras A a F para representar los dígitos que nos faltan. Por lo tanto, los dígitos que utilizamos para representar valores en *hexadecimal* son:

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

Ya hemos comentado más arriba que, como pasaba con el sistema *octal*, la ventaja que representa *hexadecimal* sobre *decimal* es que utiliza una base que es potencia de 2 (2^4 = 16). De forma similar a lo que ocurría en *octal*, esta característica nos va a permitir que cada cuatro caracteres en *binario* se puedan representar, directamente, con un carácter en *hexadecimal*.

Para comprobarlo, vamos a partir de la tabla que nos muestra valores en *hexadecimal* y en *binario*:

Binario	Octal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

De esta forma, para convertir un número escrito en *binario* al sistema *hexadecimal*, bastaría con tomar los bits agrupados de cuatro en cuatro, comenzando por la derecha y completando con los ceros necesarios a la izquierda si fuese necesario:

$11011100_{(2}$

A continuación, buscamos cada grupo de cuatro dígitos en la tabla anterior y los sustituimos por su dígito correspondiente en la tabla:

$11011100_{(2}$
D C $_{(16}$

Por lo tanto, el número en *binario* $11011100_{(2}$ corresponde con el número $DC_{(16}$ en *hexadecimal*.

Como puedes suponer, el paso de *hexadecimal* a *binario* es igual de sencillo.

Convertir valores de *hexadecimal* a *decimal*

Como en el caso del *binario* y el *octal*, la forma más sencilla de convertir un valor en *hexadecimal* a *decimal* es aplicar el *Teorema fundamental de la numeración*. Como antes, la expresión posicional del número en *hexadecimal* la escribimos en *decimal*. De este modo, al resolverla, obtendremos el resultado en *decimal*.

Para entender el siguiente ejemplo, debemos tener en cuenta el valor, en *decimal*, de los últimos 6 dígitos *hexadecimal*es:

Hexadecimal	Decimal
A	10
B	11
C	12
D	13
E	14
F	15

Veamos el ejemplo:

$$DC_{(16} = 13 \times 16^{1} + 12 \times 16^{0}$$
$$= 208 + 12 = 220_{(10}$$

Convertir valores de *decimal* a *hexadecimal*

Para realizar la conversión de un valor *decimal* a *hexadecimal*, aplicaremos de nuevo el método de las divisiones sucesivas que hemos utilizado con anterioridad. Es decir, realizaremos la división entera (sin obtener decimales) del número en *decimal* entre dieciséis (la base a la que vamos a convertir). Una vez concluida la división, tomamos el cociente y volvemos a dividirlo entre dieciséis. Este proceso continuará hasta que obtengamos un cociente igual a cero.

Llegado este momento, tomamos los restos de todas las divisiones en orden inverso a como han ido apareciendo (es decir, el último resto será el primero), acordándonos de sustituir aquellos valores que sean mayores que 9 por su dígito correspondiente en *hexadecimal*.

Como ejemplo, veamos la misma conversión del punto anterior, pero en el sentido contrario:

Dividendo	Divisor	Cociente	Resto
220	16	13	12 (C)
13	16	0	13 (D)

DC

Operaciones en binario

En el sistema binario se pueden realizar una serie de operaciones:

- **Operaciones aritméticas**, que son las mismas que se pueden realizar en el sistema decimal, como suma, resta, multiplicación, división,…
- **Operaciones lógicas**, que utilizan operadores lógicos como NOT, AND, OR,…

Operaciones aritméticas

En el sistema binario se puede realizar, al igual que en el sistema decimal, cualquier operación aritmética. Veremos las más básicas, como son la suma, resta, multiplicación y división.

Suma binaria

Para aprender a sumar, con cinco o seis años de edad, tuviste que memorizar las 100 combinaciones posibles que pueden darse al sumar dos dígitos decimales. La tabla de sumar, en binario, es mucho más sencilla que en decimal. Sólo hay que recordar cuatro combinaciones posibles:

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	0 ac 1

Las sumas 0 + 0, 0 + 1 y 1 + 0 son evidentes:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

Pero la suma de 1+1, que sabemos que es 2 en el sistema decimal, debe escribirse en binario con dos cifras (10) y, por tanto 1+1 es 0 y se arrastra una unidad, que se suma a la posición siguiente a la izquierda.

A continuación podemos ver un ejemplo:

```
    1 1 1 1   1   1
    1 0 0 1 1 0 1 0 1
+
        1 1 0 1 0 1 0 1
  ---------------------
  1 0 0 0 0 0 1 0 1 0
```

Resta binaria

En la resta binaria tenemos tres casos bastante claros. El caso que suele conllevar más dudas es en el que se resta 0 – 1. A continuación se muestra la tabla de posibilidades y un ejemplo ilustrativo de esta operación:

X	Y	X-Y
0	0	0
0	1	1 ac 1
1	0	1
1	1	0

```
    1 0 1 0 0 1   ← Minuendo
      1 0 1 1     ← Sustraendo
  - 1 1 1 1       ← Acarreos
  ---------------
    0 1 1 1 1 0   ← Resultado
```

Resta binaria

Multiplicación binaria

La multiplicación binaria guarda una gran similitud con la multiplicación decimal. El hecho de que el binario solo tenga dos símbolos simplifica mucho dicha operación.

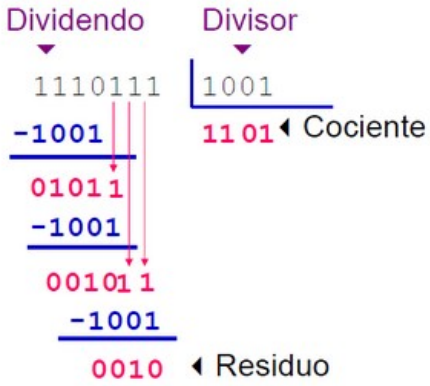
X	Y	X*Y
0	0	0
0	1	0
1	0	0
1	1	1

```
          1 0 1 1 0
  ×         1 0 0 1
  ---------------------
          1 0 1 1 0
        0 0 0 0 0
      0 0 0 0 0
  1 0 1 1 0
  ---------------------
  1 1 0 0 0 1 1 0
```

Multiplicación binaria

División binaria

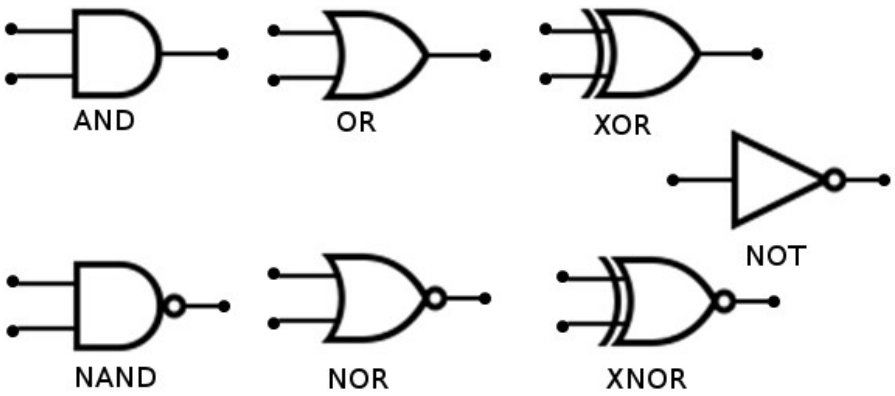
La división binaria hace uso de las restas para llevar a cabo dicha operación. Para comenzar se restará el divisor a la misma cantidad de cifras del dividendo. Por cada resta se añade un 1 al cociente y se baja la siguiente cifra del dividendo. Si no es posible la resta se añade un 0 al cociente y se baja la siguiente cifra del dividendo. En el siguiente ejemplo se podrá apreciar de una forma más clara.



Operaciones lógicas

Existen una serie de operaciones lógicas que son las que se suelen implementar en las denominadas puertas lógicas que forman los circuitos integrados de los ordenadores. A continuación veremos las tablas de opciones que representarán a dichas operaciones:

X	Y	NOT X	AND	OR	XOR	NAND	NOR	XNOR
0	0	1	0	0	0	1	1	1
0	1	1	0	1	1	1	0	0
1	0	0	0	1	1	1	0	0
1	1	0	1	1	0	0	0	1



TEMA 2. Representación de la información.

Códigos de E/S

Un código es una representación unívoca de las cantidades, de forma que a cada una de estas se le asigna una combinación de símbolos determinados y viceversa. Existen una serie de códigos dependiendo de la información que se vaya a representar. No es lo mismo representar números que caracteres alfanuméricos.

Representaciones numéricas

a) **Números naturales (0 al infinito)**

- **Representación binaria básica**: Se utilizan tantos bits como sean necesarios para representar el número. Con "n" bits se pueden representar 2^n-1 números distintos.
- **BCD**: Decimal codificado en binario. Utiliza un cuarteto o nibble (4 bits) para representar cada cifra decimal. Existen dos variantes que son la empaquetada (en un byte se representan dos cifras) y la desempaquetada (el primer nibble se pone a 1111 y en el segundo nibble se representa la cifra numérica).

b) Números enteros

Signo y magnitud: Se representa el número en representación binaria básica y se añade un bit a la derecha del número que indicará el signo (1 negativo y 0 positivo). El problema de esta representación es que el 0 tiene dos representaciones diferentes (la negativa y la positiva).

Representaciones alfanuméricas

Permiten representar números, letras y caracteres especiales.

- **FIELDATA**: Usa 6 bits y permite representar 64 caracteres. Se usaba para letras mayúsculas, números y caracteres especiales.
- **ASCII**: Código estándar americano que usa 7 bits para representar 128 caracteres.
- **ASCII Extendido**: Mejora del ASCII que aumenta a 8 bits para representar 256 caracteres.
- **EBCDIC** (Extended Binary Coded Decimal Interchange Code): 8 bits.
- **UNICODE**: Utiliza 16 bits para representar todos los símbolos de todos los idiomas

del mundo. Es independiente de la plataforma.

Detección y Corrección de Errores.

Un código redundante es aquel que suministra más información de la necesaria para poder detectar y corregir errores. A continuación vemos los métodos de detección y corrección de errores más importantes:

a) **Códigos de Paridad (Detector)**: Es un código detector de errores que permite controlar errores de 1 bit, aunque no los corrige y tampoco permite saber cuál es el bit que ha cambiado. Estos códigos añaden un bit (de paridad) a las combinaciones del código seleccionado.

b) **Código de Huffman (Detecta y Corrige errores)**: Emplea códigos de longitud variable. El número de bits para codificar un carácter depende de la frecuencia de aparición.

c) **Código de Hamming (Detecta y Corrige errores)**: Permite detectar errores de dos bits y corrige errores de 1 bit. Al código a proteger hay que añadirle un conjunto de bits para detectar diferentes paridades.

Representación de información Multimedia

Se llama información multimedia a aquella compuesta por imágenes, sonido y video. Al igual que el resto de información, esta información también puede ser procesada por un ordenador, por lo que hay diferentes formas de procesarla:

a) **Representación de una imagen**: Una imagen está formada por una matriz de puntos (pixels). Hay dos formas de representarla en un ordenador:

   a. **Imagen Raster**: Se almacena información sobre cada píxel. Cuanta más información se guarde mayor es el tamaño que ocupa. Cuando se escala una imagen raster, pierde calidad. Los formatos JPEG y BMP son los más conocidos.

   b. **Imagen vectorial**: Se almacenan vectores y figuras geométricas de la imagen. Al escalarse no pierde calidad y el tamaño que ocupa es reducido. Los formatos Flash y VML son los más conocidos para imágenes vectoriales.

b) **Representación del sonido**: Se hace mediante diferentes formatos, como pueden ser el MIDI, MOD, MP3 u OGG. MP3 es el más conocido y consigue una alta compresión a costa de sacrificar algo de sonido.

c) **Representación de Video**: Se usan para ello diferentes formatos como el formato MPEG, MP4 o Quicktime.

Compresión de la Información.

Con el fin de reducir el tamaño de los datos en las transmisiones existen algoritmos que comprimen la información:

a) <u>Compresión lossless (sin pérdida):</u>

a. **Compresores estadísticos**: aritméticos y predictivos.

b. **Compresores sustitucionales**: RLE y Lempel-Ziv.

c. **Compresores híbridos**: Una mezcla de los dos anteriores.

b) **<u>Compresión Lossy (con pérdida)</u>**: Usados para codificar archivos multimedia

a. **Codificación diferencial**.

b. **Cuantización de vectores**.

c. **Técnicas de compresión de imágenes en movimiento** (interframe).