# Capítulo 1

# Introducción

*—¿Qué sabes de este asunto?— preguntó el Rey a Alicia.*
*—Nada— dijo Alicia.*
*—¿*Absolutamente *nada?— insistió el Rey.*
*—Absolutamente nada— dijo Alicia.*
*—Esto es importante— dijo el Rey, volviéndose hacia los jurados.*

Lewis Carroll, *Alicia en el país de la maravillas*.

El objetivo de este curso es enseñarte a *programar*, esto es, a diseñar *algoritmos* y expresarlos como *programas* escritos en un *lenguaje de programación* para poder *ejecutarlos* en un *computador*.

Seis términos técnicos en el primer párrafo. No está mal. Vayamos paso a paso: empezaremos por presentar en qué consiste, básicamente, un computador.

## 1.1. Computadores

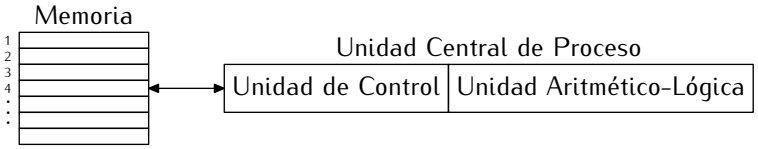
El diccionario de la Real Academia define computador electrónico como «Máquina electrónica, analógica o digital, dotada de una memoria de gran capacidad y de métodos de tratamiento de la información, capaz de resolver problemas matemáticos y lógicos mediante la utilización automática de programas informáticos.»

La propia definición nos da indicaciones acerca de algunos elementos básicos del computador:

- la memoria,
- y algún dispositivo capaz de efectuar cálculos matemáticos y lógicos.

La memoria es un gran almacén de información. En la memoria almacenamos todo tipo de datos: valores numéricos, textos, imágenes, etc. El dispositivo encargado de efectuar operaciones matemáticas y lógicas, que recibe el nombre de *Unidad Aritmético-Lógica* (UAL), es como una calculadora capaz de trabajar con esos datos y producir, a partir de ellos, nuevos datos (el resultado de las operaciones). Otro dispositivo se encarga de transportar la información de la memoria a la UAL, de controlar a la UAL para que efectúe las operaciones pertinentes y de depositar los resultados en la memoria: la *Unidad de Control*. El conjunto que forman la Unidad de Control y la UAL se conoce por *Unidad Central de Proceso* (o CPU, del inglés «Central Processing Unit»).

Podemos imaginar la memoria como un armario enorme con cajones numerados y la CPU como una persona que, equipada con una calculadora (la UAL), es capaz de buscar operandos en la memoria, efectuar cálculos con ellos y dejar los resultados en la memoria.




Utilizaremos un lenguaje más técnico: cada uno de los «cajones» que conforman la memoria recibe el nombre de *celda* (de memoria) y el número que lo identifica es su *posición* o *dirección*, aunque a veces usaremos estos dos términos para referirnos también a la correspondiente celda.

Cada posición de memoria permite almacenar una secuencia de unos y ceros de tamaño fijo. ¿Por qué unos y ceros? Porque la tecnología actual de los computadores se basa en la sencillez con que es posible construir dispositivos binarios, es decir, que pueden adoptar dos posibles estados: encendido/apagado, hay corriente/no hay corriente, cierto/falso, uno/cero... ¿Es posible representar datos tan variados como números, textos, imágenes, etc. con sólo unos y ceros? La respuesta es sí (aunque con ciertas limitaciones). Para entenderla mejor, es preciso que nos detengamos brevemente a considerar cómo se representa la información con valores binarios.

## 1.2. Codificación de la información

Una codificación asocia signos con los elementos de un conjunto a los que denominamos *significados*. En occidente, por ejemplo, codificamos los números de cero a nueve con el conjunto de signos $\{0,1,2,3,4,5,6,7,8,9\}$. Al hacerlo, ponemos en correspondencia estos símbolos con cantidades, es decir, con su significado: el símbolo «6» representa a la cantidad seis. El conjunto de signos no tiene por qué ser finito. Podemos combinar los dígitos en secuencias que ponemos en correspondencia con, por ejemplo, los números naturales. La sucesión de dígitos «99» forma un nuevo signo que asociamos a la cantidad noventa y nueve. Los ordenadores sólo tienen dos signos básicos, $\{0,1\}$, pero se pueden combinar en secuencias, así que no estamos limitados a sólo dos posibles significados.

Una variable que sólo puede tomar uno de los dos valores binarios recibe el nombre de *bit* (acrónimo del inglés «binary digit»). Es habitual trabajar con secuencias de bits de tamaño fijo. Una secuencia de 8 bits recibe el nombre de *byte* (aunque en español el término correcto es *octeto*, éste no acaba de imponerse y se usa la voz inglesa). Con una secuencia de 8 bits podemos representar 256 ($2^8$) significados diferentes. El rango $[0,255]$ de valores naturales comprende 256 valores, así que podemos representar cualquiera de ellos con un patrón de 8 bits. Podríamos decidir, en principio, que la correspondencia entre bytes y valores naturales es completamente arbitraria. Así, podríamos decidir que la secuencia `00010011` representa, por ejemplo, el número natural 0 y que la secuencia `01010111` representa el valor 3. Aunque sea posible esta asociación arbitraria, no es deseable, pues complica enormemente efectuar operaciones con los valores. Sumar, por ejemplo, obligaría a tener memorizada una tabla que dijera cuál es el resultado de efectuar la operación con cada par de valores, ¡y hay 65536 pares diferentes!

Los sistemas de representación posicional de los números permiten establecer esa asociación entre secuencias de bits y valores numéricos naturales de forma sistemática. Centramos el discurso en secuencias de 8 bits, aunque todo lo que exponemos a continuación es válido para secuencias de otros tamaños[1]. El valor de una cadena de bits $b_7b_6b_5b_4b_3b_2b_1b_0$ es, en un sistema posicional convencional, $\sum_{i=0}^{7} b_i \cdot 2^i$. Así, la secuencia de bits `00001011` codifica el valor $0\cdot 2^7+0\cdot 2^6+0\cdot 2^5+0\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0 = 8+2+1=11$. El bit de más a la izquierda recibe el nombre de «bit más significativo» y el bit de más a la derecha se denomina «bit menos significativo».

[1] Ocho bits ofrecen un rango de valores muy limitado. Es habitual en los ordenadores modernos trabajar con grupos de 4 bytes (32 bits) u 8 bytes (64 bits).

EJERCICIOS

▶ 1 ¿Cuál es el máximo valor que puede representarse con 16 bits y un sistema de representación posicional como el descrito? ¿Qué secuencia de bits le corresponde?

▶ 2 ¿Cuántos bits se necesitan para representar los números del 0 al 18, ambos inclusive?

El sistema posicional es especialmente adecuado para efectuar ciertas operaciones aritméticas. Tomemos por caso la suma. Hay una «tabla de sumar» en binario que te mostramos a continuación:

| sumandos | | suma | acarreo |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

El acarreo no nulo indica que un dígito no es suficiente para expresar la suma de dos bits y que debe añadirse el valor uno al bit que ocupa una posición más a la izquierda. Para ilustrar la sencillez de la adición en el sistema posicional, hagamos una suma de dos números de 8 bits usando esta tabla. En este ejemplo sumamos los valores 11 y 3 en su representación binaria:

```
  00001011
+ 00000011
----------
```

Empezamos por los bits menos significativos. Según la tabla, la suma 1 y 1 da 0 con acarreo 1:

```
Acarreo       1
  00001011
+ 00000011
----------
         0
```

El segundo dígito empezando por derecha toma el valor que resulta de sumar a 1 y 1 el acarreo que arrastramos. O sea, 1 y 1 es 0 con acarreo 1, pero al sumar el acarreo que arrastramos de la anterior suma de bits, el resultado final es 1 con acarreo 1:

```
Acarreo     1 1
  00001011
+ 00000011
----------
        10
```

Ya te habrás hecho una idea de la sencillez del método. De hecho, ya lo conoces bien, pues el sistema de numeración que aprendiste en la escuela es también posicional, sólo que usando diez dígitos diferentes en lugar de dos, así que el procedimiento de suma es esencialmente idéntico. He aquí el resultado final, que es la secuencia de bits 00001110, o sea, el valor 14:

```
Acarreo     1 1
  00001011
+ 00000011
----------
  00001110
```

La circuitería electrónica necesaria para implementar un sumador que actúe como el descrito es extremadamente sencilla.

EJERCICIOS

▶ 3 Calcula las siguientes sumas de números codificados con 8 bits en el sistema posicional:

a) `01111111 + 00000001` b) `01010101 + 10101010` c) `00000011 + 00000001`

Debes tener en cuenta que la suma de dos números de 8 bits puede proporcionar una cantidad que requiere 9 bits. Suma, por ejemplo, las cantidades 255 (en binario de 8 bits es `11111111`) y 1 (que en binario es `00000001`):

```
Acarreo   1 1 1 1 1 1 1
          11111111
      +   00000001
         (1)00000000
```

El resultado es la cantidad 256, que en binario se expresa con 9 bits, no con 8. Decimos en este caso que la suma ha producido un *desbordamiento*. Esta anomalía debe ser tenida en cuenta cuando se usa o programa un ordenador.

Hasta el momento hemos visto cómo codificar valores positivos. ¿Podemos representar también cantidades negativas? La respuesta es sí. Consideremos brevemente tres formas de hacerlo. La primera es muy intuitiva: consiste en utilizar el bit más significativo para codificar el signo; si vale `0`, por ejemplo, el número expresado con los restantes bits es positivo (con la representación posicional que ya conoces), y si vale `1`, es negativo. Por ejemplo, el valor de `00000010` es 2 y el de `10000010` es $-2$. Efectuar sumas con valores positivos y negativos resulta relativamente complicado si codificamos así el signo de un número. Esta mayor complicación se traslada también a la circuitería necesaria. Mala cosa.

Una forma alternativa de codificar cantidades positivas y negativas es el denominado «complemento a uno». Consiste en lo siguiente: se toma la representación posicional de un número (que debe poder expresarse con 7 bits) y se invierten todos sus bits si es negativo. La suma de números codificados así es relativamente sencilla: se efectúa la suma convencional y, si no se ha producido un desbordamiento, el resultado es el valor que se deseaba calcular; pero si se produce un desbordamiento, la solución se obtiene sumando el valor 1 al resultado de la suma (sin tener en cuenta ya el bit desbordado). Veámoslo con un ejemplo. Sumemos el valor 3 al valor $-2$ en complemento a uno:

```
Acarreo   1 1 1 1 1 1 1
          00000011
      +   11111101
         (1)00000000
                ↓
          00000000
      +   00000001
          00000001
```

La primera suma ha producido un desbordamiento. El resultado correcto resulta de sumar una unidad a los 8 primeros bits.

La codificación en complemento a uno tiene algunas desventajas. Una de ellas es que hay dos formas de codificar el valor 0 (con 8 bits, por ejemplo, tanto `00000000` como `11111111` representan el valor 0) y, por tanto, sólo podemos representar 255 valores ($[-127, 127]$), en lugar de 256. Pero el principal inconveniente es la lentitud con que se realizan operaciones como la suma: cuando se produce un desbordamiento se han de efectuar dos adiciones, es decir, se ha de invertir el doble de tiempo.

Una codificación alternativa (y que es la utilizada en los ordenadores) es la denominada «complemento a dos». Para cambiar el signo a un número hemos de invertir todos sus bits *y sumar* 1 *al resultado*. Esta codificación, que parece poco natural, tiene las ventajas de que sólo hay una forma de representar el valor nulo (el rango de valores representados

es $[-128, 127]$) y, principalmente, de que una sola operación de suma basta para obtener el resultado correcto de una adición. Repitamos el ejemplo anterior. Sumemos 3 y $-2$, pero en complemento a dos:

$$\begin{array}{rr} \text{Acarreo} & 1\ 1\ 1\ 1\ 1\ 1 \\ & 00000011 \\ + & 11111110 \\ \hline & (1)00000001 \end{array}$$

Si ignoramos el bit desbordado, el resultado es correcto.

EJERCICIOS

▶ 4 Codifica en complemento a dos de 8 bits los siguientes valores:

a) 4 b) $-4$ c) 0 d) 127 e) 1 f) $-1$

▶ 5 Efectúa las siguientes sumas y restas en complemento a dos de 8 bits:

a) $4+4$ b) $-4+3$ c) $127-128$ d) $128-127$ e) $1-1$ f) $1-2$

Bueno, ya hemos hablado bastante acerca de cómo codificar números (aunque más adelante ofreceremos alguna reflexión acerca de cómo representar valores con parte fraccional). Preocupémonos por un instante acerca de cómo representar texto. Hay una tabla que pone en correspondencia 127 símbolos con secuencias de bits y que se ha asumido como estándar. Es la denominada tabla ASCII, cuyo nombre son las siglas de «American Standard Code for Information Interchange». La correspondencia entre secuencias de bits y caracteres determinada por la tabla es arbitraria, pero aceptada como estándar. La letra «a», por ejemplo, se codifica con la secuencia de bits `01100001` y la letra «A» se codifica con `01000001`. En el apéndice A se muestra esta tabla. El texto se puede codificar, pues, como una secuencia de bits. Aquí tienes el texto «`Hola`» codificado con la tabla ASCII:

01001000 01101111 01101100 01100001

Pero, cuando vemos ese texto en pantalla, no vemos una secuencia de bits, sino la letra «H», seguida de la letra «o»,... Lo que realmente vemos es un gráfico, un patrón de píxeles almacenado en la memoria del ordenador y que se muestra en la pantalla. Un bit de valor 0 puede mostrarse como color blanco y un bit de valor 1 como color negro. La letra «H» que ves en pantalla, por ejemplo, es la visualización de este patrón de bits:

```
01000010
01000010
01000010
01111110
01000010
01000010
01000010
```

En la memoria del ordenador se dispone de un patrón de bits para cada carácter[2]. Cuando se detecta el código ASCII `01001000`, se muestra en pantalla el patrón de bits correspondiente a la representación gráfica de la «H». Truculento, pero eficaz.

No sólo podemos representar caracteres con patrones de píxeles: todos los gráficos de ordenador son simples patrones de píxeles dispuestos como una matriz.

Como puedes ver, basta con ceros y unos para codificar la información que manejamos en un ordenador: números, texto, imágenes, etc.

[2] La realidad es cada vez más compleja. Los sistemas modernos almacenan los caracteres en memoria de otra forma, pero hablar de ello supone desviarnos mucho de lo que queremos contar.