

# 1

---

# Introducción y Conceptos Básicos

---

## Contenido

<b>1. Introducción y conceptos básicos</b>	<b>1</b>
1.0.1. Definición de Informática . . . . .	1
1.0.2. Evolución histórica . . . . .	2
1.1. Ámbito de aplicación . . . . .	4
1.2. Tipos de computadores . . . . .	4
1.3. La máquina de von Neumann . . . . .	6
1.4. El funcionamiento del computador digital . . . . .	8
1.4.1. Sistemas de numeración . . . . .	8
1.4.2. Medida de volúmenes de información binaria . . . . .	12
1.4.3. Sistemas de codificación de la información . . . . .	12
1.4.4. Programas e instrucciones . . . . .	15
1.4.5. El funcionamiento de un programa . . . . .	18
1.5. El software básico de un sistema . . . . .	19
1.5.1. Niveles de abstracción . . . . .	19
1.5.2. El Sistema Operativo . . . . .	19
1.5.3. Tipos de programas . . . . .	19
1.6. Apéndice: Representación de números . . . . .	20
1.7. Apéndice: Historia del transistor . . . . .	26
1.8. Ejercicios . . . . .	28
1.9. Referencias de consulta . . . . .	29

## 1. Introducción y conceptos básicos

En este primer tema se introducen los conceptos más elementales de informática, su definición y ámbito, uso actual; se estudian los sistemas que el ordenador utiliza para codificar la información, lo que nos dará una idea de cómo trabajan los ordenadores con los datos. Se presenta asimismo la historia de la informática.

Se ofrecen Apéndices sobre conversión entre bases de sistemas de numeración, así como representación de números enteros positivos, con signo y en punto flotante, incluyendo el formato para representar números reales en forma digital, IEEE p754.

Un apéndice final presenta una amena historia del transistor, “50 años después”.

### 1.0.1. Definición de Informática

La *informática* trata de la adquisición, tratamiento y transmisión de la información en forma automática mediante computadores. La palabra *informática* proviene del acrónimo

$$\text{Informática} = \text{Información} + \text{automática}$$

también es llamada *Ciencia* o *Ingeniería* de la Computación. De hecho es éste el nombre más común para recoger la amplia variedad de áreas de estudio en las que se apoya el proceso automático de datos y la informática.

Existen muchas definiciones de Informática, pero lo importante es que se recogen en ella dos aspectos:

**información** El qué es la información y el qué no lo es, ralla los terrenos de la filosofía. En general entendemos que información es algún tipo de comunicación basada en conocimientos anteriores pero que aporta algo nuevo. La información implica forma y contenido. En cuanto a la forma de presentar la información se hace más o menos asequible y puede ofrecer nuevas sugerencias ya que la forma de presentar la información sugiere nuevas relaciones entre los elementos. En este sentido es interesante la lectura de [PC86].

**proceso** Todo lo que implique un proceso de la forma de presentación de la información o que permita la adquisición (mediante la comunicación con otros) de la misma, puede en muchos casos hacerse automáticamente y así, susceptible de ser resuelto en Informática.<sup>1</sup>

Así pues, la informática podríamos decir que es el conjunto de disciplinas que llevan a los ordenadores aquellos procesos susceptibles de ser descritos mediante pasos automáticos, aportando los computadores la precisión y velocidad de los mecanismos electrónicos.

Las *Ciencias de la Computación* serían todas aquellas áreas científicas relacionadas con la *construcción, verificación y análisis* de programas informáticos. El número de éstas es muy grande ya que desde la lógica, la matemática discreta, teoría de grafos, psicología, sociología, geometría, etc., hasta aquéllas relacionadas con los infinitos campos de aplicación, participan de su evolución y aportan a ella sus resultados. Hoy por hoy, el lenguaje informático se ha convertido en un nuevo lenguaje con el que se llegan a describir directamente problemas científicos.

### 1.0.2. Evolución histórica

Veremos en este apartado los orígenes históricos y principales problemas teóricos planteados en la computación.

El origen de las *Ciencias de la Computación* se encuentra en el desarrollo de “máquinas programables de propósito general”. Su objetivo inicial fue el resolver problemas esencialmente numéricos mediante tales máquinas.

**El ábaco** Desde hace muchos años se han ideado muchos artilugios para mecanizar y ayudarnos con los cálculos. Quizás el más antiguo sea el **ábaco** que consiste en una serie de alambres con cuentas insertadas. Pueden tenerse dos filas de cuentas y más o menos alambres. El modelo más sencillo tiene una fila de cuentas con una sola cuenta en cada alambre que indica el un dígito entre 5 y 9. La otra fila tendría 4 cuentas indicando un dígito entre 1 y 4 ó (si está subida la cuenta de la otra fila), entre 6 y 9. Su funcionamiento es muy simple, pero ha prestado (y aún presta) una gran ayuda como soporte físico de los números (Ver Figura 1).

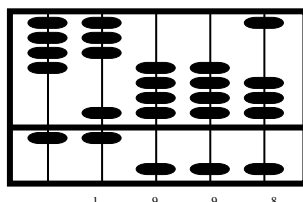


Figura 1: Ábaco simple. Al desplazar cada cuenta añadimos su valor.

**Primera calculadora** Wilhelm **Schickard** diseñó y construyó en 1623 lo que se considera la primera calculadora digital: permitía hacer cálculos automáticos de sumas, restas y, parcialmente automáticos, multiplicaciones y divisiones. Sin embargo una epidemia impidió la difusión de este invento.

Un importante ingenio mecánico, que no computadora, pues no era programable, después del ábaco, fue la máquina de calcular de Blaise **Pascal** (1623–1662), desarrollada 20 años después que

<sup>1</sup>Ya otra cuestión muy distinta es, como veremos en el siguiente tema, que todos los problemas sean susceptibles de describirse mediante algoritmos o procesos.

la de Schickard, y más limitada que la de aquél (tan sólo sumaba y restaba). Se trataba de una serie de engranajes en una caja, que presentaba sus resultados de sumas, restas y multiplicaciones a través de una ventanita. La ventaja sobre el ábaco era que presentaba sus resultados directamente legibles para el ser humano.

**La “máquina analítica”** La primera máquina interesante es la “Máquina Analítica” del inglés Charles **Babbage** (1791–1871), realizada con componentes mecánicos, pero sobre todo programable con un programa cargable y así no dedicada a una única tarea. **Babbage** propuso dos máquinas: “La Máquina de Diferencias” y “La Máquina Analítica”, ambas computadoras movidas por máquinas de vapor. Tan sólo un impresor sueco, Pehr George Scheutz consiguió construir una máquina *diferencial* que presentó en Londres en 1854.

La máquina analítica de **Babbage** se presentó alrededor del año 1930 y contenía todos los elementos que constituyen una computadora moderna diferenciándose de las calculadoras de propósito fijo. La máquina analítica de **Babbage** estaba dividida funcionalmente en dos partes:

1. una que daba órdenes y
2. otra que las ejecutaba

La parte ejecutora era una versión ampliada de la máquina de **Pascal**. La otra era la innovación, que permitía al usuario, cambiando las especificaciones de control, realizar cálculos distintos.

La máquina analítica fue la verdadera antecesora de las computadoras modernas. Tenía una parte de recepción de datos con los que iba a trabajar, tenía una unidad de control que enviaba órdenes a realizar sobre los datos y conseguía los resultados deseados. Esta máquina fue aplicada fundamentalmente para el tedioso cálculo y tabulación de funciones trascendentes, que requerían mucho esfuerzo manual. La entrada de los datos se hacía mediante las tarjetas perforadas ideadas por Hollerith y basadas en las tarjetas dentadas usadas por el francés Joseph M. **Jacquard** a principios de siglo y que tanto ayudaron a la industria de los telares mecánicos durante la “revolución industrial”.

Simultáneamente Leonardo **Torres Quevedo** (1852–1936) propuso una máquina electromecánica basada en el modelo de **Babbage**. En 1928, las tarjetas perforadas se utilizaron para recoger un problema de cálculo astrofísico: tabular las distintas posiciones de la Luna en el cielo.

Sin embargo, la máquina de **Babbage** nunca se puso en funcionamiento debido a la dificultad de los cambios mecánicos requeridos para la lectura de los datos.

Ada **Lovelace**, esposa de C. **Babbage**, hija de Lord **Byron**, matemática, completó los trabajos de su esposo y ha dejado su nombre, el de la primera mujer relevante en informática, para designar uno de los más completos lenguajes de hoy, el lenguaje Ada. Sobre 1940 aparecen dos máquinas electromecánicas:

- “Z3” del alemán Konrad Zuse con aritmética binaria (1941)
- Una máquina debida a Howard Aiken junto con IBM de redes dentadas decimales (1939–1943)

También aparece una calculadora electrónica de válvulas de vacío, la “ENIAC” en Pennsylvania (1943–1946). Estas tres máquinas (las dos electromecánicas y la de válvulas) almacenaban datos y programas en memorias separadas y era una tarea difícil introducir o modificar programas. La programación se realizaba de hecho mediante la modificación de interconexiones cableadas.

La idea de almacenar datos y programas en la misma memoria es de John **von Neumann** (1903–1957), consultor del proyecto ENIAC, cosa que puso en práctica en la máquina ‘EDVAC’. Será el propio von Neumann el que en 1946 arranca el proyecto ‘IAS’, una máquina con lógica binaria paralela y palabra de 40 bits, precursora de actuales computadoras. A partir de ahí comienza un crecimiento espectacular, pues va miniaturizándose y ganándose en velocidad de las válvulas de vacío los transistores, y de ahí los circuitos integrados. Se habla de cinco distintas ‘generaciones’ en la evolución fundamentalmente del ‘hardware’ o soporte físico de los computadores:

1. Hasta 1959. Uso de la electrónica (válvulas de vacío y memorias de ferritas)

2. 1959–1964. Aparición de los transistores. Ver Apéndices.
3. 1965–1970. Integración de múltiples transistores y circuitos en los “circuitos integrados”
4. Microtransistorización
5. Desarrollo de software ‘inteligente’

La última no está concensuada.

**La teoría subyacente** Un avance teórico importante, relativamente reciente fueron los razonamientos de Georges **Boole** (1815–1864). Boole, nacido de clase media baja, fue un auténtico autodidacta, aprendiendo a los dieciséis años cinco idiomas por su cuenta. Llegó a aprender todas las matemáticas de su época. En su trabajo *Una investigación sobre las leyes de la verdad*, publicado en 1854, propone un álgebra para las proposiciones de verdad equivalente a la usada en aritmética; con el uso de + por **or** (‘O’ lógico) y el  $\times$  por el **and** (‘Y’ lógico), etc.

El matemático más influyente del siglo XIX, David **Hilbert** (1862–1943) propuso a los matemáticos el encontrar un sistema axiomático lógico-matemático, del cual pudieran derivarse todas las matemáticas. Sin embargo Kurt **Gödel** (1906–1978) demostró en 1931 que la propuesta de Hilbert era inviable<sup>2</sup>. De otra forma, algunos problemas matemáticos son intrínsecamente irresolubles. Como respuesta Alan **Turing** (1912–1954) publicó en 1936 un trabajo en el que demostró que el “cálculo efectivo” podía considerarse como un tipo particular de máquina abstracta.

## 1.1. Ámbito de aplicación

Hoy día es difícil encontrar algún terreno del trabajo humano en el que no se utilicen los ordenadores, aunque naturalmente existen áreas donde la informática es más adecuada. Esto es así, por ejemplo, en el caso de la creatividad humanista, donde los ordenadores cumplen papeles substitutorios de la antigua ofimática (edición de textos/gráficos, etc). Las comunicaciones, especialmente desde el popularización en 1984 de los interfaces gráficos tipo WIMP (**W**indows, **I**cons, **M**enus, **P**ointers) y, posteriormente, en 1993, de la Web (interfaz estándar de presentación de datos, textuales y de otros tipos, especialmente adecuada para su transmisión por redes), han sugestionado a todo el mundo, prácticamente sin excepción, a utilizar el ordenador, por lo menos como medio de comunicación.

## 1.2. Tipos de computadores

En los procesos de tratamiento de la información en general tenemos que empezar a plantearnos cómo se representa esta información en una máquina<sup>3</sup>. En base a la forma de representar la información tenemos dos tipos de sistemas:

**Analógico** La información está mediante algún mecanismo físico directamente relacionada con la magnitud real externa que se mide. Por ejemplo, la presión, altura, etc. en algún dispositivo de control industrial. Esto permite que dentro de un rango de valores dado la magnitud se comporte como continua. Los resultados de este tipo de sistemas son también ‘continuos’.

**Digitales** Los rangos de valores de la información se cuantizan mediante valores numéricos discretos. Esto es, se miden mediante alguna ‘escala’ dando así lugar a valores discretos (dependientes de la precisión de la escala) en un números enteros. La información real continua se convierte en números más o menos cercanos a ella según la fineza de la escala o precisión (resolución) elegida. Mediante esta técnica se pueden representar todo tipo de informaciones, desde colores, sonidos, etc. imágenes unidimensionales o multidimensionales. Naturalmente

<sup>2</sup>ya que “cualquier sistema lo suficientemente amplio como para abarcar la aritmética de los números naturales deber ser inconsistente o bien contener afirmaciones que jamás podrán probarse o refutarse”

<sup>3</sup>En España se suele utilizar el término *ordenador* o computador (masculino), mientras en sudamérica se usa el de computadora, más cercano al inglés (*computer*), pero en femenino.

en este proceso de medida con una precisión finita se produce una pérdida de información. La digitalización de la información implica una pérdida de información respecto a la información original y puede ser de mayor o menor calidad, siendo usualmente más voluminosa las representaciones de más calidad.

Según esta división, también se puede hablar de computadores híbridos.

**Analógico vs digital** Aunque estamos acostumbrados a los computadores digitales, los computadores analógicos son muy frecuentes especialmente en la industria dónde se controlan directamente dispositivos mecánicos. (Ver Fig. 2.) Sin embargo los computadores analógicos son mucho

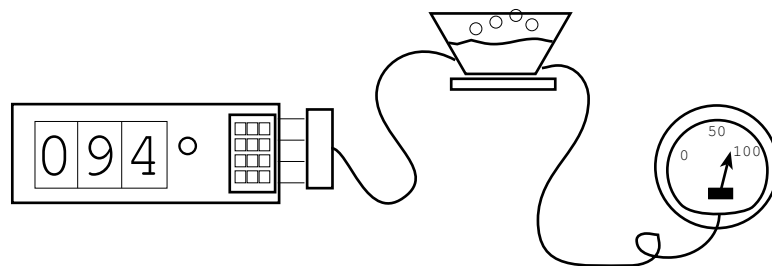


Figura 2: Modelo digital de captación de la información real (se convierte a dígitos) frente al analógico (se escala mediante circuitos eléctricos continuos)

más difíciles de programar que los digitales.

En resumen:

La *máquina analítica* de **Babbage** fue la precursora de lo que se denominan los “computadores digitales”. La palabra digital proviene de la forma de describir los datos. Normalmente concebimos y medimos las magnitudes físicas o reales mediante el concepto de número ‘real’, que consideramos ‘continuo’. Actualmente existen ordenadores capaces de utilizar magnitudes que cambian de manera continua; son los computadores analógico. Usualmente se basan en valores eléctricos que se pueden modificar suave y continuamente dentro de una escala y que alimentan a otros dispositivos que responden, asimismo de manera continua. El principal problema de este tipo de computadores es la dificultad de su ‘programación’.

Los computadores digitales, por otro lado, no son capaces de representar todo el rango de valores ‘continuos’ que se puede dar de cualquier parámetro real, sino que los representan con valores que *saltan* de un punto al siguiente sin posibles valores intermedios. Piénsese, por ejemplo en la palanca de cambios de un automóvil: sólo puede estar en los valores 1, 2, 3, 4, 5 ó -1 (marcha atrás); pero no en  $1\frac{1}{2}$  ó 1,34, por ejemplo. De hecho los datos que “mejor se le dan” a un computador son los números enteros y quizás los racionales ( $x \in \mathbf{Z}$  ó  $x \in \mathbf{Q}$ ), pero para los reales ( $x \in \mathbf{R}$ ) necesita un esfuerzo y un mecanismo de representación mucho mayores. Sin embargo, prácticamente todas las computadoras actuales son digitales y ello se debe a la facilidad del mantenimiento de programas y datos mediante dispositivos electrónicos digitales.

Ahora bien, según el criterio de la **técnica de construcción**, podríamos hablar de computadores de:

**Lógica cableada** en los que de una manera prefijada por el fabricante, el funcionamiento (los procesos que puede seguir) están ‘cableados’, o sea, físicamente fijados en los circuitos. Como ejemplo, podrían ser los ordenadores *analógicos* descritos antes, pero también, aún utilizando técnicas digitales, podríamos incluir dentro de éstos a las *calculadoras* no programables (en ellas, los datos de entrada y de salida son siempre digitales y los procesos están prefijados).

**Lógica programable** Los procesos se pueden escribir en la memoria y el computador los sigue, de manera que pueden realizarse distintos tipos de procesos.

Si nos guiamos del propósito, tendríamos computadores de *uso general* y de *uso específico*.

Si nos guiamos de su **movilidad** podríamos hablar de computadores *encastrados* o *empotrados* (como los que controlan una lavadora o a un avión) y computadores *independientes*.

Si nos guiamos por su **potencia** (o tamaño), de *supercomputadores*, *macrocomputadores* ('mainframes'), *estaciones de trabajos* 'workstations', y computadores *personales profesionales* y computadores *personales domésticos*. Ver figura 3.

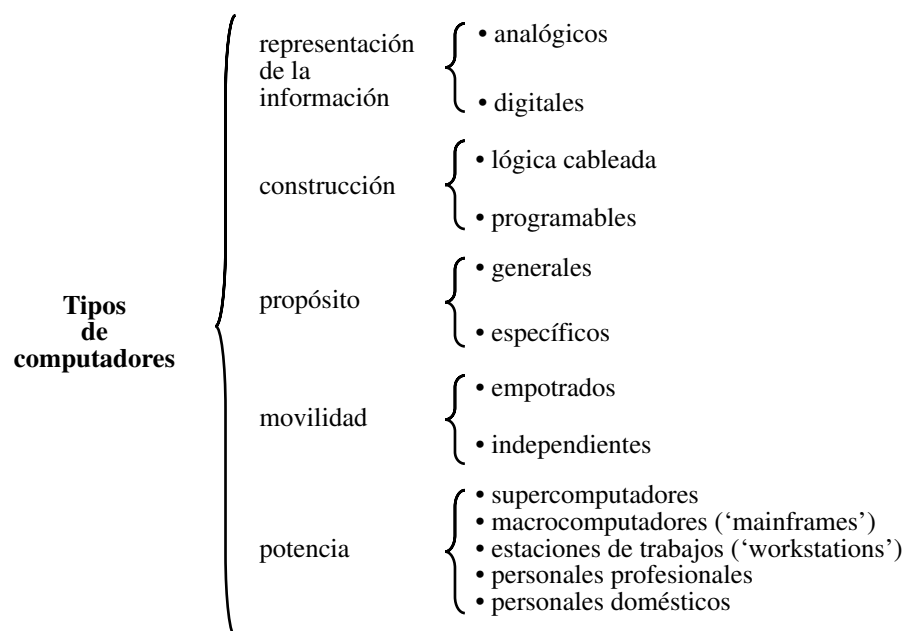


Figura 3: Clasificaciones más comunes de los computadores

### 1.3. La máquina de von Neumann

Aunque es criticado por motivos teóricos quizás el modelo computacional mejor conocido sea el de **von Neumann** que se usa hoy día para describir los lenguajes de programación convencionales y es la base de prácticamente todos los modelos de ordenadores (ver figura 4). Este modelo tiene varias características clave que se muestran de una manera u otra en todos los sistemas actuales:

1. Una **Unidad de memoria** que almacena ('memoriza') *valores* e *instrucciones*. La memoria se divide en celdillas específicas de un tamaño prefijado.
2. Una **Unidad Central de Proceso** (UCP o CPU), que de forma *secuencial* accede a la memoria.
3. Una **instrucción**, que especifica alguna secuencia particular de actividades en la UCP que modifican los contenidos de las localizaciones de la memoria o de los **registros** de la UCP.
4. Un **programa**, que consiste en una secuencia ordenada de instrucciones colocada en la memoria.
5. Un **Contador de Programa** en la UCP para seleccionar la siguiente instrucción de la memoria.
6. Un **lenguaje de programación**, que especifica cómo y qué instrucciones se pueden colocar en la memoria.

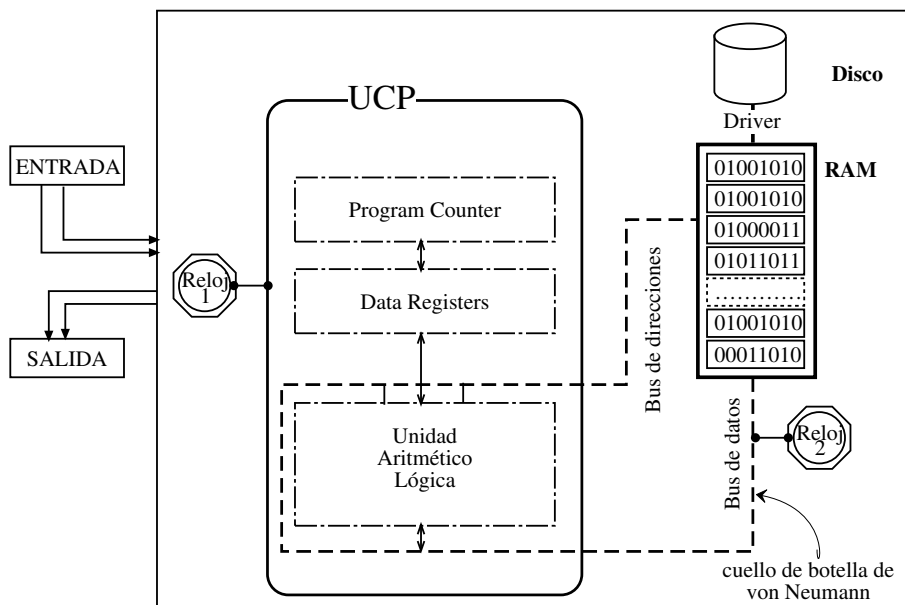


Figura 4: El modelo de von Neumann esquematizado

Una de las características más incómodas de este sencillo modelo es el hecho de acceder en secuencia y una tras otra, las instrucciones **y** los datos a través de un único cableado (bus) con la memoria. Es por eso que se denomina a este el cuello de botella del modelo de von Neumann.

La eficiencia de este modelo se suele medir en instrucciones ejecutadas por segundo. Actualmente se alcanzan rangos de decenas/centenas de millones (MIPs).

**Comentarios sobre este modelo** La principal característica del modelo de von Neumann (MvN) es que durante su ejecución, los programas únicamente pretenden “modificar la memoria”. Debemos conocer las localizaciones de los datos antes de ejecutar los programas para recoger después los resultados. Esto es internamente patente en el funcionamiento y la *semántica* de los lenguajes de programación convencionales. En estos lenguajes de programación la instrucción más importante es la de **asignación**. Todos los lenguajes de programación convencionales disponen de modos de asignación más o menos significativos. En todo caso siempre se debe especificar un destinatario que recibe y una información a escribirle. La dirección de destino queda prefijada en la sintaxis de estos lenguajes mediante una etiqueta identificadora de fácil lectura para el programador y que unívocamente identifica aquella dirección reservada en la memoria por el código que deja el compilador. Así estas localizaciones quedan designadas simbólicamente por nombres dados por el programador y el compilador reserva para cada nombre el lugar adecuado de memoria. Este tipo de símbolos con memoria asociada se denominan **variables**. El motivo es que durante la ejecución del programa va a permitirse la modificación de sus contenidos. La referencia en dos partes de un programa al mismo símbolo, se entiende como la referencia a la misma localización de memoria. El contenido de cada localización de memoria podrá variar a lo largo del programa según le vayamos asignando diferentes cosas. La elección de lugares y formas de almacenamiento se convierte pues en un tema primordial para la eficiencia de los programas, aún cuando no tenga nada que ver con las especificaciones del propio problema original.

La notación formal de la operación de asignación suele ser:

$$\langle \text{registro o localización} \rangle \leftarrow \langle \text{expresión} \rangle$$

mientras que en los lenguajes de programación usualmente se emplea el signo igual (=), aunque, para evitar confusión, Pascal y otros muchos lenguajes utilizan el :=.

$$\text{variable} := \langle \text{variable ó expresión} \rangle$$

Nótese que antes de la ejecución de la ‘asignación’ el contenido de la variable a la izquierda puede ser cualquiera, indistintamente del contenido o resultado de la expresión y que, después de la asignación, al haberse copiado sobre la variable, el contenido de esta es el mismo que el resultado que hubiera habido de la expresión o el contenido de la variable a la derecha. Esta operación es muy particular e inédita, no existiendo un equivalente claro en matemáticas formales.

## 1.4. El funcionamiento del computador digital

Como hemos visto el computador más extendido y fácil de programar es el computador digital. Dentro de los tipos de computador digital se han desarrollado diversas arquitecturas, que se verán en el tema siguiente, pero la más extendida con diferencia partió de lo que se denominó, como hemos visto, el modelo computacional de von Neumann (años 50).

### 1.4.1. Sistemas de numeración

Desde un punto de vista estrictamente matemático resulta en cierto modo un inconveniente el que el hombre de Gro-Magnon y sus descendientes no tuviesen o bien cuatro o bien seis dedos en cada mano. De hecho en restos de hace más de 30.000 años las incisiones marcadas sugieren ya el uso de los números y la base decimal de numeración.

Dado que los computadores utilizan un sistema de representación de los números con distinta base de la acostumbrada base 10, convendrá que repasemos los conceptos relativos a las bases de numeración. Más adelante veremos también cómo se representan en el computador todo tipo de números: enteros positivos, enteros negativos, números reales, etc. y cómo se codifica también otros tipos de información.

### Historia de la notación posicional

El concepto de número (por el momento hablaremos tan sólo de números enteros positivos), que tan familiar nos es hoy a nosotros, fue elaborado muy lentamente. Aún hoy esto puede verse en el modo de contar de distintas razas que hasta tiempos muy recientes han permanecido en un nivel relativamente menos complejo de vida social. En algunas de ellas los números mayores de dos y de tres no tenían ya nombre; en otras llegaban algo más lejos pero terminaban al cabo de pocos números; para los restantes decían simplemente ‘muchos’ o ‘incontables’. Sólo gradualmente se fueron acumulando en los pueblos un conjunto de nombres claramente distintos para los números.

Al principio estos pueblos no tenían la noción de número, aunque podían, a su manera, juzgar sobre el tamaño de una u otra colección de objetos con los que se encontraban a diario. Debemos concluir que los números eran directamente percibidos para ellos como una propiedad inseparable de una colección de objetos, una propiedad que ellos, sin embargo, no se preocupaban en definir. Hoy día estamos tan acostumbrados a contar que difícilmente podemos imaginar este estado de cosas, pero es posible entenderlo.

En cuanto a los símbolos utilizados, aparecieron muchos en los distintos pueblos, en los comienzos de sus culturas. Símbolos numéricos que eran muy diferentes de los actuales, no sólo en su apariencia general, sino también en los principios en que se fundaban.

Por ejemplo, el sistema decimal no se empleaba en absoluto y los antiguos *babylonios* tenían un sistema que era parcialmente decimal y parcialmente sexagesimal. Nuestros actuales símbolos arábigos, y en general nuestro método de formar los números, fueron traídos de la India a Europa por los árabes en el siglo X y arraigaron firmemente en el transcurso de pocos siglos.

La primera particularidad de nuestro sistema es que es un sistema decimal. Pero esto no es de gran importancia, puesto que habría sido posible usar, por ejemplo, un sistema duodecimal introduciendo símbolos especiales para diez y once. La particularidad más importante de nuestro sistema de designar números es que es ‘**posicional**’; esto es, *un mismo dígito tiene distinto significado según sea su posición en la cadena de ellos que forman el número*. Por ejemplo, en 1964, el 1 representa millares, el 9 centenas, el 6 decenas, mientras el cuatro, a las unidades. Este método de escritura no sólo es conciso y sencillo, sino que facilita mucho el cálculo. Los numerales romanos eran a este respecto mucho menos manejables; por ejemplo,

el número 1964 se escribe MCMLXIV; además es muy laborioso multiplicar dos números altos escritos en caracteres romanos.

La escritura posicional requiere que de un modo u otro se especifique que una cierta categoría o peso de números no existe, puesto que de no hacerlo así confundiríamos, por ejemplo, treinta y uno con trescientos uno. En forma rudimentaria, el cero ya aparece en las últimas escrituras cuneiformes babilónicas, pero su introducción sistemática fue obra de los indios (ya en el siglo IX aparece en un manuscrito el número '270' tal cual; pero es probable que el cero fuese introducido en la India bastante antes, en el siglo VI); ello les permitió elaborar un sistema de escritura completamente posicional como el que tenemos hoy día. Conceptualmente el cero no es nada y en el sánscrito de la antigua India se le llamaba 'vacío'; no obstante en conexión con otros números el cero adquiere sentido y propiedades.

Matemáticamente la representación posicional de los números tiene una equivalencia directa con su *expansión polinómica*, así un número  $N$  que en representación posicional se escriba:

$$d_n d_{n-1} d_{n-2} \dots d_1 d_0$$

siendo  $d_i$  los dígitos (como en 1964 lo son  $d_3 = 1$ ,  $d_2 = 9$ ,  $d_1 = 6$  y  $d_0 = 4$ ), sería:

$$N = \sum_{i=0}^n d_i \cdot b^i$$

siendo  $b$  la base de la representación; en nuestro caso  $b = 10$ .

Si el número tiene *parte decimal*, esta se separa a la derecha mediante una coma<sup>(4)</sup> y su expansión polinómica sería:

$$N = \sum_{i=0}^n d_i \cdot b^i + \sum_{i=1} d_{-i} \cdot b^{-i}$$

Los computadores utilizan el sistema de numeración binario (base 2, dígitos con los que se suelen representar, el 0 y el 1). A un dígito binario se le llama en informática *bit*. A un grupos de 4 bits, *nibble* y a un grupo de 8 bits se le llama *byte*.

Veamos la representación de los primeros números en cuatro bases interesantes: la decimal, la binaria, octal y hexagesimal:

10	2	8	16
0	0	0	0
1	1	1	1
2	01	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

<sup>4</sup>en los países anglosajones, como otras cosas, se hace justo al revés, esto es, se usa el punto para separar la parte entera de la decimal y la coma para separar los grupos de tres dígitos. Por ejemplo: el número  $1_1345,823,32$  sería en inglés:  $1,345,823,32$

Vemos que lo mismo que nos sobran símbolos con los nueve dígitos: (0, 1, 2, ..., 9) para los números en base 2 ó base 8, nos faltan para la base 16 con la que se emplean las letras de la A a la F.

Así, en la memoria del ordenador, en principio, los números **enteros positivos** tienen una representación directa. Bastará para ello pasar el número que sea (en base 10) a la base 2. Por ejemplo:

$$\begin{aligned}8_{(10)} &\equiv 100_{(2)} \\10_{(10)} &\equiv 1010_{(2)} \\100_{(10)} &\equiv 0110\ 0100_{(2)} \\2002_{(10)} &\equiv 0111\ 1101\ 0010\end{aligned}$$

Evidentemente se necesitan más dígitos para representar los números en base 2 que en base 10. Aproximadamente  $3,322 (= \log_2 10)$  dígitos más.

La **conversión** de la representación binaria a la decimal y viceversa, por ser ambas *posicionales*, son fáciles, en general. En cualquier representación en base  $b$ , el valor de un número  $x$  (que tiene  $n + 1$  ‘dígitos’ en su representación en esa base  $b$ ) es, como hemos visto:

$$x \equiv "d_n d_{n-1} \dots d_1 d_0" \equiv d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \dots + d_1 \cdot b^1 + d_0 \cdot b^0 = \sum_{i=0}^n d_i \cdot b^i$$

La primera “ $d_n d_{n-1} \dots d_1 d_0$ ” es la representación posicional. La segunda  $d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \dots + d_1 \cdot b^1 + d_0 \cdot b^0$ , la polinómica.

En la evaluación es más cómodo ir de los ‘pesos’ bajos a los altos, por ejemplo, para averiguar el valor decimal de  $101_{(2)}$ :

$$101_{(2)} \equiv 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 \equiv 5_{(10)}$$

Es pues directo el paso de la cadena de dígitos en una base a la decimal puesto que sabemos evaluar muy bien cosas como  $2^4$  en decimal.

La representación posicional o polinómica de cualquier número es única si los coeficientes son menores que la base.

Veamos los métodos de conversión:

**Decimal a binario** Viendo la representación podemos pensar en un método: quitar las potencias de 2 del número hasta que ya no exista residuo. Primero se buscaría la potencia más alta de 2 que hay en el número, después de restarla se hace lo mismo, así hasta que llegamos 1 ó a 0, etc. Por ejemplo, a 25 se le puede restar la mayor potencia de dos que es  $2^4 = 16$ , lo que nos da  $25 - 2^4 = 9$ . La mayor potencia de 2 que se le puede restar a 9 es  $2^3: 9 - 2^3 = 1$ ; por lo tanto, la representación binaria de 25 es 11001.

El método anterior es tedioso. Lo que se hace usualmente es dividir por 2 sucesivamente y los restos, **en orden inverso** constituyen los dígitos binarios del número:

$$\begin{array}{r|l} & 125 \\ 1 & 62 \\ 0 & 31 \\ 1 & 15 \\ \uparrow 1 & 7 \\ & 3 \\ & 1 \\ & 1 \\ & 0 \end{array} \quad \Rightarrow 125_{(10)} \equiv 1111101_{(2)}$$

**Números no enteros** Si el número contiene parte decimal hará falta descomponerlo, esto es  $102,247 = 102 + 0,247$ : se encuentran las representaciones por separado y se concatenan.

Recordemos primero que en notación posicional un número con decimales puede representarse en forma posicional o polinómica como:

$$\begin{aligned} x &\equiv "d_n d_{n-1} \dots d_1 d_0, d_{-1} d_{-2} \dots" \\ &\equiv d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \dots + d_1 \cdot b^1 + d_0 \cdot b^0 + d_{-1} \cdot b^{-1} + d_{-2} \cdot b^{-2} \dots \\ &= \sum_{i=0}^n d_i \cdot b^i + \sum_{i=1} d_{-i} \cdot b^{-i} \end{aligned}$$

Por ejemplo,  $1101,101_{(2)} = 13 + 0,625 = 13 + 1/2 + 1/4^5$

La conversión de fracciones decimales a binarias se puede llevar a cabo utilizando varias técnicas. Nuevamente el método más directo es el de quitar las potencias de  $1/2$  que se se pueda, sucesivamente, pero ahora primero las potencias de exponente más bajo se prueban antes. Por ejemplo:

$$\begin{aligned} 0,875 - \frac{1}{2} &= 0,375 \\ 0,375 - \left(\frac{1}{2}\right)^2 &= ,375 - ,25 = ,125 \\ 0,125 - \left(\frac{1}{2}\right)^3 &= ,125 - ,125 = 0 \end{aligned}$$

con lo que  $0,875_{(10)} = 0,111_{(2)}$ .

El método más rápido es duplicar la fracción decimal, entonces si aparece un 1 a la izquierda del punto decimal, se coloca un 1 a la derecha del resultado binario que se está buscando:

$$\begin{aligned} ,875 \times 2 &= 1,750 \Rightarrow 0,1 \\ ,750 \times 2 &= 1,500 \Rightarrow 0,11 \\ ,5 \times 2 &= 1,000 \Rightarrow 0,111 \end{aligned}$$

o

$$\begin{aligned} ,4375 \times 2 &= 0,8750 \Rightarrow 0,0 \\ ,8750 \times 2 &= 1,7500 \Rightarrow 0,01 \\ ,750 \times 2 &= 1,5000 \Rightarrow 0,011 \\ ,500 \times 2 &= 1,0000 \Rightarrow 0,0111 \end{aligned}$$

**Tamaño de la representación** Cuando escribimos un número sobre el papel estamos suponiendo que tendremos espacio suficiente en el ancho del papel para poder escribir y, si el número no cabe, lo escribimos más apretado o pasamos a los sucesivos renglones y páginas *que nos vayan haciendo falta*.

Normalmente en la computadora, la ‘memoria’ está asignada de una manera limitada y rigurosa para los datos de forma que cada dato tiene un espacio predeterminado donde se representa. Usualmente, con los números enteros se utilizan tamaños de un *byte*, un doble byte y cuatro bytes. No es difícil calcular “lo que nos cabe” en estos espacios ‘típicos’:

bytes	rango	total
1 byte	0 ... 255	$2^8$
1 doble byte	0 ... 65,535	$2^{16}$
4 bytes	0 ... 4,294,967,295	$2^{32}$

<sup>5</sup>Atención: en muchas ocasiones, se utilizará el punto ‘.’ como separador entre la parte entera y la decimal de los números. Esto proviene de la forma americana, que es justo la contraria a la europea, esto es, en estados unidos 2,001,994’3 (dos millones mil novecientos noventa y cuatro coma 3) se escribe 2,001,994.3

Veremos más adelante técnicas para almacenar aritméticamente números ‘reales’ con cualquier posición de la coma y factor de escala. Por otro lado, otras técnicas no estándares de los lenguajes de programación usuales permitirán manipular números con “precisión infinita”. De cualquier manera, siempre se pueden almacenar *cadena de caracteres* en las que podemos representar codificados (mediante los códigos ASCII o, como veremos, en BCD) los dígitos de números ‘grandes’.

### 1.4.2. Medida de volúmenes de información binaria

Como en el ordenador, para dirigirse a cualquier parte ya sea de su memoria RAM como de dispositivos de almacenamiento externos, se utilizan valores numéricos basados en representaciones binarias, las cantidades que podemos ‘direccionar’ serán siempre potencias de 2. Por ejemplo, si tenemos un nibble para direccionar, podríamos direccionar hasta 16 posiciones de memoria, por ejemplo. Si en vez de cuatro bits, tenemos 64, podríamos direccionar hasta  $2^{32} = 4_1294,967,296$  sitios distintos.

Usualmente la mínima cantidad de memoria que se lee o se escribe es el byte y por ello no tiene sentido direccionar menos de un byte, con lo que una máquina con un bus de direcciones de 32 bits puede direccionar hasta  $4_1294,967,296$  bytes.

Esta forma de medir las cantidades, basadas en el sistema decimal cuando lo que se tiene en realidad en el computador es el sistema binario, resulta incómoda por lo que se emplean otras unidades de volumen de información que son:

1 byte	8 bits			1 byte
1 Kb	$2^{10}$ bytes	1024 bytes		≈ mil bytes
1 Mb	$2^{20}$ bytes	1024 Kb	$1_1048,576$	≈ un millón de bytes
1 Gb	$2^{30}$ bytes	1024 Mb	$1,073_1741,824$	≈ mil millones de bytes

Hoy día, por ejemplo, una memoria RAM de un ordenador personal puede tener unos 640 Mb, mientras su disco duro tiene 40 Gb y la memoria caché junto a su procesador tiene 1 Mb.

En comunicaciones, se suele medir la velocidad de transferencia de la información en bits por segundo:

$$1\text{bit/seg} \equiv 1\text{baudio}$$

Un módem<sup>6</sup> de 33 kbaudios, teniendo en cuenta que cada 8 bits se suele enviar uno de *stop*, y los errores de transmisión (aunque no necesariamente haya un *bit de paridad*), podríamos decir que transmite 3 Kb/seg. Una conexión ADSL básica permite una velocidad de salida de 12 Kb/s y de 26 Kb/s de entrada.

Teniendo en cuenta que las direcciones ‘físicas’ de cada punto en *Internet* se basan en cuatro bytes, que se escriben separados por puntos, por ejemplo, 150.214.300.1, podemos asegurar que el número de puntos conectables actualmente en Internet es de  $4,294_1967,296 = 2^{32}$ . Otra cuestión es que para no tener que acordarnos de tantos números se asocian con ellos los nombres lógicos correspondientes a cada uno de aquellos números ‘IP’. Como, [www.malaga.es](http://www.malaga.es).

### 1.4.3. Sistemas de codificación de la información

Aparte de la codificación de los números que hemos visto, naturalmente, como todos sabemos, los ordenadores son capaces de guardar/codificar otras cosas. Una de las más importantes son el texto, las letras. Además de las letras, el ordenador podrá codificar cualquier cosa para la que establezcamos una técnica más o menos precisa de codificación. Nótese que codificar no es más que *traducir de una representación a otra*.

En el caso de los ordenadores la información se traduce a números. Así, para codificar un sonido, lo que se hace es medir la amplitud de la presión sonora un número suficiente de veces en un periodo de tiempo y guardar esos números. Típicamente, un sonido de mediana calidad de

<sup>6</sup>Aparato que sirve para modular y demodular los bits en forma de sonidos transmisibles por teléfono

representación puede tener una frecuencia de muestreo de unos 11 K, esto es, se midieron 11.000 valores cada segundo, quizás por cada canal de estéreo.

Las imágenes planas se codifican teselando la superficie y midiendo en cada cuadradito el valor de cada canal de color (rojo, verde, azul) dentro asimismo de un rango más o menos amplio de valores. Tanto el número de losetas como el de colores influirán directamente en la calidad de la imagen codificada. Existen infinidad de formatos de almacenamiento de los gráficos, especialmente debido al problema del espacio que éstos ocupan que se trata de minimizar mediante muy diversas técnicas<sup>7</sup>.

**La tabla ASCII** Sin embargo, el tipo de información más importante a codificar, aparte de los números, son las letras, el texto. Dado que existen muchos alfabetos diferentes en el mundo, este no es un problema trivial. Desde los años 60 ya se estableció el sistema actualmente estándar en *todos* los sistemas para codificar cada letra, al menos del alfabeto anglosajón. Antes, el sistema diseñado por IBM fue el sistema de codificación EBCDIC que actualmente está en absoluto desuso. Más tarde se implantó el sistema ASCII, necesariamente incompleto, dada la cantidad de posibles, no sólo ya alfabetos, como hemos dicho, sino incluso posibles símbolos distintos que pudieran necesitarse codificar dentro de una lengua.

El sistema ASCII consiste en un conjunto de 128 códigos para referirse a otros tantos signos.

Estos signos incluyen todas las letras del alfabeto anglosajón (abcdefghijklmnopqrstuvwxyz) en mayúsculas y en minúsculas. Las mayúsculas están codificadas antes que las minúsculas de manera que se tienen dos grupos de 26 códigos para las letras mayúsculas, después hay 6 signos distintos (concretamente: [ \ ] ^ \_ ' ) y después están las codificaciones de las 26 letras en minúsculas. Por lo tanto la ‘distancia’ de la letra ‘A’ a la ‘a’ (o de la ‘H’ a la ‘h’) es de 32. Aparte de estos 70 códigos hay otros pocos para otros signos de puntuación (por ejemplo, *arroba* ‘@’ tiene el código ASCII  $64_{(10)}$  ó  $40_{(16)}$ ). Todo lo que pueda ser parte de un texto debe ser codificado, el espacio es el número  $32_{(10)}$  =  $20_{(16)}$  o los dígitos del 0 al 9, que están codificados con los valores  $48_{(10)}$  =  $30_{16}$  a  $57_{(10)}$  =  $39_{(16)}$ .

Aparte de codificarse en la tabla ASCII los caracteres visibles, también se codifican una serie de *caracteres de control*<sup>8</sup>. Esta codificación está ya fuera de lugar y en desuso pues proviene de la época en la que la transferencia de información se hacía mediante conexiones lentas con protocolos ya abandonados. Sin embargo existen 32 “caracteres de control” codificados en las 32 primeras posiciones de la tabla ASCII.

No todos los caracteres de control son inútiles hoy. Los textos usualmente están compuestos de líneas, palabras, espacios, y signos de puntuación que están codificadas con los nombres de “retorno de carro”  $0D_{(16)}$  =  $13_{(10)}$  (en recuerdo a lo que se hacía con las ya antiguas máquinas de escribir), los saltos de tabulador 7, el “cambio de línea”  $A_{(16)}$  =  $10_{(10)}$ , etc..<sup>9</sup>

Los caracteres de control de la tabla ASCII son los 32 primeros caracteres (Ver Figura )•:

	DEC	OCT	HEX	BIN	teclado	descripción
NUL	000	000	0x00	00000000	[Control-@]	Null char
SOH	001	001	0x01	00000001	[Control-A]	Start of Header
STX	002	002	0x02	00000010	[Control-B]	Start of Text
ETX	003	003	0x03	00000011	[Control-C]	End of Text
EOT	004	004	0x04	00000100	[Control-D]	End of Transmission
ENQ	005	005	0x05	00000101	[Control-E]	Enquiry
BEL	007	007	0x07	00000111	[Control-G]	Bell, \a

<sup>7</sup>La más importante es la compresión, que hoy día, especialmente en las imágenes de tipo fotografía (no diagramas), son incluso codificaciones que permiten aproximaciones, consiguiendo reducciones drásticas de tamaño. Por ejemplo, una imagen que en ‘crudo’ puede ocupar 20Mb (que podría ser una foto con 300 puntos de color por pulgada), guardada mediante un formato tipo ‘JPEG’ de media calidad, puede ocupar tan sólo 200Kb.

<sup>8</sup>Estos “caracteres de control” están representados en la Figura 6 mediante sus nombres cortos ‘ACK’, ‘CR’, etc. no tienen representación gráfica ninguna, sino que representaban *acciones a tomar*.

<sup>9</sup>Por desgracia no ha habido, sin embargo, mucho acuerdo en los distintos constructores de sistemas en cómo indicar el final de una línea o renglón de texto. Sistemas operativos como UNIX utilizan el carácter 10 (line feed, LF), mientras que VMS de Digital o MacOS de Apple utilizan el 13 (carriage return, CR). Para empeorar la cosa MSDOS utiliza dos caracteres seguidos siempre e inseparables, para indicar un final de línea, la pareja CR-LF.

BS	008	010	0x08	00001000	[Control-H]	Backspace
HT	009	011	0x09	00001001	[Control-I]	Horizontal Tab, \t
LF	010	012	0x0A	00001010	[Control-J]	Line Feed, \n
VT	011	013	0x0B	00001011	[Control-K]	Vertical Tab
FF	012	014	0x0C	00001100	[Control-L]	Form Feed, \f
CR	013	015	0x0D	00001101	[Control-M]	Carriage Return, \r
SO	014	016	0x0E	00001110	[Control-N]	Serial In
SI	015	017	0x0F	00001111	[Control-O]	Serial Out
DLE	016	020	0x10	00010000	[Control-P]	Data Line Escape
DC1	017	021	0x11	00010001	[Control-Q]	Device Control 1 - XON
DC2	018	022	0x12	00010010	[Control-R]	Device Control 2
DC3	019	023	0x13	00010011	[Control-S]	Device Control 3 - XOFF
DC4	020	024	0x14	00010100	[Control-T]	Device Control 4
NAK	021	025	0x15	00010101	[Control-U]	Negative Acknowledgement
SYN	022	026	0x16	00010110	[Control-V]	Synchronous Idle
ETB	023	027	0x17	00010111	[Control-W]	End of Transmit Block
CAN	024	030	0x18	00011000	[Control-X]	Cancel
EM	025	031	0x19	00011001	[Control-Y]	End Of Medium
SUB	026	032	0x1A	00011010	[Control-Z]	Substitutue
ESC	027	033	0x1B	00011011	[Control-[]	Escape
FS	028	034	0x1C	00011100	[Control-\]	File Separator
GS	029	035	0x1D	00011101	[Control-]]	Group Separator
RS	030	036	0x1E	00011110	[Control-^]	Request to Send
US	031	037	0x1F	00011111	[Control-_]	Unit Separator
SP	032	040	0x20	00100000	(space)	
!	033	041	0x21	00100001		

El resto de la parte estándar de la tabla ASCII es:

32	0x20	56	0x38	8	80	0x50	P	104	0x68	h	
33	0x21	!	57	0x39	9	81	0x51	Q	105	0x69	i
34	0x22	"	58	0x3A	:	82	0x52	R	106	0x6A	j
35	0x23	#	59	0x3B	;	83	0x53	S	107	0x6B	k
36	0x24	\$	60	0x3C	<	84	0x54	T	108	0x6C	l
37	0x25	%	61	0x3D	=	85	0x55	U	109	0x6D	m
38	0x26	&	62	0x3E	>	86	0x56	V	110	0x6E	n
39	0x27	'	63	0x3F	?	87	0x57	W	111	0x6F	o
40	0x28	(	64	0x40	@	88	0x58	X	112	0x70	p
41	0x29	)	65	0x41	A	89	0x59	Y	113	0x71	q
42	0x2A	*	66	0x42	B	90	0x5A	Z	114	0x72	r
43	0x2B	+	67	0x43	C	91	0x5B	[	115	0x73	s
44	0x2C	,	68	0x44	D	92	0x5C	\	116	0x74	t
45	0x2D	-	69	0x45	E	93	0x5D	]	117	0x75	u
46	0x2E	.	70	0x46	F	94	0x5E	^	118	0x76	v
47	0x2F	/	71	0x47	G	95	0x5F	_	119	0x77	w
48	0x30	0	72	0x48	H	96	0x60	'	120	0x78	x
49	0x31	1	73	0x49	I	97	0x61	a	121	0x79	y
50	0x32	2	74	0x4A	J	98	0x62	b	122	0x7A	z
51	0x33	3	75	0x4B	K	99	0x63	c	123	0x7B	{
52	0x34	4	76	0x4C	L	100	0x64	d	124	0x7C	
53	0x35	5	77	0x4D	M	101	0x65	e	125	0x7D	}
54	0x36	6	78	0x4E	N	102	0x66	f	126	0x7E	~
55	0x37	7	79	0x4F	O	103	0x67	g	127	0x7F	

La transmisión de texto ASCII entre distintas plataformas (distintos sistemas operativos o computadores) es considerada la forma más estándar y menos equívoca de entenderse entre ellos. Es así por ejemplo como se transmiten las páginas de texto y las órdenes en la Web (no así cualquier otro elemento multimedia, como sonido o imagen). Todas las plataformas entienden de la misma forma los códigos ASCII y ven representados, por lo tanto el mismo texto (salvo los finales de línea antes apuntados, que son fácilmente reconocibles en cualquier caso). Sin embargo no todo el mundo habla en inglés.

Al representar tan sólo 128 signos necesitamos 7 bits. El octavo bit de todo byte antiguamente era muy importante en las antiguas formas de comunicaciones y se utilizaba para representar la ‘paridad’ (par-impar) de aquel byte.

Hoy día el octavo bit no se utiliza para controlar la correcta transmisión de los datos, no se utiliza el bit de paridad normalmente o al menos en los procesos en el interior del computador. Esto llevó a que cada fabricante ampliase los posibles signos representables mediante a un byte completo. Los nuevos 128 signos *no son ASCII* al depender de cada fabricante y se les llama ASCII’s altos. El problema es que no sólo difieren los signos que los distintos fabricantes han elegido para la parte alta de la tabla ASCII sino que incluso algunos, como Microsoft (que probablemente sea un fabricante conocido por el estudiante), cambia la representación de los ASCII’s altos de una versión a otra de su software, concretamente como veremos, de MSDOS a Windows. Esto significa que un texto con signos ASCII altos no significará lo mismo en una plataforma que en otra. Las conversiones al pasar un texto ASCII de un sistema a otro no deberían ser difíciles excepto que el sistema destino no tenga representación para signos concretos del de origen. Ver Fig. 6.

Concretamente en castellano tenemos que tener las letras ‘áéíóüíÁÉÍÓÚÛÑ’ y los signos ‘¿’ en distintas posiciones de la parte alta de la tabla ASCII.

El hecho de que los caracteres no anglosajones no estén junto a los anglosajones, por ejemplo, la ‘ñ’ está bastante lejos de la ‘n’, complica la ordenación de las palabras codificadas en ASCII.

La ordenación de palabras (que no son más que cadenas de caracteres codificados mediante la tabla ASCII en el ordenador) puede dar lugar a sorpresas si se hacen directamente mediante las posiciones ASCII que es lo inmediato, así: Avión < abajo y aseo < año.

¿Cómo hubieses diseñado la tabla ASCII para evitar estos problemas?

¿Qué solución le habrías dado al problema de los alfabetos internacionales? ¿Y a alfabetos como el chino, japonés, cirílico, hebreo, etc.?

¿Qué ocurre si se quiere ordenar una secuencia de nombres como:

```
2.Segundo
290.Renglón
...
10.Décimo renglón
1.Primerero
```

¿en qué orden quedarían? ¿cómo se deberían haber nombrado? O sea que *literalmente* 123 < 41. Si se quieren comparar cadenas que tienen dígitos para que queden bien debemos rellenar siempre a la izquierda con tantos 0’s como haga falta para que todos los números a comparar tengan la misma longitud, por ejemplo: 041 < 123.

Actualmente el estándar UTF-8 (<http://www.utf-8.com/>), dentro de los sistemas de codificación de UNICODE (<http://www.unicode.org/>) recoge la codificación de la mayoría de los signos de escritura de los lenguajes occidentales y también de los orientales, e incluso de símbolos matemáticos, etc. Para ello aunque UTF-8 utiliza entre uno y seis bytes para describir cada signo. Existen otros sistemas de codificación dentro de UNICODE, unos ya establecidos y otros aún en desarrollo.

#### 1.4.4. Programas e instrucciones

Los computadores que nos interesan son los digitales programables, y entre ellos la arquitectura más utilizada, sencilla y conocida es la de von Neumann. En el modelo de von Neumann se

Dec	Octal	Hex	Binary	Value
000	000	000	00000000	NUL (Null char.)
001	001	001	00000001	SOH (Start of Header)
002	002	002	00000010	STX (Start of Text)
003	003	003	00000011	ETX (End of Text)
004	004	004	00000100	EOT (End of Transmission)
005	005	005	00000101	ENQ (Enquiry)
006	006	006	00000110	ACK (Acknowledgment)
007	007	007	00000111	BEL (Bell)
008	010	008	00001000	BS (Backspace)
009	011	009	00001001	HT (Horizontal Tab)
010	012	00A	00001010	LF (Line Feed)
011	013	00B	00001011	VT (Vertical Tab)
012	014	00C	00001100	FF (Form Feed)
013	015	00D	00001101	CR (Carriage Return)
014	016	00E	00001110	SO (Shift Out)
015	017	00F	00001111	SI (Shift In)
016	020	010	00010000	DLE (Data Link Escape)
017	021	011	00010001	DC1 (XON) (Device Control 1)
018	022	012	00010010	DC2 (Device Control 2)
019	023	013	00010011	DC3 (XOFF)(Device Control 3)
020	024	014	00010100	DC4 (Device Control 4)
021	025	015	00010101	NAK (Negative Acknowledgement)
022	026	016	00010110	SYN (Synchronous Idle)
023	027	017	00010111	ETB (End of Trans. Block)
024	030	018	00011000	CAN (Cancel)
025	031	019	00011001	EM (End of Medium)
026	032	01A	00011010	SUB (Substitute)
027	033	01B	00011011	ESC (Escape)
028	034	01C	00011100	FS (File Separator)
029	035	01D	00011101	GS (Group Separator)
030	036	01E	00011110	RS (Request to Send)(Record Separator)
031	037	01F	00011111	US (Unit Separator)

## Tabla ASCII

de control  
(no visibles)

### Ejemplo

"	H	o	l	a	"	\0
72	111	108	97		0	

+32

Dec	Octal	Hex	Binary	Value	Dec	Octal	Hex	Binary	Value	Dec	Octal	Hex	Binary	Value
032	040	020	00100000	SP (Space)	065	101	041	01000001	A	097	141	061	01100001	a
033	041	021	00100001	!	066	102	042	01000010	B	098	142	062	01100010	b
034	042	022	00100010	"	067	103	043	01000011	C	099	143	063	01100011	c
035	043	023	00100011	#	068	104	044	01000100	D	100	144	064	01100100	d
036	044	024	00100100	\$	069	105	045	01000101	E	101	145	065	01100101	e
037	045	025	00100101	%	070	106	046	01000110	F	102	146	066	01100110	f
038	046	026	00100110	&	071	107	047	01000111	G	103	147	067	01100111	g
039	047	027	00100111	'	072	110	048	01001000	H	104	150	068	01101000	h
040	050	028	00101000	(	073	111	049	01001001	I	105	151	069	01101001	i
041	051	029	00101001	)	074	112	04A	01001010	J	106	152	06A	01101010	j
042	052	02A	00101010	*	075	113	04B	01001011	K	107	153	06B	01101011	k
043	053	02B	00101011	+	076	114	04C	01001100	L	108	154	06C	01101100	l
044	054	02C	00101100	,	077	115	04D	01001101	M	109	155	06D	01101101	m
045	055	02D	00101101	-	078	116	04E	01001110	N	110	156	06E	01101110	n
046	056	02E	00101110	.	079	117	04F	01001111	O	111	157	06F	01101111	o
047	057	02F	00101111	/	080	120	050	01010000	P	112	160	070	01110000	p
048	060	030	00110000	0	081	121	051	01010001	Q	113	161	071	01110001	q
049	061	031	00110001	1	082	122	052	01010010	R	114	162	072	01110010	r
050	062	032	00110010	2	083	123	053	01010011	S	115	163	073	01110011	s
051	063	033	00110011	3	084	124	054	01010100	T	116	164	074	01110100	t
052	064	034	00110100	4	085	125	055	01010101	U	117	165	075	01110101	u
053	065	035	00110101	5	086	126	056	01010110	V	118	166	076	01110110	v
054	066	036	00110110	6	087	127	057	01010111	W	119	167	077	01110111	w
055	067	037	00110111	7	088	130	058	01011000	X	120	170	078	01111000	x
056	070	038	00111000	8	089	131	059	01011001	Y	121	171	079	01111001	y
057	071	039	00111001	9	090	132	05A	01011010	Z	122	172	07A	01111010	z
058	072	03A	00111010	:	091	133	05B	01011011	[	123	173	07B	01111011	{
059	073	03B	00111011	;	092	134	05C	01011100	\	124	174	07C	01111100	
060	074	03C	00111100	<	093	135	05D	01011101	]	125	175	07D	01111101	}
061	075	03D	00111101	=	094	136	05E	01011110	^	126	176	07E	01111110	~
062	076	03E	00111110	>	095	137	05F	01011111	_	127	177	07F	01111111	
063	077	03F	00111111	?	096	140	060	01100000	`					
064	100	040	01000000	@										

Figura 5: Tabla ASCII universal y estándar válida en todos los sistemas.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SPC	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
80	À	Á	Â	Ç	É	Ê	Ë	Ï	Ñ	Ó	Ô	Õ	Ö	Ù	Ú	Û
90	ä	å	æ	ç	è	é	ê	ë	ï	í	î	ï	ñ	ó	ô	õ
A0	†	°	€	£	§	•	¶	β	⊗	⊙	⊚	⊛	′	″	≠	Æ
B0	∞	±	≤	≥	¥	μ	∂	Σ	Π	π	∫	≡	≈	Ω	æ	ø
C0	¿	¡	¬	√	ƒ	≈	Δ	«	»	…	NBS	À	Á	Â	Ë	Ï
D0	-	-	"	"	'	'	÷	◊	ÿ	ÿ	/	⊗	<	>	fi	fl
E0	‡	•	,	„	%	Â	Ê	Á	Ë	È	Í	Î	Ï	Ì	Ó	Ô
F0	•	ò	ú	û	ù	ı	ˆ	˜	-	˘	˙	˚	˛	˜	˘	˙

Figura 6: Tabla del sistema operativo Apple MacOS. Codificación “MacOSRoman”

almacenan en la RAM tanto los datos codificados como el propio programa actual con el que funcionará el ordenador.

Un programa es un conjunto de *instrucciones* para la máquina escritas en un *lenguaje*. En general estas instrucciones son de varios tipos que podríamos clasificar en aquéllas de transferencia de datos, de tratamiento de datos, de bifurcación y saltos (que modifican el orden de ejecución de la secuencia del programa) y otras.

Las instrucciones que finalmente ejecuta el procesador central del computador deben naturalmente estar escritas en un lenguaje que él entienda: el lenguaje máquina (también llamado ensamblador aunque no sea éste exactamente lo mismo). El lenguaje máquina está compuesto de un conjunto usualmente no muy grande de posibles acciones muy simples propias del modelo de procesador concreto que tenga la máquina (Motorola PPC 604, Pentium II, Sparc...)<sup>10</sup>. Naturalmente el lenguaje máquina está diseñado para ser leído directamente por el procesador de manera eficiente y está lógicamente codificado en secuencias de bits poco inteligibles para el ser humano.

El lenguaje máquina es el lenguaje de más bajo nivel. Un lenguaje inmediatamente por encima de él en legibilidad es el lenguaje Ensamblador, que no es más que el lenguaje máquina escrito instrucción a instrucción con nombres más legibles, pero con la misma estructura. El lenguaje ensamblador no es pues más que el lenguaje máquina escrito mediante palabras más legibles por una persona. Éste se traduce palabra a palabra a instrucciones que controlan directamente el hardware y por lo tanto se requiere conocer muy bien cada chip de la máquina concreta para programar en ensamblador.

El lenguaje ensamblador se compila (aunque particularmente en este caso se dice “se ensambla”) directamente en código máquina.

Si en vez de dirigirnos en nuestro lenguaje directamente a las operaciones que puede ejecutar el procesador sobre el que trabajamos nos podemos expresar con instrucciones más amplias y

<sup>10</sup>En este punto es interesante comentar cómo, finalmente tras periodos de pruebas desde principios de los 80, se ha visto que el tipo de procesador más eficiente especialmente por admitir frecuencias de reloj más altas, es el de tipo RISC (conjunto de instrucciones reducidas y simples) frente al CISC (complejo conjunto de instrucciones más ‘potentes’). Los chips de tipo RISC no sólo permiten velocidades mucho mayores debido a que consumen menos potencia sino que dejan la optimización del código al programa.

generales decimos que estamos trabajando a más alto nivel. Un lenguaje de más alto nivel que el ensamblador es el lenguaje C o el Modula2. En estos lenguajes cada acción que yo solicito al escribir se convierte en conjuntos de acciones del procesador o del hardware, en general, que realizan la acción de más alto nivel. En esta expansión el lenguaje de alto nivel se termina convirtiendo en lenguaje máquina, ya ilegible por nosotros. Al programa que nosotros escribimos se le denomina *código fuente*, al final que ejecuta en código máquina el ordenador, *objeto* o *ejecutable*. Al proceso de convertir las instrucciones escritas en lenguaje de alto nivel en lenguaje máquina se le denomina *traducción* o *compilación*. Se dice compilación cuando la traducción se hace de una vez dejando un resultado independiente del fuente, se dice *interpretación* cuando el ejecutable en realidad no llega a crearse sino que se va mandando a la máquina el conjunto de instrucciones adecuado conforme se lee el código fuente. En cada caso se debe, naturalmente, tener o bien un compilador o bien un intérprete que realice estas traducciones.

#### 1.4.5. El funcionamiento de un programa

Se denomina *ejecutar* un programa o aplicación a darle el control de la máquina.

Para ejecutar un programa (ejecutable) primero se debe *cargar* apropiadamente en la memoria RAM del computador. Esta operación la lleva a cabo una rutina del Sistema Operativo llamada ‘cargador’ (loader). Esta fase incluye el situar el código del programa en una zona libre de la RAM, en en situar los datos ya inicializados del mismo en otra zona y el reservar un espacio para la posible memoria que el programa pueda solicitar al sistema durante su ejecución. Esta memoria puede estar limitada de manera que el sistema disponga de otros bloques de memoria libre para otros posibles programas.

Una vez cargado el programa se salta con el Contador de Programa (PC) de la unidad de control al comienzo del mismo. Previamente, dado que el programa deberá saltar a otras partes del él mismo, se relocalizan sus direcciones anteriormente relativas para hacerlas ya absolutas antes de ser ejecutado. Esto también lo hace el cargador.

Cada vez que el programa requiera saltar a otra parte se hace cambiando el PC. En cada paso, accediendo a la dirección indicada por el PC, se lee de la RAM la instrucción a ejecutar y se aumenta, si no hay solicitados saltos el PC a la siguiente instrucción.

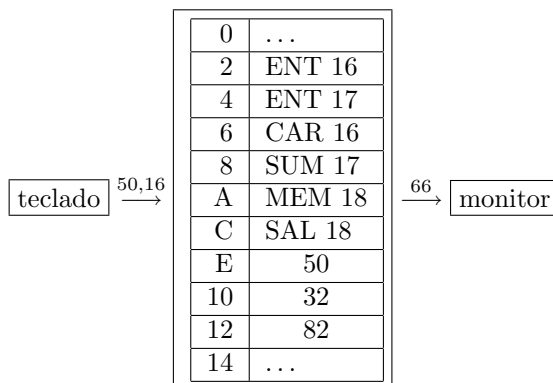
**Ejemplo** Supongamos que disponemos del siguiente repertorio de instrucciones máquina:

ENT $m$	Lee desde teclado una secuencia de dígitos numéricos, los codifica como un número entero y lo almacena en la posición de memoria $m$
SAL $m$	Escribe en la pantalla como cadena de dígitos decimal el contenido numérico de la posición $m$ de memoria
CAR $m$	Carga en la ALU un dato numérico proveniente de la posición $m$ de la RAM
MEM $m$	Escribe en la RAM el contenido numérico de la ALU
SUM $m$	Añade sobre el contenido de la ALU el de la posición de memoria $m$

Se desea escribir un programa directamente en lenguaje de esta máquina que sume dos números introducidos por el usuario del programa desde el teclado y presente el resultado en la pantalla.

Controlaremos directamente las posiciones de memoria sobre las que trabajaremos. Sean estas  $0E_{(16)}$  y  $10_{(16)}$  dejando el resultado de la suma en la  $12_{(16)}$ . Si programáramos en un lenguaje de alto nivel (como el Modula2), sería éste el que determinaría (junto con el cargador) el que determinaría

la posición de cada un de estos datos. Tendríamos:



## 1.5. El software básico de un sistema

Como hemos visto los ordenadores se pueden concebir como una parte *hardware* y otra *software*. De la parte ‘dura’ no nos preocuparemos en esta asignatura, pero sí de todo lo relativo a la estructura de su parte ‘lógica’<sup>11</sup>. Para estructurar la parte del software de un ordenador un criterio muy importante es el del **nivel** del software.

### 1.5.1. Niveles de abstracción

Una parte del software se dice de bajo nivel si sus procesos e instrucciones se refieren y controlan directamente los componentes físicos de la máquina.

### 1.5.2. El Sistema Operativo

Usualmente los ordenadores vienen con un *Sistema Operativo* (SO) que mediante instrucciones la mayoría de muy bajo nivel ofrecen conjuntos de instrucciones de más alto nivel, esto es, que realizan procesos más globales e independientes de los elementos físicos más particulares de cada componente hardware de la máquina. Por ejemplo, el SO puede tener entre sus muchos procesos, que se cargan al encender el ordenador y quedan allí hasta que se apague o se bloquee, procesos detallados para dibujar las líneas de texto que lo pueda querer ver, letra a letra, renglón a renglón en el monitor. Los detalles de en qué lugar de memoria hay que escribir cada letra sin embargo son muy particulares del ordenador, el monitor, el SO, etc. Sin embargo el SO ofrecerá un servicio, mediante una llamada a una rutina suya, con el cual se realizará todo el proceso de escritura de mensajes de texto en el monitor sin preocupar al que utiliza tal servicio con los detalles técnicos y características del monitor incorporado.

Mientras más alto es el nivel de software más independiente de los detalles técnicos de la máquina concreta será y por lo tanto si podemos poner en todos los ordenadores conjuntos de procesos que no se refieran a detalles, y sean suficientemente independiente, de muy alto nivel, tendremos la posibilidad de dar las mismas órdenes a distintos ordenadores y que estos las entiendan y realicen acciones similares.

### 1.5.3. Tipos de programas

Alrededor del hardware podemos entender entonces que está el SO y por encima de este están la multitud de aplicaciones, programas, rutinas de utilidad, fijadas todo el tiempo al SO o utilizadas esporádicamente. Procesos de muy distinto tipo que pueden estar continuamente funcionando o ser servicios añadidos al SO para completar sus funcionalidades, etc.

Veremos muy en detalle en las prácticas cómo existen programas que son capaces de construir otros programas. Éstos son los compiladores e intérpretes y nos permiten desarrollar nuevo software para el ordenador.

<sup>11</sup>Los franceses trataron de traducir ‘hardware’ por ‘material’ y ‘software’ por ‘logicial’

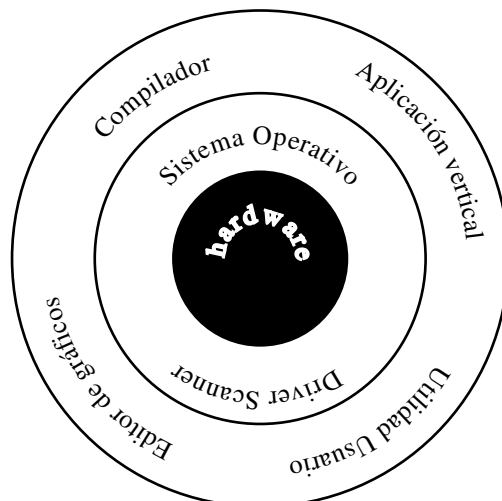


Figura 7: Estructura de capas y niveles del software

Aparte de estos particulares programas (que sólo utilizan los especialistas programadores), están las aplicaciones. Una aplicación puede estar más o menos orientada a una actividad concreta o ser de uso más general. Un programa que, por ejemplo, gestione la clientela de una Clínica Dental particular considerando sus características individuales, es considerada una aplicación ‘vertical’, mientras que un programa como un editor de textos es considerada una ‘aplicación horizontal’ o genérica.

También podemos hablar de rutinas de utilidad, ya sean para el mantenimiento del propio sistema, como las que se utilizan para limpiar la información de los discos, para buscar y eliminar ‘virus’, para observar el consumo de tiempo de las demás aplicaciones... como de utilidad para el usuario: una calculadora, un contador de palabras de ficheros del disco, etc.

Normalmente todas las aplicaciones están almacenadas en su uso normal, en el disco duro del ordenador<sup>12</sup>. Es pues el disco duro el que determina el estado y posibilidades más inmediatas del ordenador. En el disco duro se tiene como componente más importante el SO del ordenador que es lo primero que se carga al encenderlo y le da el estilo de trabajo. Después estarán una serie de aplicaciones instaladas por el usuario que podrán ejecutarse según haga falta.

## 1.6. Apéndice: Representación de números

**Representación de números negativos** Una vez determinado el tamaño del contenedor en la memoria de nuestro número entero, surge el problema de como representar su signo posiblemente negativo. Se conviene en poner un bit de signo delante con lo que el significado no es aritméticamente la conversión directa de la imagen en memoria y, además, se pierde este bit, en este modo de representación para todo el rango representable dentro del contenedor dado.

Por ejemplo, en un byte y suponiendo esta forma de representar directamente el signo con el bit más alto, el número  $12_{(10)}$  es  $0000\ 1100$ , mientras que  $-12_{(10)}$  sería  $1000\ 1100$ . Nótese que en ésta representación ya no ponemos el símbolo  $_2$  porque no es exactamente en base 2.

Los rangos serían:

bytes	rango	total
1 byte	$-127, -126, \dots, -0, 0, 1, 2, 127$	$2^8$
1 doble byte	$-32767, \dots, -0, 0, \dots, 32767$	$2^{16}$
4 bytes	$-2_{1147,483,647}, \dots, -0, 0, \dots, 2_{1147,483,647}$	$2^{32}$

Este sistema, pese a su sencillez, contiene dos problemas:

<sup>12</sup>Hoy día la tendencia es a interconectar todos los ordenadores y se habla de repositorios de información y programas a través de internet e independientes de cada ordenador.

1. Da una doble, un tanto absurda representación del 0, como  $-0$  y  $+0$ .
2. Las operaciones de suma con números negativos no son iguales a las de suma con positivos.

Es por esto que universalmente se utilizan las representaciones de los números negativos en complementos.

**Complementos para representar los números negativos** Hay dos tipos de complementos básicamente. En el sistema decimal son el complemento a 10 y el complemento a 9. El complemento a 10 (*ca10*), en general se expresaría como “complemento a  $\langle n \rangle$ ” siendo  $n$  el valor de la base de representación, en el caso de la decimal, 10; en la binaria, 2.

El complemento a 10 con de un número  $N$  con  $n$  dígitos es  $10^n - N$ .

Para calcular el ca10 de un número decimal, lo que se hace es restar cada dígito de la cifra 9 y luego sumar 1 al resultado total. Por ejemplo:

$$\begin{aligned} \text{ca10}(87) &\rightarrow 13 &= 10^2 - 87 \\ \text{ca10}(23) &\rightarrow 77 &= 10^2 - 23 \end{aligned}$$

El motivo de calcular el ca10 es que la resta se hace muy fácil si tenemos en cuenta que si la representación está limitada a  $n$  dígitos,  $N + 10^n = N$ :

$$r = N_1 - N_2 = 10^n + r = N_1 + (10^n - N_2) = N_1 + \text{ca10}(N_2).$$

Para restar dos números basta con *sumar* al sustraendo el ca10 del sustraendo ignorando, si apareciera, una nueva posición decimal a la izquierda<sup>13</sup>. Por ejemplo, en notación normal y en ca10, la resta de 89 menos 23 sería:

$$\begin{array}{r} 89 \\ -23 \\ \hline 66 \end{array} \rightarrow \begin{array}{r} 89 \\ +77 \\ \hline 166 \end{array}$$

Por supuesto, para que esto sea correcto hay que eliminar el 1 que “sobra” en el resultado, cosa, por otro lado obligada, ya que en todas las operaciones se utilizan dos dígitos.

Se dejan como ejercicio el calcular  $40 - 20$  en ca10. ¿Cuál es el ca10 de 00? ¿y el de 01?

Nótese que siempre *hay que dejar bien claro el número de dígitos con los que se está trabajando*.

**Complemento a 9** Aquí se resta, igualmente de 9 cada dígito del número, pero no se suma 1 al resultado. Por ejemplo  $\text{ca9}(23) = 76$  y  $\text{ca9}(87) = 12$ . Cuando se restan dos números, se hace como antes, pero sumando ahora el dígito de *acarreo*, antes despreciado, al resultado obtenido. El acarreo puede que sea o que no sea 0. El ejemplo anterior sería:

$$\begin{array}{r} 89 \\ -23 \\ \hline 66 \end{array} \rightarrow \begin{array}{r} 89 \\ +76 \\ \hline 165 \\ +1 = 66 \end{array}$$

**Complementos a 1 y a 2** En representación binaria el complemento a 10 se convierte en complemento a 2 (ca2) y el complemento a 9 en complemento a 1 (ca1). Por ejemplo en ca1:

$$\begin{array}{r} 11001 \\ -10110 \\ \hline 00011 \end{array} \rightarrow \begin{array}{r} 11001 \\ +01001 \\ \hline 100010 \\ +1 \\ \hline 00011 \end{array}$$

<sup>13</sup>Ver ejercicio 12

El ca2 de un número  $x$  con  $n$  dígitos es

$$2^n - x.$$

Por ejemplo, con 4 bits,

$$\text{ca2}(0011_{(2)}) = 10000_{(2)} - 0011_{(2)} = 16_{(10)} - 3_{(10)} = 13_{(10)} = 1101_{(2)}$$

El objetivo de este sistema es el de simplificar las operaciones de sumas y restas cuando se utilizan mezclados números positivos y negativos. Se puede así operar sin tener en cuenta el signo del operando. Veamos todos los casos:

1. Cambiar de signo:

$$\begin{aligned} \text{ca2}(x) &= 2^n - x \\ -(-x) &= -\text{ca2}(x) = 2^n - \text{ca2}(x) = 2^n - (2^n - x) = x \end{aligned}$$

2. Suma de números negativos:

$$\begin{aligned} (-x) + (-y) &= \text{ca2}(x) + \text{ca2}(y) \\ &= 2^n - x + 2^n - y = \underbrace{2^n}_{\text{acarreo}} + (2^n - (x + y)) \end{aligned}$$

pero  $2^n$  es un bit en la posición  $n + 1$  que aparecerá como acarreo y que no cabe en nuestros  $n$  bits y que como se hace siempre en la suma en ca2, se desprecia.

3. Suma de positivo pequeño más negativo grande

$$\begin{aligned} x - y &= x + \text{ca2}(y) \\ &= x + 2^n - y = 2^n - (y - x) \end{aligned}$$

que es un número válido en ca2 ya que el valor entre paréntesis es positivo.

4. Suma de positivo grande y negativo pequeño

$$\begin{aligned} x - y &= x + \text{ca2}(y) = x + 2^n - y \\ &= \underbrace{2^n}_{\text{acarreo}} + (x - y) \end{aligned}$$

que como en el caso 2 conlleva el desprecio de un bit de acarreo. Por otra parte este es el caso más simple de suma con negativos planteado al principio de este apartado sobre complementos, y ya se han planteado varios ejemplos de este caso.

Resumiendo, en este sistema se pueden sumar números positivos y negativos sin hacer diferencias entre ellos. Sólomente hay que ignorar los bits de acarreo superiores.

En el caso del complemento a 1, ca1, se tiene:

$$\text{ca1}(c) = 2^n - 1 - c.$$

Se deja como ejercicio comprobar los cuatro apartados anteriores.

**Ventajas del complemento a 2** La primera ventaja es que se tratan por igual, ante la suma, los números positivos que los negativos. El complemento a 2 es algo mejor que el ca1 por el hecho de que la suma es más simple, pues siempre es más fácil despreciar un dígito que tener que sumarlo, aunque, naturalmente en el momento del cálculo, sí se tiene que sumar ese dígito para obtener el ca2.

Sin embargo las operaciones de multiplicación son algo más complejas en cualquiera de los tipos de complemento que en el de la convención son el bit de signo.

El cálculo del ca2 es muy fácil: basta con hacer el “complemento lógico” de bits: cambiar 1s por 0s y viceversa y sumar 1 al resultado. Por ejemplo, 7 con  $n = 10$  bits es 0000000111 nos da en ca2 1111111001.

**Reparto de los valores en ca2** Con la convención más artificiosa del bit de signo explicada antes, la distribución una vez definido el tamaño del contenedor (1, 2 ó 4 bytes) de los valores era sencilla y simétrica, aunque se repetía el 0.

Con la representación en ca2 el número negativo mayor (en valor absoluto) representable será aquel que “nos quepa” en el contenedor después de ser restados sus dígitos de 1 y sumado al resultado 1. Si repartimos por igual los positivos y los negativos, los números negativos tendrán siempre el bit más alto (msb) en 1, después de haberse hecho un ca2. Para calcular su valor real habrá que recalcularles su ca2 y ponerles el tradicional – delante.

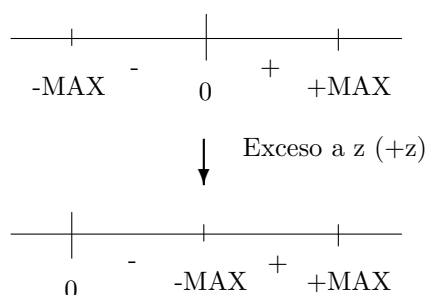
Si el contenedor tiene  $n$  bits, el número  $2^{n-1}$  que tendrá todos los bits a 0 menos el último más alto (msb) tiene la representación “en negativo” como ca2 siguiente: complementando a 2 cada dígito obtenemos todos a 1 menos el último que queda a 0; sumando 1 obtenemos todos a 0 menos el último que queda a 1. El siguiente,  $2^{n-1} + 1$  nos daría un ca2 que tendría el msb a 0 con lo cual no sabríamos si el valor original era positivo o negativo. Así pues, interpretando como negativos, ya en ca2, los números con el msb a 1, tendremos los siguientes rangos:

bytes	rango	total
1 byte	$-128, \dots, 0, \dots, 127$	$2^8$
1 doble byte	$-32768, \dots, 0, \dots, 32767$	$2^{16}$
4 bytes	$-2^{147}, 483, 648, \dots, 0, \dots, 2^{147}, 483, 647$	$2^{32}$

Se deja como ejercicio el hacer las representaciones en 1 byte de 124,  $-124$ ,  $-1$ , 0, 127,  $-127$ .

**Extensión de signo** Muchas veces es necesario cambiar el tamaño del contenedor de una información numérica. Esto en complemento a 2 podría involucrar quizás, en principio, el cálculo del número original en valor absoluto y el ca2 en el nuevo contenedor, ya que como sabemos, el ca2 depende siempre del tamaño del contenedor. Sin embargo esto no es así, es fácil demostrar que cuando se transfiere un número a un contenedor más largo, sea de  $n$  a  $m$  dígitos ( $m > n$ ), basta con rellenar con el valor en el bit  $n$ -simo del primero, los nuevos  $m - n$  bits del segundo. O sea, si era negativo, rellenar con 1s los nuevos bits de la izquierda, si positivo, con ceros.

**Representación en exceso a  $z$**  Esta representación se utiliza para los exponentes de los números en punto flotante, como veremos. En él, los números se representan en binario pero incrementando su valor en  $z$ . Se trata sencillamente de desplazar los números negativos por encima del 0.



Esto es

$$\text{caz}(x) = x + z$$

en binario. Para que haya un reparto igualado entre los números negativos y los positivos, mitad y mitad, debe ser  $z = 2^{n-1}$ , esto es se desplaza la mitad del rango. De esta forma los números originalmente negativos tienen en esta representación, tras sumarle  $z$ , la mitad baja del rango (msb a cero), mientras que los originalmente positivos quedan en la mitad alta (tienen al final el msb a 1).

El rango de valores representables será:  $-2^{n-1} \dots 0 \dots 2^{n-1} - 1$  como con el ca2. Por ejemplo, en el caso 8 bits,  $z = 2^7 = 128 = 100\,0000$ , y tendríamos:

codificado:	0	...	127	128	...	255
representa:	-128	...	-1	0	...	127

Si el número  $n$  original es negativo, el cálculo del *caz* es en realidad similar al cálculo del *ca2* pero teniendo un tamaño de un bit menos, ya que lo que se hace es en realidad  $z - |n|$  siendo  $z$  uno más del valor representable dentro de  $n - 1$  bits. Así el resultado del *caz* con números negativos es equivalente al de *ca2* excepto que el *ca2* tendría el *msb* a 1. Con los números positivos ocurre lo mismo. Si es positivo, el *ca2* es el mismo número, y bastaría sumarle  $z$ , esto es ponerle el *msb* a 1.

Una vez calculado el *caz*, para ver cual sería el número original bastaría con restar  $z$  del valor representado directamente. Así, con  $z = 2^{n-1}$ , ya que  $z = 2^{n-1}$  es un 1 (el *msb*, seguido de ceros) la representación equivale a la *ca2* excepto que tiene el *msb* invertido.

Ejemplos:

$n_{(10)}$	$ca2(n)_{(2)}$	$caz(n)_{(2)}$	$caz(n)_{(10)}$
-3	1111 1101	0111 1101	125
0	0000 0000	1000 0000	128
-128	1000 0000	0000 0000	0
127	0111 1111	1111 1111	255

con rango  $-128 \dots 127$ .

**BCD** En los sistemas BCD la representación de los dígitos decimales va convirtiendo independientemente dígito a dígito, a binario mediante alguna tabla de conversión convenida.

Para poder representar cada dígito decimal se necesitan al menos 4 bits. Es un caso particular del código ASCII, con el que se pretendían codificar más símbolos y se llegan a usar 7 bits (u 8, en las ASCII extendidas normales).

Ahora bien, 4 bits permiten 16 dígitos distintos, se desprecian, pues, el  $(16 - 10)/16 = 37,5\%$  Su interés es relativo. Se compensa porque la entrada y salida de datos en los ordenadores se hace mediante cadenas de dígitos más o menos largas y, por tanto, esta codificación es muy segura y rápida. Y, además, porque los errores de redondeo aquí no existen y en muchas aplicaciones, especialmente las bancarias y, en general, las de gestión, no deben darse errores de redondeo (¡ni de ningún otro tipo!, se supone).

Existen varias formas de tabular las representaciones de los dígitos decimales (del 0 al 9), la más sencilla y directa es la que normalmente es llamada BCD, y establece  $0 \rightarrow 0000$ ,  $1 \rightarrow 0001$ ,  $\dots$ ,  $9 \rightarrow 1001$ . Pudiéndose utilizar las posibilidades restantes para indicar signo o valores absurdos, como infinito, etc. Pero existen además el **2421**, cuyo nombre nos indica los pesos de cada posición. El sistema en exceso a 3 (en que los dígitos se representan incrementados en 3) y que simplifica los cálculos aritméticos. El 2 entre 5, empleado en comunicaciones, que siempre tiene dos bits a 1 y 3 a 0, para las verificaciones en la transmisión.

Las **operaciones aritméticas** se hacen dígito a dígito (grupos de 4 bits) siguiendo las reglas convencionales. El único inconveniente es el de tener que vigilar el que nos excedamos de 9 en las sumas, cosa que no indica aquí ningún bit de acarreo. Sin embargo en el código en exceso a 3 sí lo indica el bit de acarreo.

El código BCD es ún muy utilizado en lenguajes de gestión como el COBOL y otros. Aunque es muy semejante al ASCII, la longitud de una representación en BCD es la mitad que la ASCII, cabiendo en cada byte, dos dígitos decimales.

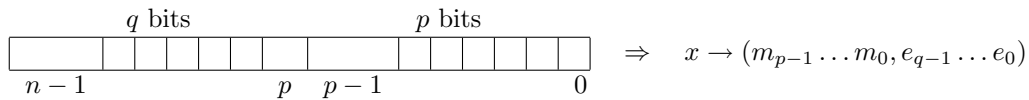
**Representación en coma flotante** Los sistemas hasta ahora vistos son capaces de representar números enteros con o sin signo codificado o en complemento. También se pueden representar fácilmente con estos sistemas los que se denominan números en punto fijo, esto es, número en los cuales la posición de la coma (o punto) decimal tiene una posición fija. Para ello basta con en la representación que sea se multiplique por el factor de escala *fijo* que sea. El escalado de estos números puede dar cabida a números con decimales pero esto está propenso a errores y desbordamientos en el caso de que el tratamiento aritmético de los números no sea sencillamente el de sumas y restas.

Es evidente que es necesario que exista un mecanismo interno y una representación adecuados al manejo de cantidades reales con la posición del punto decimal variable según los cálculos. Así pues, los sistemas de manejo de punto flotante deben almacenar a la vez, para cada número,

unos dígitos representativos y un factor de escala.

Curiosamente las primeras antiguos ordenadores (Zuse, Stilbitz...) tenían coma flotante y fue von Neumann en 1946 quien despreció, por trivial, su uso en su nuevo modelo computacional. Hoy día, desmintiendo a von Neumann, todos los ordenadores los usan.

En la representación en coma flotante, se dividen los números de  $n$  bits en dos partes en general: una para los dígitos significativos y otra para el factor de escala, como exponente de una potencia  $r$ . A cada parte se le llama *mantisa* y *exponente* y tienen unas longitudes de  $p$  y  $q$  bits, respectivamente, siendo  $p + q = n$ , el tamaño de la representación.



con

$$V(x) = M \cdot r^E.$$

La mantisa  $M$  y el exponente  $E$  se representan en alguno de los sistemas de representación de coma fija ya estudiados y conviene que el exponente utilice la misma base que sirve para la representación de los números de la mantisa; de esta forma, un desplazamiento a la izquierda o a la derecha de la coma corresponde a un decremento o incremento de  $E$ . El valor usual de  $r$  es 2 (y alguna vez 16).

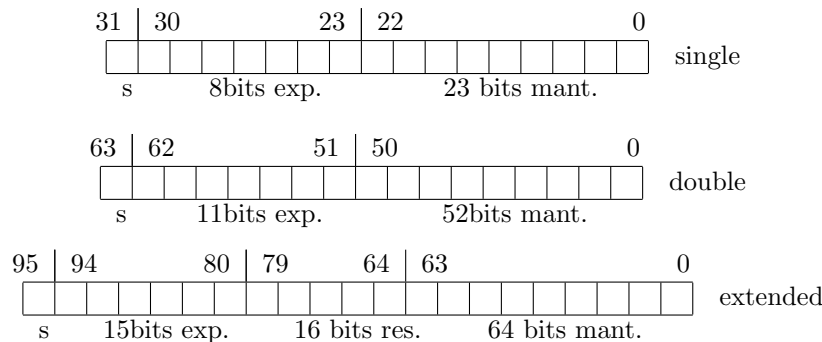
La mantisa suele expresarse como fracción, esto es:

$$\text{si } r = 2; \quad 0,0875 \rightarrow \underbrace{101011}_{\text{mantisa}} \quad \underbrace{00101}_{\text{exponente}}$$

El exponente suele representarse en base 2 y en exceso a  $z = 2^{q-1}$ .

**El estándar IEEE p754** Utiliza tres formatos básicos: simple precisión con 4 bytes, doble precisión, con 8, y precisión extendida, con 12 bytes.

El reparto de la información en cada uno es:



Estas representaciones emplean mantisa fraccionaria (un número decimal en base 2) y en forma de bit de signo independiente, que siempre se almacena en el bit más alto de la representación. La base del exponente es  $r = 2$  y el exponente se representa en exceso  $z = 2^{q-1} - 1$ . Las mantisas de los formatos simple y doble se suponen con un 1... delante que no se guarda. De esta forma los valores representables tiene los rangos:

$$\begin{aligned} (-1)^s \times 1.M \cdot 2^{E-127} & \quad \text{para simple precisión} \\ (-1)^s \times 1.M \cdot 2^{E-1023} & \quad \text{para doble precisión} \end{aligned}$$

Nótese que los exponentes tienen para  $z$  los valores

simple	127
doble	1023
extendido	16383

En el caso de los formatos simple y doble, la información siempre debe estar normalizada estando la mantisa entre 1.0 y 2.0, pero, como se ha dicho, con la parte entera omitida (*hidden bit*), con lo que se gana un bit de precisión. Por supuesto, el exponente se ajusta para conseguir esta mantisa entre 1.0 y 2.0.

En el formato extendido, sin embargo, guarda explícitamente este bit, despreciando la ganancia en precisión.

Los formatos pueden también guardarse ‘denormalizados’ (*denormalized*). Cuando un número en formato punto flotante tiene un exponente 0, el número está cerca del menor valor permitido. Con el objetivo de preservar el mayor rango posible de valores representables, se permite que la mantisa tenga su bit más alto a cero (cosa imposible en los formatos normalizados). En el formato extendido tenemos que tener el bit entero a cero además de cero en el exponente, pero en los de simple y doble precisión hay que suponer que el *hidden bit* es cero cuando el exponente es nulo.

El formato extendido también permite a un número tener un bit entero a cero sin tener el exponente a cero. A tal formato se le denomina ‘innormalizado’ (*unnormalized*). Sin embargo estos formatos no se usan internamente durante el cálculo y es necesario reconvertirlos.

Existen además valores que no tienen sentido como números al no estar normalizados pero que reciben un significado extra:

$E = 255$	$M \neq 0$	resultado sin sentido (p.e.: 0/0)
$E = 255$	$M = 0$	$\pm\infty$ según bit de signo
$E = 0$	$M = 0$	0
$E = 0$	$M \neq 0$	núm. pequeño desnormalizado: $(-1)^s \times 2^{-126} \times 0.M$

También se podrá hacer en formato doble con el exponente 1022 en vez de 126. Los números desnormalizados permiten un salto entre valores más fino en la región cercana al 0, dando una distribución de errores más uniforme. Esto permite representar módulos desde:

$$2^{-126} \cdot 0,111 \dots 11 \quad \text{hasta} \quad 2^{-126} \cdot 0,000 \dots 01$$

También se podrá hacer en formato doble con el exponente 1022 en vez de 126. De esta forma se pueden llegar a representar  $2^{23}$  ó  $2^{52}$  números entre el 0 y el  $2^{-126}$  (o entre el 0 y el  $2^{1022}$ ) con separación uniforme entre ellos.

Como ejemplo, vaya ahí el valor, en formato extendido de una de las constantes que almacena internamente el microprocesador Motorola 68882 (coprocesador anexo a la CPU MC68030):

$$\pi \approx +3,1415926535897932 \rightarrow 4000 \quad \text{c90fdaa22168c235}$$

Se deja como ejercicio el comprobarlo.

Los rangos de los números en simple y doble precisión con 4 y 8 bytes son:

precisión	notación ordenador	notación matemática
simple	1.2E-38...3.4E38	$1,2 \cdot 10^{-38} \dots 3,4 \cdot 10^{38}$
doble	2.3E-308...1.7E308	$2,3 \cdot 10^{-308} \dots 1,7 \cdot 10^{308}$

**Resolución y error relativo** Mientras que en los sistemas en coma fija la resolución es uniforme e igual a 1, para el caso de los representaciones en punto flotante variará a lo largo del rango, en virtud del exponente que le afecte.

Considerando sólo la mantisa, su resolución es uniforme y viene dada por una unidad del dígito menos significativo.

## 1.7. Apéndice: Historia del transistor

Tomado de el periódico “El País” (Miércoles 10 diciembre de 1997).

### El transistor, cincuenta años después

Emilio Méndez, catedrático de Física en la Universidad del Estado de Nueva York en Stony Brook.

Mañana hace 50 años, John Bardeen y Walter Brattain, de los Laboratorios Bell (entonces de ATT), hicieron el descubrimiento que cinco días más tarde los llevaría a la invención del transistor: la propagación de carga eléctrica positiva a través de un semiconductor cargado mayoritariamente con electrones. El 23 de diciembre de 1947 los científicos demostraban un transistor capaz de amplificar 15 veces la voz humana.

A pesar del entusiasmo de ATT, que llevaba casi una década buscando un nuevo dispositivo para reemplazar las válvulas de vacío, la noticia de la invención del transistor (aparecida en *The New York Times* el 1 de julio de 1948 en una oscura sección sobre Noticias de la Radio) tuvo poco eco, y al principio su utilidad se limitó a la telefonía, la radio, y los aparatos contra la sordera. Cincuenta años después, no es arriesgado llamar a nuestra época la Edad del Transistor, tan amplio ha sido su impacto.

El transistor original, de contacto por puntas, era un dispositivo muy primitivo, de difícil operación y poco fiable. Pronto dio paso al más sencillo y fácil de fabricar transistor de unión bipolar de William Shockley, que sólo en los últimos años ha sido desplazado parcialmente por el transistor de efecto campo debido al menor consumo de energía y ventajas de fabricación de éste. Ambos tipos pueden funcionar como un interruptor o como un amplificador, gracias a su capacidad de variar la conductividad eléctrica entre los dos extremos de un material semiconductor mediante un tercer terminal de control. La otra gran aplicación es como almacenadores de información.

**Premio Nobel** Pocas veces se ha concedido con mayor justicia el premio Nobel que cuando se entregó hoy hace 41 años a Shockley, Bardeen y Brattain “por su investigación en semiconductores y su descubrimiento del efecto transistor”.

Las técnicas inventadas a finales de los años cincuenta para el grabado (litografía) en una misma oblea de silicio de circuitos integrados por varios transistores han hecho posible una disminución exponencial en el coste y el tamaño de los dispositivos, y la creación de funciones cada vez más complejas, que han culminado con los microprocesadores actuales, poderosos ordenadores en un solo circuito integrado.

A principios de los años cincuenta, fabricar un transistor costaba unas 600 pesetas, lo que hoy cuestan 40 millones de ellos. En 1970, el tamaño típico de un transistor era de unas 12 micras, o una décima del grosor de un cabello; en 1997 es 40 veces menor. En los últimos 30 años el número de transistores en un circuito integrado se ha duplicado aproximadamente cada 18 meses. Esta trayectoria, que se conoce como ley de Moore, no es por supuesto el resultado de ninguna ley física sino un objetivo de desarrollo que la industria electrónica se ha marcado, y que hasta ahora ha podido cumplir a base de una competencia feroz y de la inversión de enormes recursos.

Es difícil predecir hasta cuándo se mantendrá la ley de Moore, pero si se siguiera cumpliendo en los próximos 10 años, para entonces el número de transistores en un circuito de 10 centímetros cuadrados sería de 20.000 millones, cada uno de 0,1 micras.

El mayor desafío que impone la ley de Moore es desarrollar la tecnología para el proceso de litografía, en el que se usa luz con longitud de onda comparable al tamaño de los transistores. Actualmente, la fabricación de los dispositivos más avanzados es posible con el empleo de láseres ultravioleta de 0,24 micras, pero los instrumentos ópticos necesarios para la siguiente generación de dispositivos no se comercializarán hasta el año 2000. Y teniendo en cuenta que las lentes de cuarzo no transmiten la luz por debajo de las 0,15 micras, la fabricación de dispositivos con dimensiones menores exigirá un cambio drástico en los sistemas de litografía.

Se barajan otras posibles tecnologías de grabado para cuando se alcance este límite, por ejemplo rayos X o haces de electrones, pero los sistemas desarrollados hasta ahora distan mucho de ser prácticos para la producción de circuitos a gran escala. Aunque estas barreras parecen de momento insuperables, los avances anunciados hace apenas tres meses dan pie al optimismo.

Un transistor actual puede almacenar una unidad de información digital (conocida como un bit) mediante la presencia (“uno”) o ausencia (“cero”) de una cierta cantidad de carga eléctrica. Por tanto, la capacidad de los dispositivos de memoria conocidos como RAM (random access memory) está limitada por el número de transistores que es posible integrar en un circuito. A mediados de septiembre, Intel anunció una nueva generación de circuitos que distinguen dos estados intermedios (un tercio y dos tercios) además de los ceros y unos de los dispositivos convencionales. Estos cuatro niveles permiten almacenar dos bits por transistor, duplicando así la densidad de información sin necesidad de reducir ninguna dimensión.

Una semana después del anuncio de Intel, IBM hizo público el desarrollo de un nuevo método para fabricar circuitos integrados, en los que el cobre reemplaza al aluminio para las conexiones metálicas entre los transistores. Este avance llega en un momento clave. A medida que los transistores se han reducido de tamaño y su funcionamiento interno se ha hecho más rápido, el tiempo debido a las conexiones se ha hecho cada vez más importante, hasta el punto que por debajo de las 0,2 micras retardará las operaciones del circuito. El empleo del cobre, con mejor conductividad eléctrica que el aluminio, eliminará este problema a la vez que abaratará la producción de los circuitos.

Otra gran barrera para futuras generaciones de circuitos integrados es económica, ya que los gastos de producción también siguen una ley exponencial, duplicándose cada cuatro años. Pero el desafío último será físico: confinados en dimensiones por debajo de las 0,1 micras, los electrones dejan de obedecer las reglas de la física clásica y su número es tan bajo que hay que tener en cuenta el comportamiento individual de cada uno de ellos. Científicos en Japón, Europa y Estados Unidos trabajan febrilmente en diseños de transistores que funcionen aprovechando las propiedades cuánticas de los electrones o el movimiento de un único electrón, y en ideas radicalmente nuevas, en busca de un dispositivo tan distinto del transistor actual como éste lo fue en su día de las válvulas de vacío.

© Copyright DIARIO EL PAIS, S.A.

## 1.8. Ejercicios

- ▷ 1 Representar los números  $1994_{(10)}$  y  $-123_{(10)}$  en complemento a 2 en dos bytes. Representar el resultado en binario, octal y hexagesimal.
- ▷ 2 Dados los números 1000, 0101, 1010, 1111 y 0111; representarlos en octal y hexagesimal. Convertirlos en su equivalente decimal suponiendo a), que están codificados en complemento a dos, b), que están codificados en binario puro y c), que están codificado con bit de signo. En todos los casos en que haga falta suponer que el tamaño de la representación es de 1 nybble (medio byte).
- ▷ 3 Consultar una tabla ASCII para mostrar el aspecto de las siguientes cadenas de caracteres mostradas entre comillas (no se incluyen): a) ‘Hola’; b) “Hoy es 13”; c) ‘1994/3’.
- ▷ 4 Convertir los números  $6,2836_{(10)}$  y  $3,1416_{(10)}$  a binario
- ▷ 5 Representar en BCD el número  $1456_{(10)}$
- ▷ 6 Dado el número  $1456_{(10)}$ ; representarlo en BCD, en BCD en incremento a tres, en BCD 2421 y en BCD 2 entre 5. Previamente hacer las tablas de conversión de los dígitos decimales.
- ▷ 7 Se pretende codificar el sonido de un instrumento musical mediante un muestreo de 11.000 números positivos de 16 bits. Si disponemos de 100 Mb de RAM y la grabación es en estéreo, ¿qué tiempo en segundos podríamos almacenar?
- ▷ 8 Si en una conexión internet mediante un navegador la página que estamos bajando contiene una imagen de 60Kb y nuestra módem es de 33Kbaudios, ¿qué tiempo en segundos, como mínimo, tardará en ser bajada?
- ▷ 9 Codificar la palabra ‘Carlos’ mediante sus códigos ASCII. Dar los números en decimal y en hexagesimal.
- ▷ 10 Localizar en la bibliografía una tabla ASCII de MSDOS y otra de Windows, compararlas. Codificar la frase “España va bien” en ambas en hexagesimal. ¿Cuántos bytes ocuparía?
- ▷ 11 Representar y comparar las longitudes de las representaciones de ‘1998’ en los formatos BCD, binario y ASCII.
- ▷ 12 Demostrar matemáticamente que dentro de un contenedor capaz para  $0 \dots N - 1$ ,  $x + \text{caN}(y) = x - y$ .

## 1.9. Referencias de consulta

Sobre la historia de la informática se pueden encontrar algunos textos corrientes en castellano. [Cle] es un texto de lectura muy fácil con curiosas anécdotas sobre la historia, siendo además riguroso y completo. [LG84] Es un texto de fácil lectura con un capítulo inicial de esquemática introducción histórica. Aunque es algo antiguo en su tratamiento, el texto general permanece vigente y es organizado tratando todos los tópicos que puedan sonar extraños al principiante.

[Kog91] Es un texto de amplitud y profundidad que exceden el nivel de este curso, pero tiene una interesante descripción del modelo de von Neumann en su primer capítulo.

Un poco más de referencias y anécdotas históricas se puede encontrar en el texto un poco anticuado de Pardo Clemente: [PC86].

La historia de la notación posicional del Apéndice 1.4.1 está tomada de [AKL79].

## Referencias

[AKL79] A.D. Aleksandrov, A.N. Kolmogorov, and M.A. Laurentiev. *La matemática*. Alianza Universidad, 1979.

[Cle] Ezequiel Pardo Clemente. *Informática General*. Ediciones Júcar, ?

[Kog91] Peter M. Kogge. *The architecture of symbolic computers*. McGraw-Hill, 1991.

[LG84] Guillermo Levine Gutierrez. *Introducción a la computación y a la programación estructurada*. McGraw-Hill, 1984.

[PC86] Ezequiel Pardo-Clemente. *Informática General*. Ediciones Júcar, 1986.