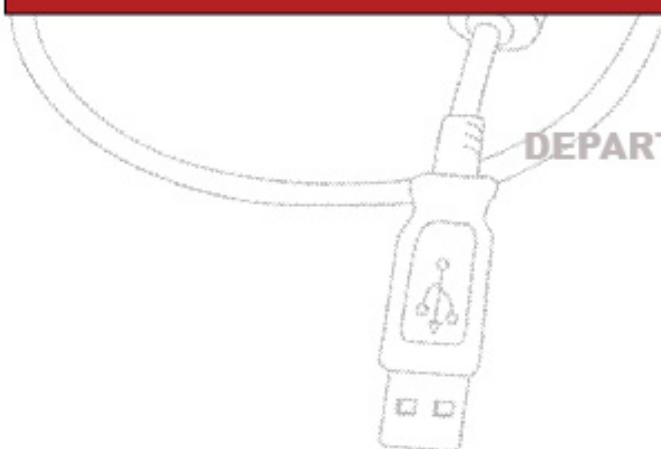


HW

1 SIR



Fundamentos



DEPARTAMENTO DE INFORMÁTICA
I.E.S. ZAIDÍN-VERGELES

1. Introducción.

La **Informática** trata de la adquisición, representación, tratamiento y transmisión de la información¹. El objeto de estudio de la informática es el ordenador. Un ordenador se puede considerar un **sistema de información**, entendiendo que está formado por un conjunto de elementos que emiten, reciben e interpretan información.

En estos procesos se representa la información en magnitudes físicas. Cuando las magnitudes físicas pueden tomar un valor cualquiera dentro de un rango prefijado se dice que el sistema correspondiente es un **sistema analógico**.

Ejemplo: La velocidad de un vehículo se mide generando una tensión proporcional a la velocidad, la cual puede tomar un valor cualquiera desde 0 hasta un valor máximo, igual que la tensión a la que representa.

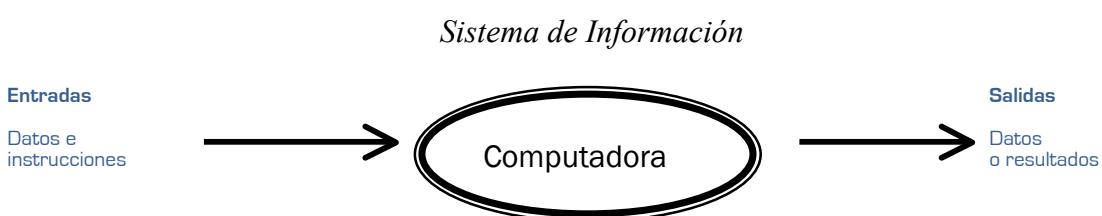
Frente a los sistemas analógicos se encuentran los **sistemas digitales** en los que las magnitudes físicas que se utilizan para representar la información sólo toman valores discretos dentro de un intervalo dado.

Ejemplo: Un sistema que utiliza señales de tensión para representar la información en el que estas señales toman sólo los valores – 5 V., 0 V. y 5 V. es un sistema digital.

Un sistema digital restringido a sólo dos valores discretos se dice que es un **sistema digital binario**. Concretamente, las computadoras son sistemas digitales binarios, pues utilizan sólo las dos cifras binarias o bits² (0 y 1) para representar la información.

Ejemplo: Podemos comparar un bit con el estado de una bombilla: 0 apagada y 1 encendida.

Las computadoras son *máquinas de propósito general* capaces de aceptar unos datos de entrada, efectuar con ellos *operaciones lógicas* (comparar, seleccionar, copiar ...) y *aritméticas* (sumar, restar, etc.) y proporcionar unos datos de salida, todo ello bajo el control de un conjunto de instrucciones (**programa**) previamente almacenado en la propia computadora. Por tanto, puede considerarse una computadora u ordenador como un sistema de información en el que las salidas o resultados son función (dependen) de sus entradas, constituidas éstas por **datos e instrucciones**.



¹ También podemos definir **informática** como la ciencia que se ocupa del estudio del tratamiento racional de la información por medio de las computadoras.

² Bit = **B**inary **d**igit. En el ordenador un bit es físicamente una celda de memoria (constituida por transistores o un transistor y un condensador), un punto magnético en un disco o una cinta, o un pulso de alto o bajo voltaje viajando a través de un circuito. Grupos de bits forman unidades de almacenamiento en el computador llamadas **bytes** (8 bits).

El término **informática** es un neologismo de origen francés resultado de la contracción de los vocablos **información** y **automática**. En los países anglosajones se utiliza **Computer Science** como término análogo.

2. Sistemas informáticos.

2.1. Definición de Sistema Informático.

Un Sistema Informático es un conjunto de dispositivos con al menos una CPU o Unidad Central de Proceso (en sus siglas en inglés) que están conectados entre sí a través de canales (modo local) o a través de líneas externas de comunicación (modo remoto). Dichos elementos se integran por medio de una serie de componentes lógicos o software con los que pueden llegar a interaccionar uno o varios agentes externos, entre ellos el hombre.

2.2. Elementos de un Sistema Informático.

Todo Sistema de Información dispone de dos elementos básicos: el hardware y el software.

La palabra **hardware** (*lit. ferretería, quincalla*) define el **soporte físico**, la maquinaria capaz de procesar o mantener información, en otras palabras, la parte tangible de la informática. Hay que tener en cuenta que no hace referencia a la tecnología empleada ya que el concepto es independiente de su implementación que podrá ser mecánica, electromecánica, electrónica o de cualquier otra tecnología. Refiriéndonos a una computadora, hardware es el conjunto de circuitos electrónicos, cables, armarios, dispositivos electromecánicos y todos los elementos físicos que la componen.

Por otra parte, el término **software** de una computadora define el **soporte lógico** de la misma, es decir al conjunto de instrucciones que llevan a cabo una tarea determinada y los datos que utilizan estas instrucciones. Es la parte no tangible de los ordenadores. Las instrucciones se hallan agrupadas en conjuntos lógicos que tienen un fin común y que se denominan programas. Para que una computadora funcione es necesario utilizar estos programas (pues una computadora con tan sólo soporte físico no funciona).

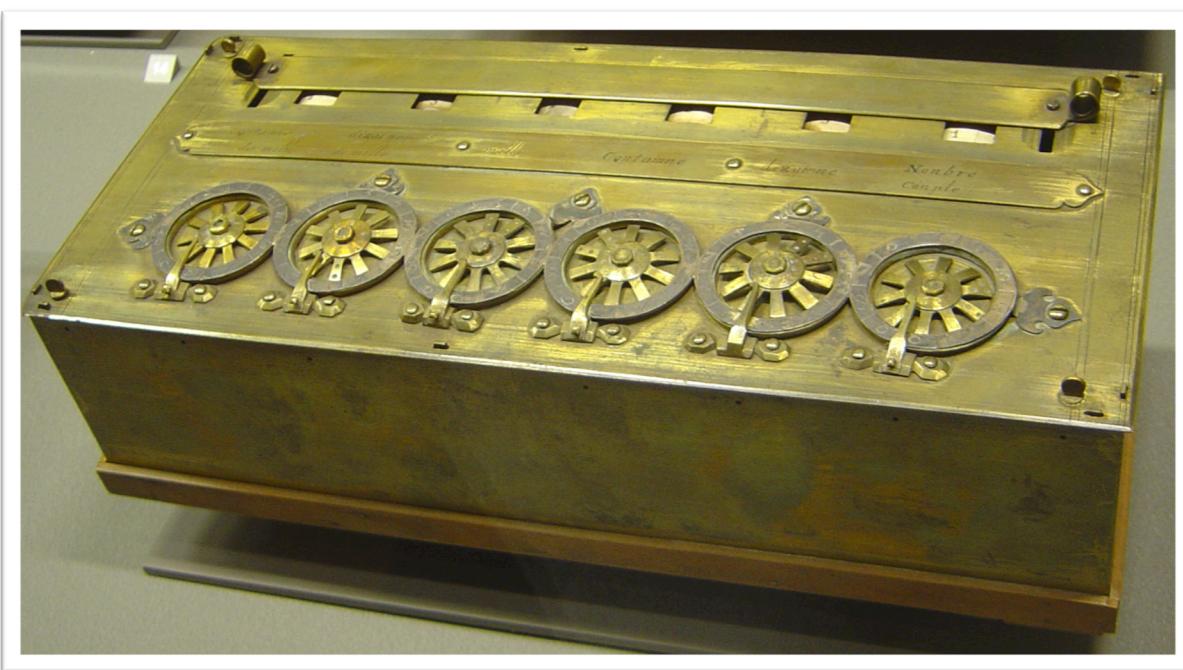
La palabra hardware no sólo se utiliza para designar los dispositivos físicos de la computadora, sino también todo lo relacionado con ellos. Así la Teoría de la Conmutación³, la Electrónica, etc., son disciplinas directamente relacionadas con el hardware. Lo mismo podemos decir del software. Software no son sólo los programas de una computadora en concreto sino también todas las materias relacionadas con el desarrollo de programas, como por ejemplo la Ingeniería del Software (Análisis y diseño de aplicaciones, Lenguajes y Sistemas Informáticos, etc.).

³ Trata del diseño de circuitos basándose en el álgebra de Boole.

2.3. Evolución Histórica desde un punto de vista físico.

Las computadoras, entendidas como máquinas que procesan datos, no se pueden considerar un invento reciente, sino que tiene detrás un largo proceso evolutivo que comenzó con objetos como el **ábaco**, utilizado por antiguas culturas como la china para realizar cálculos sencillos, y que llega hasta nuestros días.

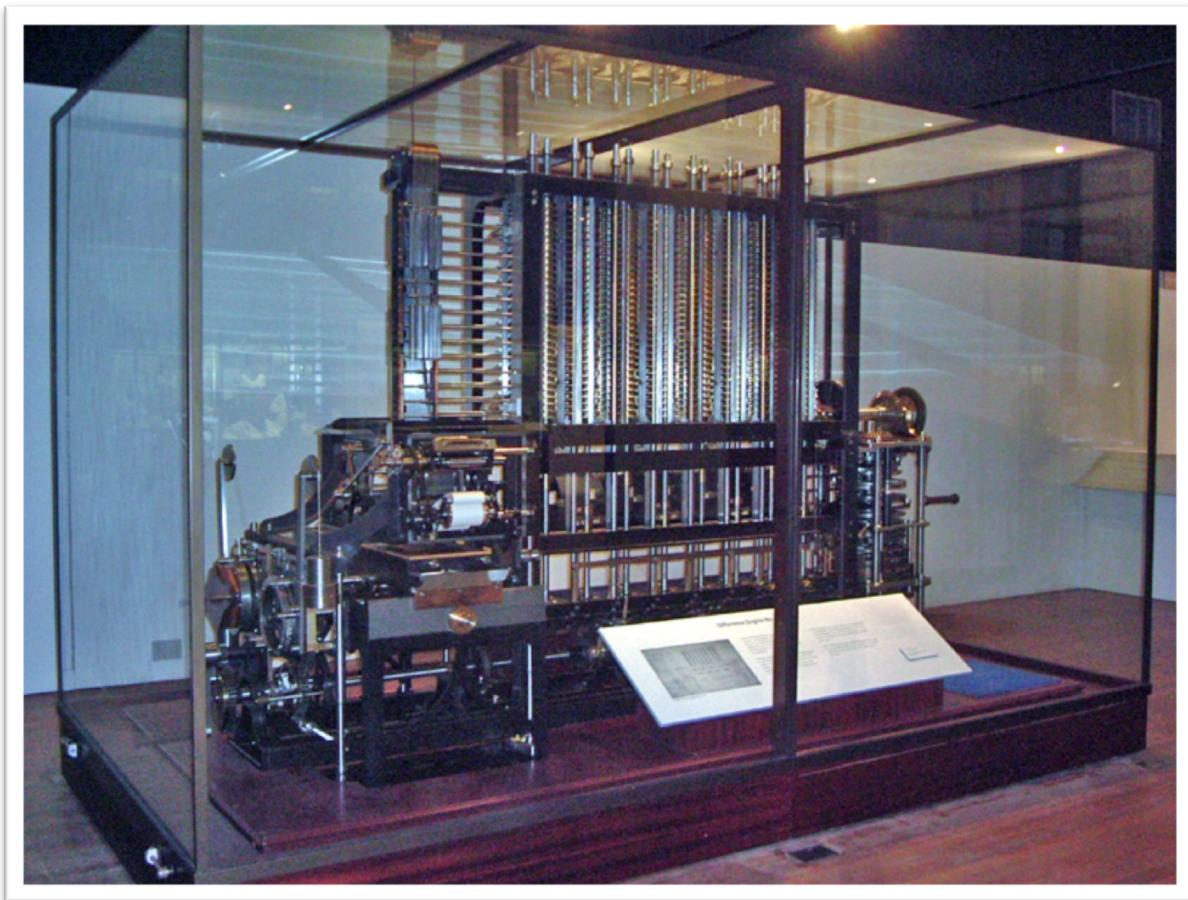
En el s. XVII en Europa, el interés creciente por la navegación y las nuevas ciencias como la astronomía, impulsó el desarrollo de lo que podemos denominar como calculadoras mecánicas. En 1614 **John Napier** inventó las tablas logarítmicas que permitían efectuar complejas multiplicaciones como simples sumas. En 1642 **Blaise Pascal** inventó una máquina capaz de efectuar sumas mediante un sistema de ruedas dentadas que fue llamada *la Pascalina*. Posteriormente, en 1671, **Leibnitz** aumentó su versatilidad agregándole la posibilidad de restar, multiplicar y dividir.



En el s. XIX se dará un nuevo empuje al cálculo automático de manos de **Charles Babbage**⁴, quien diseñó la primera computadora de uso general llamada “**Máquina Diferencial**” que posteriormente sería mejorada y renombrada como “**Máquina Analítica**”, aunque ninguna de las dos llegarían a construirse. Los descubrimientos de Babbage interesaron a su ayudante **Lady Ada Byron**⁵, considerada el primer programador de la historia.

⁴ **Charles Babbage** (Teingmouth 1791-1871) fue un matemático e ingeniero británico considerado el inventor de las máquinas calculadoras programables. Diseñó y parcialmente implementó una máquina a vapor, de diferencias mecánicas, para calcular tablas de números. También diseñó, pero nunca construyó, la **máquina analítica** para ejecutar programas de tabulación o computación; por estos inventos se le considera como una de las primeras personas en concebir la idea de lo que hoy llamaríamos una computadora, por lo que se le considera como "El Padre de la Computación". Dedicó sus últimos años y recursos a inventar una máquina infalible que fuese capaz de predecir los ganadores de las carreras de caballos. En el Museo de Ciencias de Londres se exhiben partes de sus mecanismos inconclusos.

⁵ **Lady Ada Byron** (1815-1852), hija del famoso poeta, colaboró con Babbage y predijo que su máquina analítica podría ser usada tanto para un uso práctico como científico. Sugirió a Babbage escribir un “plan” para que la máquina calculase números de Bernouilli. Este “plan” es considerado el primer programa de



La Máquina Analítica de Babbage

En 1804, **Joseph Jacquard** inventó un telar que se servía de tarjetas perforadas para controlar la creación de complejos diseños textiles. Estás tarjetas estaban hechas de papel, cartón o plástico con unas perforaciones distribuidas de forma que representan información (en binario para las computadoras). Fueron usadas posteriormente por **Herman Hollerith** en 1890 como soporte de un sistema mecánico que desarrolló para llevar a cabo el censo de los Estados Unidos, realizándolo en sólo dos años y medio en lugar de los 7 que se tardaba anteriormente. Fundó la Tabulating Machine Company que más tarde se uniría a otras compañías formado la Computing-Tabulating-Recording-Company que posteriormente pasaría a llamarse *International Business Machines (IBM)*. Por entonces ya estaba claro que el sistema binario era el que daba mejor soporte a un ordenador. A los dispositivos que almacenaban una cifra binaria se les denominó biestables (dos estados) y su evolución desde el punto de vista electrónico fue determinante en los siguientes pasos que se dieron.

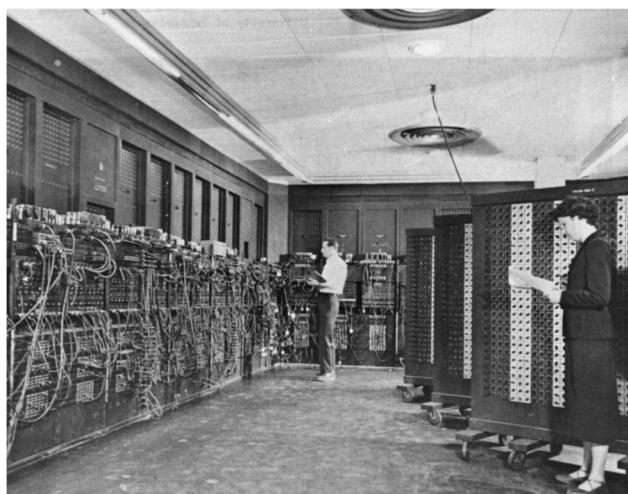
A partir de la Segunda Guerra Mundial la historia de los ordenadores se clasifica por lo que denominamos generaciones y se caracteriza por los rápidos avances experimentados en el ámbito de la electrónica. Podemos diferenciar las siguientes generaciones:

ordenador y por ello se considera a Ada Byron el primer programador de la historia. Existe un lenguaje de programación que lleva su nombre: **ADA**.

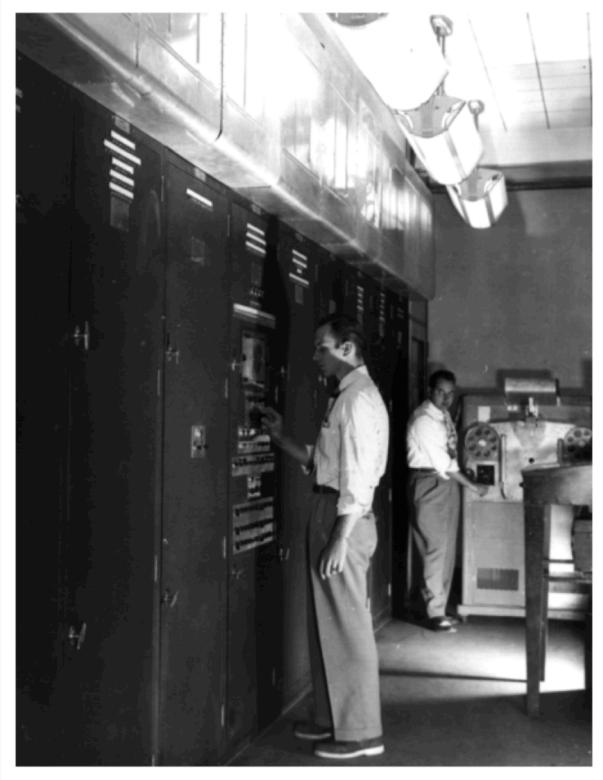
1^a Generación (1940 – 1956).

Comprende los primeros grandes ordenadores basados en la arquitectura von Neuman (que veremos más adelante) que surgen como una necesidad vital al considerárseles un instrumento armamentístico durante la gran guerra. Sus características principales son:

- Uso de tecnología basada en **válvulas de vacío** en lugar de los interruptores electromecánicos para dar soporte a los biestables.
- Empleo de computadoras con fines militares y científicos.
- Máquinas muy grandes, pesadas y muy lentas en sus cálculos (5000 cálculos por segundo).
- Destacan máquinas como el **ENIAC**⁶ (1946) o el **EDVAC**⁷ (1949). La primera aún trabaja con el sistema decimal, mientras que la segunda fue la primera computadora en adoptar el sistema binario entre otras muchas novedades que han perdurado hasta la arquitectura actual.



ENIAC (1946) ↑



EDVAC (1949) →

2^a Generación (1956 – 1963).

Esta etapa coincide con la aparición, en 1956, del **transistor**, componente electrónico mucho más pequeño y fiable, además de presentar un menor consumo, que las válvulas de vacío.

⁶ **Electronic Numerical Integrator And Computer.**

⁷ La **EDVAC** (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) por sus siglas en inglés, fue una de las primeras computadoras electrónicas. A diferencia de la ENIAC, no era decimal, sino binaria y tuvo el primer programa diseñado para ser almacenado. Este diseño se convirtió en el estándar de arquitectura para la mayoría de las computadoras modernas. El diseño de la EDVAC es considerado un éxito en la historia de la informática.

Sus principales características son:

- Máquinas basadas en transistores, mucho más pequeñas y con menos consumo.
- Aunque se utilizan sobre todo para cálculo científico aparecen las primeras computadoras con fines comerciales.
- Aparece la serie IBM 7090 que empieza a comercializarse para grandes empresas.
- Aparecen los primeros periféricos y el concepto de **supercomputadora**⁸ como ordenador con capacidades de cálculo muy por encima de los comunes en la época.
- Surgen los primeros lenguajes de programación y los sistemas **batch** o de procesamiento por lotes, que permitían la ejecución de un programa sin la supervisión directa del usuario (frente a los sistemas interactivos que necesitan el control del usuario).



Consola de la IBM 7090

3^a Generación (1964 – 1971).

El principal hecho que caracteriza esta etapa es la aparición de los **circuitos integrados**, es decir la integración en un solo chip de todas la unidades funcionales de la CPU. Esto traería consigo un gran cambio en cuanto al tamaño de las computadoras y su capacidad de proceso.

Las principales características que se dan en esta generación son:

- En un principio se usa una tecnología **SSI** o pequeña escala de integración, aunque pronto se pasa a **MSI** o escala de integración media, es decir, de los primeros circuitos que integraban decenas de transistores se evoluciona a circuitos con cientos de transistores en el mismo chip.
- Aparecen nuevos soportes de almacenamiento como los discos flexibles (creados por IBM) y nuevos dispositivos como el monitor.

⁸ Las supercomputadoras aparecieron en la década de los sesenta y fueron introducidas por la empresa CDC (Control Data Corporation) y diseñadas por Seymour Cray quien da nombre a muchas de estas máquinas.

- Aparecen también nuevas técnicas y lenguajes de programación.
- Se distingue entre los conceptos de miniordenador (computadora multiusuario de prestaciones intermedias), mainframe y estación de trabajo.
- Se emplean por primera vez lenguajes de alto nivel no específicos, o sea de propósito general, como el **C**, **Basic**, **Pascal**, etc.

A mediados de los 60 IBM lanzó el System 360, la primera arquitectura de ordenador que permitía intercambiar los programas y periféricos entre los distintos equipos, al contrario de lo que ocurría hasta entonces, cuando cada equipo era una caja cerrada incompatible con los demás. El proyecto casi le cuesta ir a la quiebra a IBM, pero una vez comercializado tuvo tanto éxito que se constituyó en líder indiscutible del mercado.

4^a Generación (1971 – 1981).

Se caracteriza por la aparición del microordenador y de la computadora personal o doméstica. Los avances en la electrónica permiten integrar un mayor número de circuitos en una sola pastilla. Esto reduce el espacio y el consumo haciendo más asequibles los ordenadores. Sus características principales son:

- Aparece la tecnología **LSI** (alta escala de integración) que empleaba miles de transistores en una sola pastilla.
- Aparece el **microprocesador**, entendido como aquel circuito integrado que contiene todos o algunos de los elementos hardware de la CPU.
- Proliferan los lenguajes de programación.
- Se democratiza el uso del ordenador con la aparición de modelos domésticos como el Amstrad CPC, el ZX Spectrum o los Commodore 64 o 128. En 1981 IBM presenta su primer PC.



Commodore 64



ZX Spectrum

5^a Generación (1982 – 1991):

El microprocesador sigue su evolución geométrica reduciendo el tamaño y el consumo e incorporando mayores funcionalidades. Sus características más importantes son:

- Tecnología **VLSI** de muy alta escala de integración (del orden de 100.000 transistores).
- Aparecen los microprocesadores de uso específico.
- Evolución muy rápida de la tecnología (lo que se vino en llamar **Ley de Moore**⁹).
- Desarrollo y expansión de la tecnología multimedia.
- Apple inventa la arquitectura abierta que permitirá añadir, modernizar y ampliar componentes y la interfaz gráfica de usuario o GUI que utiliza un conjunto de imágenes y elementos gráficos (**iconos**) para representar la información y las acciones y en la que se basan todos los sistemas de la actualidad.
- Las computadoras bajan sus precios, son usadas cada vez más y en más ámbitos.
- Aparecen microprocesadores que trabajan en paralelo y se extiende el uso generalizado de las redes.

6^a Generación (1992 - actualidad):

Se utilizan tecnologías superiores de integración com **ULSI** (Ultra Large Scale Integration, hasta 1M de transistores) y la **GLSI** (Giga Large Scale Integratión, más de un millón de transistores en la misma pastilla).

En la actualidad la fabricación de CPU's está basada en múltiples procesadores que trabajan al mismo tiempo de forma que algunas máquinas pueden llegar a realizar más de un billón de operaciones por segundo. Además se ha extendido la conectividad de computadoras mediante el empleo de redes, sobre todo Internet, y cada vez crece más el uso de aplicaciones soportadas por ésta.

Además nuevas tecnologías están apareciendo o se encuentra en pleno desarrollo como la Inteligencia artificial distribuida, el empleo de la teoría del caos, el uso de sistemas difusos¹⁰, la holografía para implementar memorias, transistores ópticos, nanotecnología, etc.

2.4. Evolución histórica desde un punto de vista funcional.

Desde la perspectiva funcional los ordenadores no han evolucionado tanto como desde un punto de vista físico o electrónico, ya que mayoritariamente sigue vigente el esquema de funcionamiento diseñado por **von Neumann**, con una serie de módulos funcionales comunes (dispositivos de entrada y salida, memoria principal y

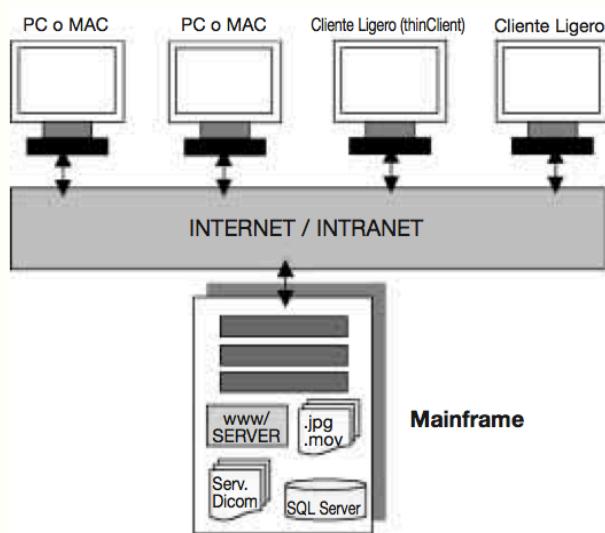
⁹ La **Ley de Moore** expresa que aproximadamente cada 18 meses se duplica el número de transistores en un circuito integrado.¹ Se trata de una ley empírica, formulada por el co-fundador de **Intel**, **Gordon E. Moore** el 19 de abril de 1965, cuyo cumplimiento se ha podido constatar hasta hoy.

¹⁰ En Inteligencia artificial, la **lógica difusa**, o lógica borrosa se utiliza para la resolución de una variedad de problemas, principalmente los relacionados con control de procesos industriales complejos y sistemas de decisión en general, la resolución la compresión de datos. Los sistemas de lógica difusa están también muy extendidos en la tecnología cotidiana, por ejemplo en cámaras digitales, sistemas de aire acondicionado, lavarropas, etc. Los sistemas basados en lógica difusa imitan la forma en que toman decisiones los humanos, con la ventaja de ser mucho más rápidos. Estos sistemas son generalmente robustos y tolerantes a imprecisiones y ruidos en los datos de entrada. Algunos lenguajes de programación lógica que han incorporado la lógica difusa serían por ejemplo las diversas implementaciones de **Fuzzy PROLOG** o el **lenguaje Fril**.

secundaria, microprocesador y buses). Donde si se han realizado grandes cambios en la forma de comunicarse y operar entre sí.

El Sistema Informático ha evolucionado desde una primera situación en la que todos los elementos del mismo se encontraban centralizados en un mismo lugar (en un solo chasis en un sistema monopuesto o a través de salas de ordenadores en un sistema multipuesto), encontrándonos con sistemas aislados, hasta la situación actual en la que los componentes del sistema pueden encontrarse repartidos en diferentes lugares físicos dando lugar a sistemas conectados en red que pueden llegar a colaborar entre sí dando lugar a los llamados **sistemas distribuidos**.

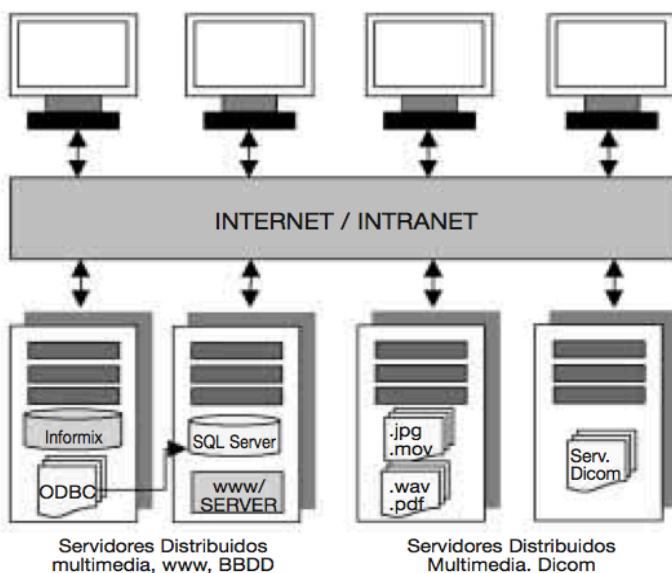
Figura 1. Sistema centralizado



Los sistemas actuales distribuidos se pueden organizar de forma vertical u horizontal.

- **Distribución vertical:** Existen varios niveles, como el primer nivel o corporativo, segundo nivel o departamental y tercer nivel o puesto personal, y en cada uno se usan distintos tipos de equipos con configuraciones características.
- **Distribución horizontal:** Todos los equipos tienen la misma categoría, por lo menos no existe un equipo central en el primer nivel de la jerarquía. Suelen existir un conjunto de ordenadores conectados que cooperan entre sí pero sin que ninguno de ellos centralice la información.

Figura 2. Sistema Distribuido



3. Representación de la Información.

3.1. Introducción.

Como hemos visto el ordenador procesa datos/información, a saber:

- Instrucciones que forman el programa (indica qué debe hacerse).
- Los datos con los que debe trabajar Instrucciones y datos deben almacenarse dentro del ordenador.

La información internamente se representará mediante un alfabeto, utilizando un código o representación interna. Físicamente la información se presentará mediante una combinación de bits (que viene de la expresión inglesa binary digit, y que a la hora de la implementación física se convertirá en dos niveles de tensión diferenciados).

Externamente, manejamos información. Utilizamos un lenguaje, también llamado código, que se construye sobre un alfabeto. Si queremos comunicarnos con el ordenador, debemos usar un alfabeto que después se pueda representar dentro del mismo, formado por:

- Caracteres Alfabéticos: Mayúsculas y minúsculas del alfabeto inglés.
- Caracteres Numéricos: Del cero al nueve.
- Caracteres Especiales: {}, #, \$, %, &, -, +, *, /, (,), ?, !, ?, [,]
- Caracteres de Control: Representan órdenes de control al ordenador: EOL, EOT, SYNC, ESC, BEEP, CTRL.
- Caracteres Gráficos Permiten representar figuras o iconos elementales.

Generalmente nos referiremos a estas clases como:

- Caracteres alfanuméricos: que abarcan las dos primeras.
- Caracteres de texto: que abarcan las tres primeras categorías.

Es necesaria, por tanto, una codificación externa mediante códigos de Entrada/Salida normalizados, que se puedan traducir de la notación externa a la representación binaria correspondiente.

$$\{a,b,...,z,A,...,Z,0,...,9,(,...,]\} \rightarrow \{0,1\}^n$$

$$\alpha \rightarrow \beta$$

Donde β está codificado mediante n bits. Los códigos de Entrada/Salida, aunque normalizados, pueden variar en función del fabricante.

Las operaciones que normalmente realizamos con los símbolos tienen su equivalente en las operaciones sobre el alfabeto binario. Para facilitar la traducción entre los códigos externos (que nosotros entendemos) y los internos (binario, difícil de comprender para el ser humano) existen lo que se denominan códigos intermedios, como pueden ser el **código octal** (base 8) o el **hexadecimal** (base 16).

3.2. Sistemas de numeración.

3.2.1. Sistemas de representación más usuales.

Los sistemas de numeración pueden ser aditivos o posicionales. Los Sistemas de numeración posicionales son los más usuales y son los que vamos a tratar. Un sistema de numeración en **base b** utiliza b símbolos de un alfabeto y el valor de cada número dependerá de la posición que ocupe cada símbolo. Por lo tanto, un número no será más que una secuencia de cifras (elegidas entre los b símbolos posibles).

Sistema de Numeración Decimal:

Símbolos permitidos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Ejemplo:

$$245,63 = 2 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 + 6 \cdot 10^{-1} + 3 \cdot 10^{-2}$$

Si generalizamos esto:

Un número: $N \equiv \dots n_4 n_3 n_2 n_1 n_0 n_{-1} n_{-2} \dots$

Su valor: $N \equiv \dots + n_4 \cdot b^4 + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + \dots$

Recordad que $b^{-n} = 1/b^n$, por lo que $x \cdot b^{-n} = x/b^n$.

Sistema de Numeración Octal:

Símbolos permitidos: 0, 1, 2, 3, 4, 5, 6, 7 Base: 8

$$175,37_8 = 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 + 3 \cdot 8^{-1} + 7 \cdot 8^{-2} = 125,484375_{10}$$

Representación en base dos:

- En el sistema de Numeración en Base Dos o Binario $b = 2$, y sólo se usan dos elementos para representar cualquier número: 0 y 1.
- Estos elementos del alfabeto se denominan cifras binarias o bits.

Ejemplo: Tabla de números en binario del 0 al 7 (con tres bits).

Código binario:	000	001	010	011	100	101	110	111
Equivalente decimal:	0	1	2	3	4	5	6	7

3.3 Transformaciones entre bases binaria y decimal.

De base 2 a base 10.

La base es 2. Simplemente hay que sumar los pesos de las posiciones en las que hay un uno.

Aprovechando la expresión vista antes de N:

$$110100_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^2 = 52_{10}$$

$$0,10100_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-3} = 0,625_{10}$$

$$10100,001_2 = 2^4 + 2^2 + 2^{-3} = 20,125_{10}$$

De base 10 a base 2.

a) Parte entera:

El nuevo número estará formado por los restos de las sucesivas divisiones por dos y el último cociente, teniendo en cuenta que éste será el **MSB** (Most Significative Bit) y el primer resto será el **LSB** (Lesser Significative Bit).

Ejemplo: $26_{10} = 11010_2$

b) Parte fraccionaria:

Se obtiene al multiplicar sucesivamente por dos la parte fraccionaria. El nuevo número se forma con las partes enteras que serán siempre 0 ó 1.

Ejemplos:

$$0,1875_{10} = 0,0011_2$$

$$74,625_{10} = 1001010,101_2$$

Ambos códigos no han de corresponderse en longitud, lo cual puede llevarnos a un error de truncamiento cuando tengamos un número fijo de posiciones y no alcancemos un número exacto y nos veamos obligados a detenernos.

3.4. Operaciones aritméticas con variables binarias.

Una variable binaria puede contener un número binario.

a	b	a + b	a - b	a * b	a / b
0	0	0	0	0	indeterminado
0	1	1	1_{-1}	0	0
1	0	1	1	0	∞
1	1	0_{+1}	0	1	1

Ejemplos:

$$\begin{array}{r}
 1110101 \\
 + 1110110 \\
 \hline
 11101011
 \end{array}
 \quad
 \begin{array}{r}
 1101010 \\
 - 1010111 \\
 \hline
 0010011
 \end{array}$$

$$\begin{array}{r}
 1101010 \\
 \times 11 \\
 \hline
 1101010 \\
 1101010 \\
 \hline
 100111110
 \end{array}
 \quad
 \begin{array}{r}
 1101010 \\
 \times 101 \\
 \hline
 1101010 \\
 1101010 \\
 \hline
 1000010010
 \end{array}$$

$$1101 \div 11 = 100 \text{ resto } 001$$

En resumen:

- Multiplicar por dos es añadir un cero a la derecha o mover el punto decimal a la derecha.
- Dividir por dos es mover el punto decimal a la izquierda o eliminar un cero a la derecha.

Ejemplos:

$$10101,0101_2 \times 10_2 = 101010,101_2$$

$$1101,1010_2 \times 2^5_{10} = 1101,1010_2 \times 100000_2 = 110110100_2$$

$$1010100_2 \div 2_{10} = 1010100_2 \div 10_2 = 1010100_2 \times 0,1_2 = 101010,0_2$$

$$10101,101_2 \div 2^6_2 = 10101,101_2 \times 0,000001_2 = 0,010101101_2$$

Vista la dificultad de la operación resta, que afecta tanto a la resta como a la división, se busca una representación que permita realizar las restas como sumas: la notación en complementos.

3.5. Notación y operaciones en complementos.

- Permite representar los números negativos.
- Facilita las operaciones de resta.

Cuando se opera con complementos siempre hay que igualar ambos operandos en número de bits, añadiendo ceros a la izquierda en el operando que los necesite, antes de hallar el complemento.

3.5.1. Complemento a la base menos uno de un número.

\bar{N} , el complemento a la base menos uno de un número N, se consigue restando cada una de las cifras del número, de la base menos uno: $\bar{N} = (b^n - 1 - N)$. Por ejemplo, en la base 10, la base menos uno es 9:

$$\begin{array}{r} 99 \\ \overline{25} = \frac{-25}{74} \end{array}$$

Complemento a 9, para base 10:

$$\begin{array}{r} 99 \\ \overline{63} = \frac{-63}{36} \end{array} \quad \begin{array}{r} 999 \\ \overline{162} = \frac{-162}{837} \end{array}$$

Complemento a 1, para base 2:

En la representación en binario natural, base $b = 2$, se habla de **complemento a 1**:

$$\begin{array}{r} 11111 \\ \overline{10010} = \frac{-10010}{01101} \end{array} \quad \begin{array}{r} 111111 \\ \overline{101010} = \frac{-101010}{010101} \end{array}$$

Para obtener el complemento a 1 de un número N en base 2, sólo es necesario cambiar los ceros por unos y viceversa.

Resta en complemento a la base menos uno:

La operación resta se consigue sumando al minuendo el complemento a la base menos uno del sustraendo. Si existe un acarreo, se elimina y se suma al resultado.

Ejemplos:

- Complemento a 9.

$$\begin{aligned} 77 - 63 &= 77 + \overline{63} = 77 + 36 = 113 \Rightarrow 13 + 1 = 14 \\ 1100 - 16 &= 1100 + \overline{0016} = 1110 + (9999 - 0016) = 1100+9983 = 11083 \Rightarrow \\ &1083+\cancel{1} = 1084 \end{aligned}$$

- Complemento a 1.

$$\begin{aligned} 1000111 - 10010 &= 1000111 + \overline{0010010} = 1000111 + (1111111 - 0010010) = \\ 1000111 + 1101101 &= 10110100 \Rightarrow 0110100 + \cancel{1} = \textcolor{green}{0110101} \end{aligned}$$

- Como comprobación:

$$1000111 - 0010010 = 0110101$$

3.5.2. Complemento a la base de un número.

\bar{N} , el complemento a la base de un número, N, se consigue restando cada una de las cifras del número, de la base menos uno, y sumando uno al resultado:

$$\bar{N} = b^n - N = b^n - N - 1 + 1 = (b^n - 1 - N) + 1$$

Esto es, se consigue calculando el complemento a la base menos uno, y sumando uno al resultado.

De otra forma: el complemento a la base de un número N de n dígitos enteros es igual a $b^n - N$, o lo que es lo mismo, un 1 seguido de tantos ceros como dígitos tenga el numero N (en definitiva da igual donde se sume el 1, si al principio o al final). Ejemplo para el complementario de 25:

$$99 - 25 = 74 + 1 = 75 \rightarrow 99 + 1 = 100 - 25 = 75$$

Por ejemplo, en la base 10:

$$\begin{array}{r} 99 \\ \overline{25} = \underline{-25} \\ \hline 74 \\ + 1 \\ \hline 75 \end{array}$$

Complemento a 10:

$$\begin{array}{r} 99 \\ \overline{63} = \underline{-63} \\ \hline 36 \\ + 1 \\ \hline 37 \end{array} \quad \begin{array}{r} 999 \\ \overline{162} = \underline{-162} \\ \hline 837 \\ + 1 \\ \hline 838 \end{array}$$

Complemento a 2 :

En la representación en binario natural, base $b = 2$, se habla de **complemento a 2**:

$$\begin{array}{r} 11111 \\ \overline{10010} = \underline{-10010} \\ \hline 01101 \\ + 1 \\ \hline 01110 \end{array} \quad \begin{array}{r} 111111 \\ \overline{101010} = \underline{-101010} \\ \hline 010101 \\ + 1 \\ \hline 010110 \end{array}$$

Para obtener el complemento a 2 de un número N, sólo es necesario calcular el complemento a 1 y sumar uno al resultado; esto es, cambiar los ceros por unos y viceversa, y sumar uno al resultado.

Resta en complemento a la base:

La operación resta se consigue sumando al minuendo el complemento a la base del sustraendo. Si existe un acarreo, se elimina.

Ejemplos:

- Complemento a 10.

$$\begin{aligned} 77 - 63 &= 77 + \overline{63} = 77 + ((99 - 63) + 1) = 77 + (36 + 1) = 77 + 37 = 114 \Rightarrow 14 \\ 1100 - 16 &= 1100 + \overline{0016} = 1110 + ((9999 - 0016) + 1) = 1100 + (9983 + 1) = 1100 + 9984 = 11084 \Rightarrow 1084 \end{aligned}$$

- Complemento a 2.

$$\begin{aligned} 1000111 - 10010 &= 1000111 + \overline{0010010} = 1000111 + ((1111111 - 0010010) + 1) \\ &= 1000111 + (1101101 + 1) = 1000111 + 1101110 = 10110101 \Rightarrow 110101 \end{aligned}$$

Sabiendo que es sencillo realizar sumas dentro de un ordenador y que los ordenadores son capaces de realizar estas operaciones de forma veloz, se consigue:

- Realizar las restas como una inversión (cambiar ceros por unos y viceversa) y una suma.
- Realizar las multiplicaciones como desplazamientos y sumas.

3.6. Códigos intermedios de representación.

Habíamos visto anteriormente que existía un código de E/S similar al que utilizan los seres humanos, α , y un código interno, β , derivado del binario que utilizan los ordenadores.

$$\begin{aligned} \{a, b, \dots, z, A, \dots, Z, 0, \dots, 9, (\dots,]\} &\rightarrow \{0, 1\}^n \\ \alpha &\rightarrow \beta \end{aligned}$$

A mitad de camino entre α y β están los códigos intermedios, que son más amigables para el ser humano que la representación interna, pero que están más próximos a ésta que los códigos de E/S. Generalmente los códigos de E/S se presentan mediante algún código intermedio.

Es inmediato pasar de los códigos intermedios a β , que será algún tipo de representación binaria. Las representaciones intermedias utilizan bases potencias de 2: 8 ó 16.

¿ Cuántos bits son necesarios para representar los símbolos de un alfabeto ?

$$\begin{aligned} \alpha &\rightarrow \beta \\ \{0, \dots, 9, a, \dots, z, A, \dots, Z, ;, :, ?, \dots\} &\rightarrow \{0, 1\}^n \end{aligned}$$

Para cada alfabeto tenemos: $|\alpha| = m$

Sin embargo, para un n dado, tenemos que $| \beta | = 2^n$

En general, $m \neq 2^n$

Necesitamos representar todos los caracteres, m , por lo tanto hay que exigir que:

$$m \leq 2^n$$

Buscaremos el menor n entero que verifique que:

$$n \geq \lg_2 m = 3,32 \log m$$

Por ejemplo, ¿ cuántos bits necesitamos para representar el código octal?

$$n \geq \lg_2 8 = 3$$

¿ Y el código hexadecimal?

$$n \geq \lg_2 16 = 4$$

En general, para un código con 10 símbolos:

$$n \geq \lg_2 10 = 3,32 \log 10 = 3,32$$

El menor entero que lo verifica es $n = 4$. Obviamente, si $2^n > m$, hay combinaciones de bits que no se usan.

3.6.1. Códigos intermedios: octal y hexadecimal.

Código octal.

¿ Cuántos bits son necesarios para representar 8 caracteres distintos? La respuesta es 3.

$$b = 8 = 2^3$$

$$B = \{0,1,2,3,4,5,6,7\}$$

En la siguiente tabla podemos ver la correspondencia entre los sistemas decimal, octal y binario para los dígitos del sistema octal:

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

Ejemplos:

- 71_8 es un número octal válido = 57_{10}
- 87_8 no es un número octal válido

Código Hexadecimal:

¿ Cuántos bits son necesarios para representar 16 caracteres distintos ?

$$b = 16 = 2^4, B = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$$

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

La respuesta es 4. En la siguiente tabla podemos ver la correspondencia entre los sistemas decimal, hexadecimal y binario para los dígitos del sistema hexadecimal:

Ejemplos:

- $1A_H = 1 \cdot 16^1 + 10 \cdot 16^0 = 26_{10}$, es un número válido
- $F3_H = 15 \cdot 16^1 + 3 \cdot 16^0 = 243_{10}$, es un número válido
- $G2_H$, no es un número válido ya que $G \notin B$

3.6.2. Conversiones entre bases.

Conversión de octal a decimal:

Aplicamos:

$$N = \sum_{i=-n}^{n} n_i \cdot b^i, \text{ con } b=8$$

Ejemplos:

$$55_8 = 5 \cdot 8^1 + 5 \cdot 8^0 = 45_{10}$$

$$0,42_8 = 4 \cdot 8^{-1} + 2 \cdot 8^{-2} = 0,53125_{10}, \text{ o de forma equivalente:}$$

$$0,42_8 = 0,100010_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-5} = 0,53125_{10}$$

Conversión de decimal a octal:

Mediante sucesivas divisiones por ocho para la parte entera y sucesivas multiplicaciones por ocho de la parte decimal.

Ejemplo:

$$2001,2910_{10} = 3721,2247_8$$

Conversión de binario a octal:

Existen dos formas de realizarla:

- ### I. Binario \Rightarrow Decimal \Rightarrow Octal

- De binario a Decimal $\Rightarrow \sum n_i b^i$
 - Divisiones por ocho de la parte entera
 - Multiplicaciones por ocho de la parte fraccionaria

Ejemplo:

$$101010_2 = 2^5 + 2^3 + 2^1 = 42_{10} = 52_8$$

- II. Agrupar los números binarios en bloques de 3, tanto a un lado como al otro del punto decimal (recordad que $8 = 2^3$).

Ejemplo:

$$101\ 010 = 52_{10} = 5 \times 8 + 2 \times 8^0 = 42_{10}$$

Conversión de octal a binario:

La manera más fácil de hacerlo es usando el inverso al método 2 del apartado anterior: desagrupar cada cifra en octal por su equivalente, con tres cifras, en binario.

Ejemplos:

$$537,24_8 = 101011111,010100_2$$

$$175,22_8 = 001111101,010010_2$$

Conversión de hexadecimal a decimal:

Aplicamos:

$$N = \sum_{i=-m}^n n_i \cdot b^i, \text{ con } b=16$$

Ejemplos:

$$\begin{aligned}ABC_H &= A \cdot 16^2 + B \cdot 16^1 + C \cdot 16^0 = 2748_{10} \\11,AB_H &= 1 \cdot 16^1 + 1 \cdot 16^0 + A \cdot 16^{-1} + B \cdot 16^{-2} = ? \\1AB,2CDH &= ?\end{aligned}$$

Conversión de decimal a hexadecimal:

Mediante sucesivas divisiones por dieciséis para la parte entera y sucesivas multiplicaciones por dieciséis de la parte decimal.

Ejemplo:

$$2001,2910_{10} = ?_{16}$$

Conversión de binario a hexadecimal:

Existen dos formas de realizarla:

- Binario \Rightarrow Decimal \Rightarrow Hexadecimal:
- De binario a Decimal $\Rightarrow \sum n \cdot b^i$
- Divisiones por 16 de la parte entera
- Multiplicaciones por 16 de la parte fraccionaria

Ejemplo:

$$10101010_2 = 2^7 + 2^5 + 2^3 + 2^1 = 170_{10} = AA_H$$

- Dado que $16 = 2^4 \Rightarrow$ Agrupar los números binarios en bloques de 4, tanto a un lado como al otro del punto decimal

Ejemplo:

$$10101010_2 = AA_H = A \times 16^1 + A \times 16^0 = 10 \times 16^1 + 10 \times 16^0 = 170_{10}$$

Conversión de hexadecimal a binario:

Es el caso inverso al método 2 del apartado anterior: desagrupar cada cifra en hexadecimal por su equivalente binario (cada cuatro bits).

Ejemplos:

$$25DF_H = ?$$

$$1ABC.C4_H = ?$$

Pregunta: Teniendo en cuenta todos estos conceptos, ¿cuál sería el método más rápido para pasar el número 16275 a binario natural?

3.7. Códigos de entrada/salida.

Se usan códigos normalizados entre los que cabe destacar:

3.7.1. BCD (Binario Codificado en Decimal).

Usa 6 bits ($n = 6$) por lo que se pueden codificar hasta 64 caracteres (2^6). Puede llevar un séptimo bit como bit de verificación.

La disposición de los bits es la siguiente:

$b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$

Donde:

- b_6 es el bit opcional de verificación.
- b_5 y b_4 son los llamados bits de zona (ejemplo: 00 para los números)
- b_3, b_2, b_1, b_0 son los bits de posición, que coinciden con los números 1 al 9 en binario natural y el 0 como 10. Con este código no se pueden representar las letras minúsculas.

Ejemplo:

Carácter	BCD con 7 bits	Representación interna
'0'	112_o	1 0 0 1 0 1 0
'9'	111_o	1 0 0 1 0 0 1
'A'	061_o	0 1 1 0 0 0 1
'('	034_o	0 0 1 1 1 0 0

3.7.2. EBCDIC (Código de Intercambio Normalizado BCD Extendido).

Es la representación interna típicamente utilizada por los sistemas IBM. Usa 8 bits para cada codificación ($n = 8$), por lo tanto podemos codificar 256 caracteres que se disponen de la siguiente forma:

$b_0 \ b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7$

Donde:

- b_0 y b_1 marcan los distintos bloques del código:
 - 00: caracteres no usados y de control
 - 01: caracteres especiales
 - 10: minúsculas
 - 11: cifras y mayúsculas
- $b_2 \dots b_7$, son los bits de posición. Por ejemplo, dentro de estos, los bits b_2 y b_3 para $b_0 \text{ y } b_1 = 11$, serían:
 - 00: A - I
 - 01: J - R
 - 10: S - Z
 - 11: Números

Ejemplo:

Carácter	EBCDIC	Representación interna
'0'	$F0_h$	1 1 1 1 0 0 0 0
'9'	$F9_h$	1 1 1 1 1 0 0 1
'A'	$C1_h$	1 1 0 0 0 0 0 1
'('	$4D_h$	0 1 0 0 1 1 0 1

3.7.3. ASCII: American Standard Code for Information Interchange.

Código Normalizado Americano para Intercambio de Información. Es el más usado en los ordenadores personales y al que suelen hacer referencia casi todos los manuales de programación. Originalmente usa 7 bits ($n = 7$) y opcionalmente un bit adicional de verificación. Por lo tanto, puede representar hasta $m = 128$ caracteres distintos.

Distribución de los bits:

$b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$

Ejemplo:

Carácter	ASCII 7 bits	Representación interna
'0'	060 ₁₀	0 1 1 0 0 0 0
'9'	071 ₁₀	0 1 1 1 0 0 1
'A'	101 ₁₀	1 0 0 0 0 0 1
'('	050 ₁₀	0 1 0 1 0 0 0

3.7.4. Detección de errores en la información codificada.

¿Qué ocurre cuando $2^n > m$?

La **eficiencia** (τ) de un código es :

$$\tau = \frac{m}{2^n} = \frac{\text{los que se representan}}{\text{los que pueden representarse}}$$

Un código es más eficiente cuantas menos combinaciones de bits desperdicie.

Ejemplo: Código ASCII, 95 símbolos:

- Sin bit de verificación, $n= 7$; $\tau = \frac{m}{2^n} = \frac{95}{2^7} = 0,742$
- Con bit de verificación, $n=8$: $\tau = \frac{m}{2^n} = \frac{95}{2^8} = 0,371$

La **redundancia** (R) de un código viene determinada por la fórmula:

$$R = (1 - \tau) \times 100 \%$$

La redundancia está inversamente relacionada con la eficiencia. Un código poco eficiente será redundante y viceversa.

Ejemplo: Código ASCII, 95 símbolos:

- Sin bit de verificación, $n = 7$: $R = 25'8 \%$
- Con bit de verificación, $n = 8$: $R = 62'9 \%$

Bits de verificación:

Un código redundante puede utilizarse para detectar errores al transmitir información. ¿Qué ocurre cuando transmitimos sobre un alfabeto de 8 símbolos y sólo utilizamos $n = 3$ bits? ¿Qué ocurre cuando transmitimos sobre un alfabeto de 10 símbolos y sólo utilizamos $n = 4$ bits?

La adición de bits redundantes (*bits de verificación*) mediante algún algoritmo previamente fijado permite encontrar errores en la transmisión, si dicho algoritmo se aplica en ambos extremos de la comunicación.

Es posible aplicar distintos algoritmos:

- **Bit de paridad**

Añade al comienzo de cada secuencia un cero o un uno según alguno de los criterios siguientes:

- Paridad par**: el bit (cero o uno) hace que el número de unos sea par.
- Paridad impar**: el bit (cero o uno) hace que el número de unos sea impar.

En el otro extremo del canal de comunicación, si se produce un error en la información se detectará ya que fallará la paridad.

Ejemplo: BCD utilizando criterios de paridad par e impar.

Mensaje	Enviado con Paridad Par	Enviado con Paridad Impar
1 000 001	01 000 001	11 000 001
1 011 011	11 011 011	01 011 011
1 010 000	01 010 000	11 010 000
1 101 000	11 101 000	01 101 000

¿Qué ocurre si falla un número par de bits? Existen métodos basados en bit de paridad mucho más elaborados.

- **Verificación en cuenta fija.**

Cada símbolo o carácter enviado tiene un número fijo de ceros y unos. Con ocho bits podemos exigir que haya cuatro ceros y cuatro unos. Para detectar el error se cuenta el número de ceros y de unos y debe coincidir.

Ejemplo: Alfabeto = {A,B,C,D,E,F}, $n = 4$ y verificación en cuenta fija (2 ceros y 2 unos).

Símbolo	Código	Símbolo	Código
'A'	0011	'B'	0101
'C'	1001	'D'	0110
'E'	1010	'F'	1100

3.8. Representación interna.

Hemos visto los códigos de E/S, pero esta representación no es eficiente dentro del ordenador. Hay que realizar una traducción del código E/S a la representación interna. Internamente se utiliza alguna variante del binario natural, que es mucho más compacta.

¿ Cómo se almacena internamente la información? La información se agrupa internamente (en la ALU y en la CPU) en función de la longitud de palabra que procese el ordenador. El tamaño del código de representación interna será un múltiplo de la longitud de palabra, que suele ser de 8, 16, 32 ó 64 bits. Los distintos tipos de datos que se pueden almacenar en un ordenador tendrán distintas representaciones. El objetivo es agilizar los cálculos, ya que si se almacenaran directamente mediante su código de E/S se perdería la representación posicional de los números.

3.8.1. Lógicos, caracteres y complejos.

En los lenguajes de programación suele haber distintos tipos de datos básicos: enteros, reales, lógicos, caracteres y complejos. Estos tres últimos suelen representarse en función de los dos primeros.

Lógicos:

Representarán el valor cierto (.T.) o falso (.F.), o bien una variable booleana (0 ó 1); suele representarse como un caso especial de un entero: falso = 0 y cierto = 1.

Caracteres:

Se representa el código de E/S (p.e. ASCII) directamente sobre palabras del ordenador; un conjunto de caracteres ocupará una colección de palabras consecutivas en la memoria. Se puede representar más de un carácter por palabra.

Complejos de simple y de doble precisión:

Es el caso más sencillo de datos estructurados; internamente no se representa la i, sino que se almacenan como dos números reales de simple o doble precisión consecutivos; el procesador, en función del tipo, sabrá cómo interpretar esa información.

3.8.2. Representación interna de los datos de tipo entero.

Fundamentos:

- se dispone de n posiciones, para el signo y el valor, en función de la longitud de palabra,
- la notación es $a_{n-1}, a_{n-2}, \dots, a_1, a_0$, siendo a_{n-1} el bit más significativo (**msb**) y el bit a_0 menos significativo (**lsb**).

Existen dos alternativas de representación:

- **Enteros sin signo**, es decir, disponemos de los n bits para representar el valor absoluto:

Ejemplo: $n = 8$

$$00000000_2 = 0_{10}$$

$$00000001_2 = 1_{10}$$

...

$$11111110_2 = 254_{10}$$

$$11111111_2 = 255_{10}$$

Se da que $0 \leq N \leq 2^n - 1$

- **Enteros con signo**, El significado de los bits varía: $a_{n-1}, a_{n-2}, \dots, a_1, a_0$ en función de la representación en:

- **Signo y Magnitud**: a_{n-1} indica el signo, si es 1 entonces $N < 0$, si es 0, $N \geq 0$.

Tanto si a_{n-1} es 0, como si es 1, a_{n-2}, \dots, a_1, a_0 representan el valor absoluto del número en binario natural. El rango de esta representación es $-2^{n-1} + 1 \leq N \leq 2^{n-1} - 1$. Además se puede representar ± 0 (lo que supone un inconveniente).

- **Complemento a 1**: a_{n-1} indica el signo, si es 0, $N \geq 0$, y a_{n-2}, \dots, a_1, a_0 representan el valor absoluto del número en binario natural; si es 1 entonces $N \leq 0$, a_{n-2}, \dots, a_1, a_0 representan al número en complemento a 1.

El rango de representación es $-2^{n-1} + 1 \leq N \leq 2^{n-1} - 1$ y además se puede representar ± 0 (inconveniente).

- **Complemento a 2**, Idem, pero si es 1 entonces $N < 0$, a_{n-2}, \dots, a_1, a_0 representan al número en complemento a 2. El rango de representación es $-2^{n-1} \leq N \leq 2^{n-1} - 1$. Además tiene una única representación para el 0.

- **Notación sesgada**: el número que se representa siempre es positivo; pasamos de una representación $-2^{n-1} \leq N \leq 2^{n-1} - 1$ a otra $0 \leq M \leq 2^{n-1} - 1$, en la que los números negativos van antes que el cero o los números positivos en binario natural. Esta notación es muy ventajosa para comparar dos números, sin tener en cuenta su signo.

Para n bits, se establece el sesgo (S), de forma genérica, como:

$$S = 2^{n-1}$$

Internamente se representa:

$$M = N + S$$

Rango:

$$\begin{aligned} 0 \leq M \leq 2^n - 1 \\ -2^{n-1} \leq N \leq 2^{n-1} - 1 \end{aligned}$$

Ejemplo: $n = 4$ bits,

$$\begin{aligned} N_1 &= +7_{10} \quad M_1 = N_1 + S = 7 + 24 - 1 = 15_{10} = 1111_S \\ N_2 &= -6_{10} \quad M_2 = N_2 + S = -6 + 8 = 2_{10} = 0010_S \end{aligned}$$

Resumen:

Veamos un ejemplo de todas las notaciones que se han visto, para $n = 4$:

S. y M.	C. a 1	C. a 2	Sesgada
0111 = +7	0111 = +7	0111 = +7	1111 = +7
0110 = +6	0110 = +6	0110 = +6	1110 = +6
0101 = +5	0101 = +5	0101 = +5	1101 = +5
0100 = +4	0100 = +4	0100 = +4	1100 = +4
0011 = +3	0011 = +3	0011 = +3	1011 = +3
0010 = +2	0010 = +2	0010 = +2	1010 = +2
0001 = +1	0001 = +1	0001 = +1	1001 = +1
0000 = +0	0000 = +0	0000 = +0	1000 = +0
1000 = -0	1111 = -0	1111 = -1	0111 = -1
1001 = -1	1110 = -1	1110 = -2	0110 = -2
1010 = -2	1101 = -2	1101 = -3	0101 = -3
1011 = -3	1100 = -3	1100 = -4	0100 = -4
1100 = -4	1011 = -4	1011 = -5	0011 = -5
1101 = -5	1010 = -5	1010 = -6	0010 = -6
1110 = -6	1001 = -6	1001 = -7	0001 = -7
1111 = -7	1000 = -7	1000 = -8	0000 = -8

Ejercicio: Represéntense en las cuatro notaciones los números -1453 y 1998, sabiendo que se dispone de una precisión de 2 bytes.

La representación más utilizada es el complemento a dos, dada su facilidad para realizar las restas.

N_{min} y N_{max} para distintas longitudes de palabra:

Precisión	N_{max} $2^{n-1} - 1$	N_{min} Comp. a 1: $-2^{n-1} + 1$	Comp. a 2 -2^{n-1}
8 bits	127	-127	128
16 bits	32767	-32767	32768
32 bits	2147483649	-2147483649	2147483650
64 bits	$9,223372 \cdot 10^{18}$	$-9,223372 \cdot 10^{18}$	$9,223372 \cdot 10^{18}$

Existen otras formas de representar enteros como la Representación de enteros en BCD.

Representación de enteros en BCD:

- **BCD empaquetada:** dos dígitos decimales por byte.
- **BCD desempaquetada:** un dígito decimal por byte.

En este tipo de representaciones se suele utilizar un byte para el signo (0000 para números positivos y 0001 para números negativos).

3.8.3. Los números reales: la notación IEEE 754.

Al representar números reales nos podemos encontrar con los siguientes problemas de:

- **precisión:** ¿ Cómo representar π ?
- **magnitud:** ¿ Cómo representar los números reales (especialmente aquellos que son muy grandes o muy pequeños) ?

Por ejemplo: $C = 2,9979 \cdot 10^8 \text{ m/s}$
 $\text{Carga fundamental} = 1,602 \cdot 10^{-19} \text{ C}$

En notación exponencial, científica o en coma flotante cualquier número se puede representar de la forma:

$$N = M \cdot B^E$$

...donde **N** es el número, **M** es la mantisa, **B** es la base y **E** es el exponente. Esta representación puede modificarse, conservando el valor de N, si se reajustan adecuadamente M y E.

$$13257,3285 = 13257,3285 \cdot 10^0 = 1,32573285 \cdot 10^4 = 0,132573285 \cdot 10^5 = 132573285 \cdot 10^{-4}$$

Como hemos visto en el ejemplo anterior:

- Aumentar en una unidad el valor de E supone dividir M por B:

$$123,456 = 123,456 \cdot 10^0 = 12,3456 \cdot 10^1 = 1,23456 \cdot 10^2$$

- Disminuir en una unidad el valor de E supone multiplicar M por B:

$$123,456 = 123,456 \cdot 10^0 = 1234,56 \cdot 10^{-1} = 12345,6 \cdot 10^{-2}$$

Estas tareas, dentro de un ordenador, ya suelen venir integradas con los nuevos procesadores. Antes, había que realizarla con los co-procesadores aritméticos y si no se disponía de co-procesador, era necesario realizarla mediante software (también puede ser necesario realizar estas operaciones mediante un programa si queremos disponer de una precisión mayor de la que realmente tiene nuestro ordenador).

Al principio cada fabricante tenía su propia notación. Entre 1977-1985 se crea la norma IEEE-754, actualmente aceptada. Esto facilita el desarrollo de software aritmético genérico.

Representación interna. Notación IEEE-754:

- **E** ha de ser entero,
- **B = 2** (se puede omitir),
- Sólo es necesario almacenar **M** y **E** con sus respectivos signos.

s	e	m
---	---	---

$$\begin{aligned} |s| &= 1 \\ |e| &= n_e \\ |m| &= n_m \\ n &= 1 + n_e + n_m \end{aligned}$$

Con esta representación se pueden utilizar algoritmos de comparación de enteros para comparar números reales.

- Signo **s** = 0 para los positivos y **s** = 1 para los negativos.
- Exponente:
 - Entero en notación sesgada, con $S = 2^{n_e-1} - 1$
 - Si $N = M \cdot B^E \Rightarrow e = E + S$
Ejemplo: $n_e = 8 \Rightarrow S = 2^{8-1} - 1 = 127$

La Mantisa debe estar Normalizada y Empaquetada:

- **Normalizada:** Una mantisa está normalizada cuando su bit = 1 más significativo se encuentra en la posición de las unidades. En caso contrario se dice que la mantisa está “denormalizada”.
- **Empaquetada:** Se dice que la mantisa está empaquetada cuando sólo se almacena la parte fraccionaria del número normalizado.

La mantisa normalizada tendrá la forma $M = 1.m$. Si la mantisa está empaquetada sólo se almacenará m .

Se evita perder bits significativos cuando se realicen varias operaciones consecutivas. Además, siempre se tendrán n_m bits significativos.

La unidad aritmético lógica (ALU) tiene que desempaquetar de la representación interna antes de realizar las operaciones. También tiene que empaquetar el resultado antes de volver a guardarla en la memoria del ordenador.

Ejemplo: Obtener la representación interna de las mantisas de los siguientes números teniendo en cuenta que $n_m = 12$: $N_1 = 1001,1100110 \cdot 2^{-5}$ y $M_2 = 0,00000110110 \cdot 2^{34}$.

N	con m normalizada	m empaquetada
$1001,1100110 \cdot 2^{-5}$ $0,00000110110 \cdot 2^{34}$	$1,0011100110 \cdot 2^{-2}$ $1,10110 \cdot 2^{28}$	001110011000 101100000000

Casos Especiales:

s	e	m
---	---	---

- $e = e_{min} = 0$ y $m = 0 \Rightarrow N = 0$
- $e = e_{min} = 0$ y $m = 0 \Rightarrow$ la mantisa se almacena denormalizada y $S = 2^{ne-1} - 2 \Rightarrow E = -2^{ne-1} + 2$
- $e = e_{max} = 11\dots1$ y $m = 0 \Rightarrow \pm\infty$
- $e = e_{max} = 11\dots1$ y $m \neq 0 \Rightarrow \text{NaN: Not a Number}$, equivalente a indeterminada

Precisión y Redondeo:

Es difícil representar un número de forma exacta con $n_m + 1$ bits. El modelo de redondeo que se utiliza es el llamado Redondeo al par. Considera el bit menos significativo, que está en la posición $-n_m$ y los bits que ocupan las posiciones $-n_m - 1$ y $-n_m - 2$ del número a redondear, que se denominan *bit de redondeo* y *bit retenedor*, respectivamente.

El valor del bit de redondeo es el que se obtiene al realizar la operación. El valor del bit retenedor es 1 si alguno de los bits en las posiciones $-n_m - 2, -n_m - 3, \dots$ es 1. En caso contrario, vale 0.

Consideraciones:

- Si el bit de redondeo es 0, se trunca.
- Si el bit de redondeo es 1 y el bit retenedor es 0, sumo 1 al bit en n_m , salvo que ya sea par ($m_{nm} = 0$).
- Si el bit de redondeo es 1 y el bit retenedor es 1, se suma 1 al bit en la posición n_m .

Ejemplo:

Resultado ALU	Acción	Mantisa Redondeada
1.01101 10	Sumar 1	1.01110
1.01100 10	Truncar	1.01100
1.01101 10	Sumar 1	1.01110
1.01100 11	Sumar 1	1.01101
1.01100 01	Truncar	1.01100
1.01100 00	Truncar	1.01100

En cuanto a la precisión, hay que prestar atención a tres casos:

- ***Underflow*** o desbordamiento al cero.
- ***Overflow*** o desbordamiento.
- Comparación de dos números reales muy próximos.

Los dos primeros pueden provocar que un número no pueda representarse en nuestro ordenador. El tercero puede provocar que concluyamos que dos números son iguales cuando realmente no lo son.

Tipos de precisión:

Los números reales podrán representarse, al menos, con dos precisiones: simple y doble.

	n	n_m	n_e	S	E_{min}	E_{max}
Simple	32	23	8	127	-126	127
Doble	64	52	11	1023	-1022	1023

Formas de almacenar los números en las palabras de la memoria:

Palabras de 16 bits

Precisión sencilla:	s	exponente(8 bits)	mantisa(7 bits)	
			mantisa(16 bits)	

Precisión doble:	s	exponente(11 bits)	mantisa(4 bits)	
			mantisa (16 bits)	
			mantisa (16 bits)	
			mantisa (16 bits)	

Palabras de 32 bits

Precisión sencilla:	s	exponente(8 bits)	mantisa(23 bits)	

Precisión doble:	s	exponente(11 bits)	mantisa(20 bits)	
			mantisa (32 bits)	