TEMA 1

Introducción

PARTE I. Introducción a la Informática

1.Introducción.

- La Informática es un concepto "relativamente" moderno.
- Las primeras computadoras eran máquinas enormes, muy caras y muy complejas. Eran utilizadas para fines científicos y especiales y, por lo tanto, tenían poco efecto en la mayoría de la gente.
- Actualmente hay millones de ordenadores, de todos los tipos y tamaños en oficinas, fábricas, escuelas, hospitales, bancos, tiendas y en la mayoría de nuestras casas.
- Aun sin tener un contacto directo con los ordenadores no podemos negar o ignorar su presencia cotidiana. Basten los siguientes ejemplos:
    - Reserva y compra de un billete de tren.
    - Pagar en una tienda con nuestra tarjeta de crédito.
    - Recibir alguna propaganda por correo.
    - Acudir al hospital para hacernos un chequeo.
    - Recibir la nómina mensual.
    - Uso de Internet
    - Juegos
    - Manejo de máquinas y herramientas

2. Definición y origen del término Informática.

A lo largo de la historia, el hombre ha necesitado continuamente transmitir y tratar información, por ello no ha parado de crear máquinas y métodos para procesarla. Con este fin surge la Informática como una ciencia encargada del estudio de estas máquinas y métodos.

La Informática nace de la idea de ayudar al hombre en los trabajos rutinarios y repetitivos, generalmente de cálculo y de gestión.

Una definición de Informática podría ser esta: Informática es la ciencia que estudia el tratamiento automático de la información.

Dentro del trabajo en informática entran:

- El desarrollo de nuevas máquinas.
- El desarrollo de nuevos métodos de trabajo.
- La construcción de aplicaciones Informáticas.
- La mejora de los métodos y aplicaciones existentes.

El término Informática se creó en Francia, y procede de la contracción de las palabras: **Infor**mación Auto**mática**.

3. Aportación de los ordenadores.

El enorme desarrollo que ha tenido en los últimos 40/50 años la Informática, se debe sin duda al tremendo avance introducido por la tecnología electrónica en los ordenadores digitales.

Los ordenadores han permitido afrontar problemas que con los medios anteriores eran inabordables:

Muchos problemas sólo podían solucionarse por **procedimientos aproximados**, porque aun cuando se disponía del procedimiento teórico exacto, no era posible aplicarlo por requerir un tiempo de cálculo enorme.

- En muchos problemas se disponía de soluciones exactas calculadas para datos iniciales concretos muy estudiados y muy escogidos. Cualquier cambio en estos datos no era aceptado por requerir unos esfuerzos ingentes de reajustes.
- Muchos problemas, especialmente el **control de procesos industriales**, requerían un tiempo de respuesta tan breve para el proceso de miles y miles de datos, que era inabordable bajo cualquier coste su planteamiento, por no disponer de posibilidad material de realizar dichos cálculos a la velocidad necesaria.
- Para los problemas que nos obligan a disponer de una enorme cantidad de datos almacenados en línea y en forma continua, no existían sistemas de archivo de capacidades y costes aceptables.
- Problemas que requieren alto nivel de conectividad entre equipos.

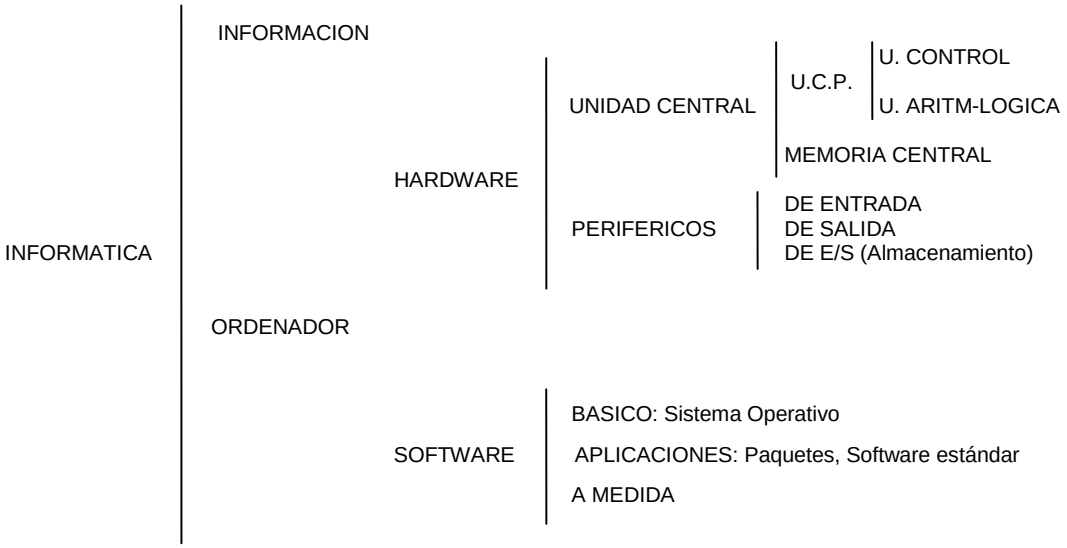
La principal aportación de los ordenadores ha sido la de permitir abordar con éxito estos problemas, pero también la de dar a cualquier Proceso de Datos, por insignificante que sea:

- Mayor velocidad de cálculo y procesos.

- Mayor fiabilidad (≅ 100%).
- Mayor seguridad y protección en los archivos.
- Mayor comodidad y velocidad en la recuperación de información archivada (de forma local o distribuida).
- Menores costes globales.

Pero hay que advertir que en el Proceso de Datos se plantean siempre soluciones metodológicamente independientes de los dispositivos físicos sobre los cuales vamos a construir el Sistema de Información final. Si la solución es buena conceptualmente, aguantará los cambios tecnológicos que puedan sobrevenir sin traumas ni grandes reajustes de la estructura fundamental.

4. Elementos y conceptos fundamentales.



Desde el punto de vista informático, el elemento físico utilizado para el tratamiento de los datos (materia prima) y obtención de la información (resultado) es el **ordenador**.

Un ordenador es una máquina compuesta de elementos físicos de tipo electrónico, capaz de realizar una gran variedad de trabajos a gran velocidad y con gran precisión siempre que se le den las instrucciones adecuadas.

El conjunto de órdenes o instrucciones que dirigen el proceso de un ordenador se llama **programa**. Al conjunto de uno o varios programas que realizan un

determinado trabajo completo se le denomina **aplicación** informática o simplemente aplicación.

El término **Sistema Informático** para referirse al conjunto de elementos necesarios para la puesta en funcionamiento de una aplicación.

La **información** es el elemento a tratar, y se define como todo aquello que permite adquirir cualquier tipo de conocimiento; por tanto existirá información cuando se dé a conocer algo que se desconoce. La información se elabora a partir de su materia prima: los **datos**.

Los datos suelen ser magnitudes numéricas directamente medidas o captadas, pero también pueden ser nombres o conjuntos de símbolos, imágenes, sonidos, etc.

El conjunto de operaciones que se realizan sobre un conjunto de datos conducentes a la obtención de información se denomina **tratamiento de la información.** Estas operaciones se recogen en el esquema siguiente:

- Tratamiento de la información
  - Entrada
    - Recogida de datos
    - Depuración de datos
    - Almacenamiento de datos
  - Proceso
    - Aritmético
    - Lógico
  - Salida
    - Recogida de resultados
    - Distribución de resultados

Se denomina **algoritmo** o proceso al conjunto de operaciones necesarias para transformar los datos iniciales en los resultados que se desean obtener en un determinado trabajo.

El algoritmo de resolución de un problema se determina en la fase de análisis, previa a la fase de automatización, según el siguiente esquema:

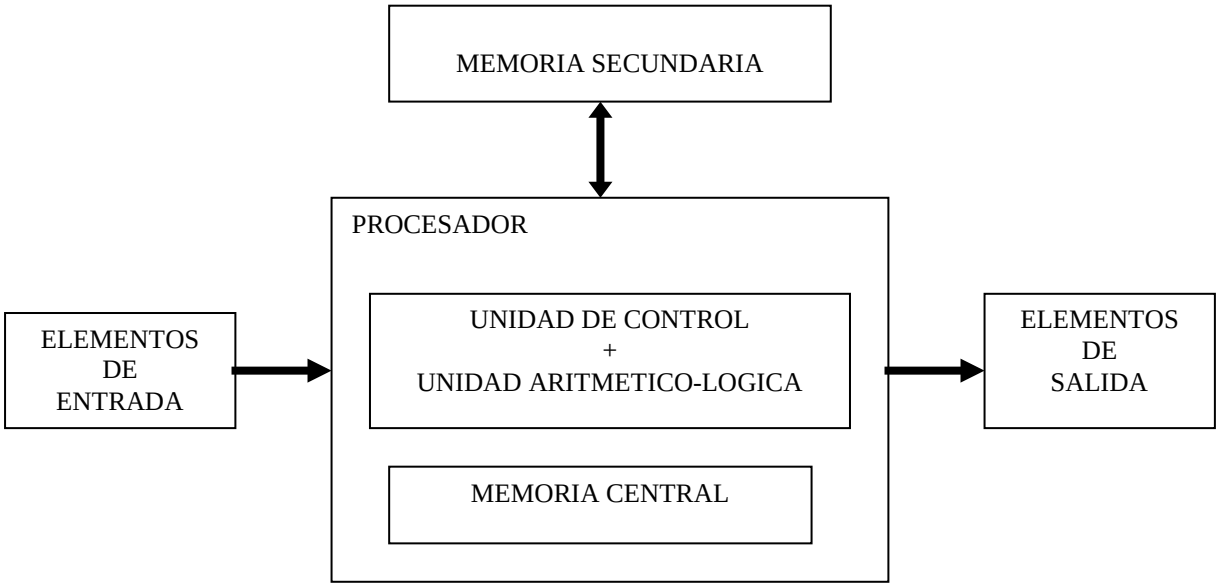
La informática se sustenta sobre tres pilares básicos:

- El elemento físico (Hardware)

- El elemento lógico (Software)
- El elemento humano

5. Esquema básico del elemento Hardware.

El hardware (Hw) representa la parte física de un sistema informático, es decir, todos los elementos materiales que lo componen. Su esquema básico es el siguiente:



Unidad central de proceso. (UCP o CPU (Central Process Unit))

Es el elemento principal del ordenador, y su función es la de coordinar, controlar o realizar todas las operaciones del sistema.

Consta fundamentalmente de:

- Unidad de Control. Gobierna el resto de elementos. Interpreta las instrucciones del programa, controla su ejecución y la secuencia en que éstas deben ejecutarse.
- Unidad Aritmético-Lógica. Se encarga de realizar las operaciones elementales de tipo aritmético y lógico.

Memoria central.

También denominada memoria principal, es el elemento encargado de almacenar los programas y los datos necesario para que el sistema realice un determinado trabajo.

Periféricos.

- **De entrada**. Son los elementos encargados de introducir los datos y los programas desde el exterior a la memoria central. Estos dispositivos, además de recibir la información del exterior la adaptan para que ésta sea inteligible por la máquina.
- **Memoria auxiliar**. Son dispositivos de almacenamiento masivo de información y su característica principal es la de retener esta información durante el tiempo que se desee, recuperándola cuando sea requerida.
- **De salida**. Son los dispositivos cuya misión es la de recoger y proporcionar al exterior los datos de salida de cada uno de los trabajos que se realicen en el sistema.

6. Software

El Sofware (Sw) de un sistema informático es el conjunto de programas necesarios para realizar tareas encomendadas al mismo.

Otra definición podría ser esta: El Sw es la parte lógica que dota al equipo físico de capacidad para realizar cualquier tipo de trabajo. Gracias al Sw (formado por multitud de programas que interactúan unos con otros) pueden ser manejados todos los recursos de que dispone un sistema para resolver cualquier aplicación informática.

En la actualidad, en un sistema informático, tiene mayor peso específico el Sw que el Hw (coste, mantenimiento, etc.).

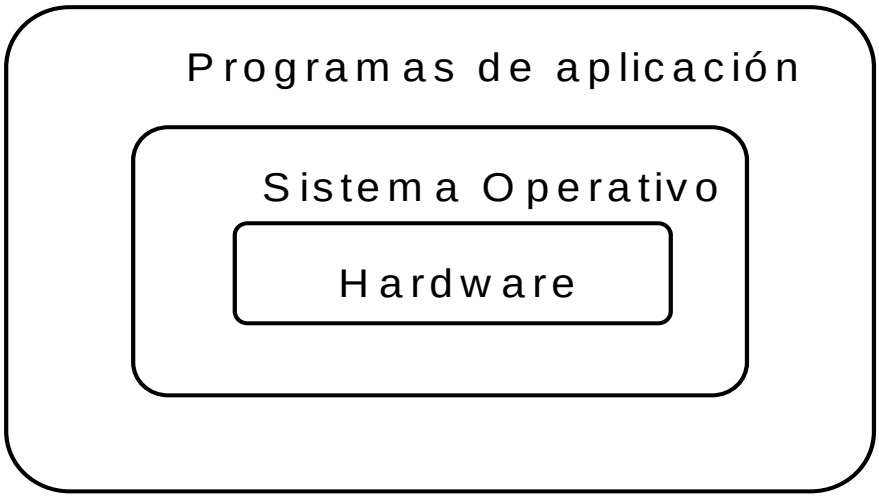
Desde el punto de vista de su funcionalidad, podemos clasificar el Sw en dos grandes grupos:

- **Sistema Operativo** (programas de sistemas)

  Controlan y optimizan la operación de la máquina. Es imprescindible para el funcionamiento de la máquina.

- **Programas de aplicación** (paquetes y software a medida).

  Indica a la máquina cómo resolver problemas específicos del usuario.



6.1. El Sistema Operativo.[1]

Hemos visto que el Sistema Operativo (SO) es el conjunto de programas de sistema de un ordenador. Dicho de otra forma, es un conjunto de programas que hace que el hardware sea algo útil. Controla que todos los dispositivos funcionen, o sea, controla el tráfico de información entre los distintos dispositivos.

Es lo que da personalidad al ordenador, en el sentido de que dos ordenadores con el mismo S.O. se comportan de la misma manera y al revés, el mismo ordenador trabajando bajo dos S.O. diferentes puede trabajar de distinta forma.

Ejemplos de SO comerciales son: MS-DOS, WINDOWS, UNIX, LINUX, etc.

Objetivos del Sistema Operativo.

Todos los SO tienen dos objetivos fundamentales. Uno es el de hacer posible el **uso eficiente de los recursos** del sistema y el otro es el de **ocultar las dificultades que supone el control directo del Hardware**. Varios SO estándar tienen un tercer objetivo, que es el de **presentar un interfaz estándar** al software de aplicación, independiente del hardware.

El primer objetivo para utilizar los recursos de forma eficiente se ve entorpecido por el hecho de que algunos dispositivos del ordenador funcionan mucho más rápidamente que otros; por ello, una de las misiones del SO es la de garantizar

[1] Ver anexo Sistemas Operativos

que los dispositivos más rápidos, como el procesador, no sean retenidos por otros dispositivos más lentos, como los periféricos.

Para conseguir el segundo objetivo de simplificar la operación del hardware de un ordenador, los SO generan una **máquina virtual**. Las personas que escriben programas sólo necesitan conocer la máquina virtual, sin necesidad de entrar en los detalles reales del hardware.

El tercer objetivo, es decir, el de presentar una máquina virtual estándar, está creciendo en importancia. Así se logran grandes ahorros en tiempo y coste en el desarrollo del software de aplicación. Los beneficiados son tanto las casas de software como los usuarios, ya que los éstos pueden ejecutar el mismo software en distintos tipos de hardware.

Funciones de un Sistema Operativo.

De manera esquemática, las funciones de un SO podemos resumirlas en:

- Cargar, iniciar y controlar la ejecución del programas.
- Controlar las operaciones de E/S.
- Coordinar la comunicación entre los dispositivos del sistema.
- Repartir los recursos entre los usuarios del sistema.
- Proteger los recursos contra accesos no permitidos.
- Diagnosticar los errores del sistema.
- Organizar y controlar los ficheros.
- Gestión de redes.

El SO es un administrador de recursos; administra el Hardware, los programas y los datos. Está orientado además a facilitar, automatizar y mejorar el rendimiento de los procesos en el ordenador (encadenamiento de programas, simultaneidad de operación de periféricos , tratamiento de errores, etc.).

El SO maneja las asignaciones de porciones de recursos físicos a cada programa individual (entre los distintos usuarios si es una máquina compartida), maneja directamente el Hardware para conseguir una operación eficiente al mismo tiempo que garantizando un buen tiempo de respuesta, y hace posible que distintos programas (de uno o varios usuarios) compartan la máquina sin interferirse mutuamente (**multiprogramación**).

Maneja las entradas y salidas de los distintos dispositivos, maneja la creación y mantenimiento de los ficheros y facilita la carga y ejecución de los programas compartiendo entre ellos el tiempo y la memoria disponible.

Características deseables de un SO.

Los SO tienen ciertos objetivos que cumplir y deben realizar una serie de tareas. Además deben tener una serie de características, entre las que se incluyen: **eficiencia, fiabilidad, facilidad de mantenimiento** y **un pequeño tamaño**.

La eficiencia supone que el SO debe ejecutar sus funciones de forma rápida. La fiabilidad del sistema es crucial, ya que un fallo en el SO puede inutilizar el ordenador que éste controla. Un SO es fácil de mantener cuando es posible modificarlo de manera sencilla para mejorarlo o corregir pequeños defectos. Para ello es crucial que el SO sea un programa bien estructurado.

Finalmente, además de todas estas características, un SO debe ser lo más pequeño posible, de tal manera que ocupe el menor espacio posible de almacenamiento. Habrá que buscar por tanto un punto de equilibrio entre un SO superpotente pero con un elevado tamaño y coste de administración y un SO no tan "inteligente" ni capaz pero con un tamaño y coste de administración menor.

6.2. Los lenguajes de programación.

La característica más importante de un computador es que se trata de una máquina programable; esto significa que sólo efectuará el proceso que se le indique mediante un programa, una secuencia de instrucciones almacenada en su memoria. La sintaxis y la semántica de estas instrucciones componen un lenguaje propio del procesador. Son diseñadas al mismo tiempo que él. Es el lenguaje máquina, el único lenguaje que entiende directamente, que es capaz de decodificar.

Puesto que es difícil de usar (sus instrucciones son secuencias de pulsos digitales, a nivel físico; o unos y ceros, a nivel lógico), se crean lenguajes que faciliten la programación.

El lenguaje máquina.

Cada ordenador puede operar directamente bajo un lenguaje que le es propio: el lenguaje máquina.

Como internamente el ordenador sólo reconoce dos signos diferentes, las instrucciones del lenguaje máquina deberán ser forzosamente una secuencia de unos y ceros. Por otra parte, un lenguaje máquina está tan ligado a la estructura

física de un ordenador particular que es totalmente dependiente de la máquina concreta que se está usando.

Cualquier programa, para poder ser ejecutado en un ordenador debe ser previamente almacenado en su memoria principal en lenguaje máquina.

Sin embargo, programar en lenguaje máquina, además de resultar absolutamente tedioso presenta otros inconvenientes como:

- El programa debe expresarse en términos de instrucciones muy elementales y muy poco potentes (las que el procesador entiende).
- Es necesario que el programador lleve el control explícito de cada dirección concreta de memoria.
- El programador debe consultar continuamente una tabla de los códigos que corresponden a cada instrucción (se trabaja únicamente con unos y ceros).
- El programa solo sirve para una máquina concreta, no es *portable*

Lenguaje ensamblador

Para facilitar la lectura y escritura de programas se han desarrollado otros lenguajes de programación.

El primer paso fue el desarrollo de lenguajes ensambladores. Este tipo de lenguaje se compone de las mismas instrucciones máquina, pero a las que se les han asignado nombres simbólicos, llamados nemotécnicos.

Para que el procesador comprenda un programa escrito en lenguaje ensamblador es necesario traducirlo. Es un trabajo que puede ser manual (consultando unas tablas) o puede crearse un programa que lo automatice, el programa "ensamblador".

Los ensambladores traducen cada instrucción a la correspondiente en lenguaje máquina y el programa traducido es el que realmente ejecuta el ordenador. Cada instrucción en lenguaje ensamblador corresponde con una única instrucción en lenguaje máquina. Las instrucciones de estos lenguajes son muy elementales, por lo que se llaman lenguajes de bajo nivel. Además no están unificados ya que dependen del ordenador con el que se trabaje, es decir, un programa para un sistema no vale en absoluto para otro distinto.

A pesar de la ventaja que supone el lenguaje ensamblador frente al lenguaje máquina, es también muy pesado escribir programas en ensamblador. Por ello, a mediados de los años 50, unos diez años después del nacimiento de los ordenadores digitales, comenzaron a desarrollarse otros lenguajes de programación, de uso más fácil, en los que las instrucciones se expresan de forma más próxima al lenguaje hablado.

Lenguajes simbólicos (Alto nivel)

Permiten usar códigos de operaciones nemotécnicos, nombres simbólicos para las variables y las direcciones de memoria, etc. Aparece el concepto de macroinstrucción (una instrucción en lenguaje de alto nivel equivale a varias instrucciones en lenguaje máquina cuando son traducidas)

Ventajas sobre los lenguajes máquina:

- Independencia del ordenador concreto a utilizar (tanto más cuanto más alto nivel).
- El programador puede concentrarse más en las peculiaridades del algoritmo.
- El aprendizaje de la programación resulta mucho más sencillo.
- Disminuye el riesgo de error y el tiempo de desarrollo y puesta a punto del programa.
- Se facilita la comprensión del programa para quien no lo ha escrito.
- Portabilidad

Además de clasificar los lenguajes en L. máquina, ensambladores y simbólicos podemos hacer otra clasificación más exhaustiva:

Lenguajes de programación	- Lenguaje máquina - Ensamblador (Bajo nivel)	
	- Alto nivel (Simbólicos)	- Gestión - Científicos - Propósito general - Específicos

En función del propósito de cada lenguaje tendrán características diferentes y ofrecerán más facilidades para ciertas operaciones. Así, un lenguaje orientado a la gestión ofrecerá facilidades para manejar grandes volúmenes de información, mientras un lenguaje científico ofrecerá mayor velocidad y exactitud en los cálculos.

6.3. Tipos de traductores.

Hemos dicho que un ordenador únicamente "entiende" su propio lenguaje máquina.

Así pues, un lenguaje escrito en un lenguaje de alto nivel deberá ser traducido a lenguaje máquina para poder ser ejecutado.

Según sea la forma de traducir el programa escrito en un lenguaje de alto nivel al que se denomina **programa fuente** al programa escrito en lenguaje máquina denominado **programa objeto**, se distinguen dos grandes grupos de traductores:

- **Intérpretes**
- **Compiladores**

6.3.1 Compiladores

Los programas compiladores realizan la traducción de un programa fuente, escrito en lenguaje de alto nivel, a su versión en código máquina. El programa traducido queda en forma de fichero, normalmente en el disco.

Usualmente, el programa compilador genera un informe con el resultado de la compilación. En él se recogen los errores, si los ha habido, así como información sobre las variables empleadas, número de líneas, etc.

6.3.2 Intérpretes

Este tipo de programas traductores procede de la siguiente forma: traduce una instrucción y la ejecuta, a continuación la siguiente, y así hasta el final del programa fuente. Pero no se genera un programa objeto independiente, como en el caso de los compiladores, que quede almacenado en el disco. Esto supone que la ejecución del programa es inseparable de su traducción, se vuelve a traducir cada vez que se ejecuta.

La ventaja fundamental de trabajar con un intérprete es que se dispone de un mayor control sobre la ejecución del programa. Esto supone una mayor facilidad en la depuración del programa, ya que suelen proporcionar herramientas como la posibilidad de seguimiento de las variables durante la ejecución; puntos de parada; ejecución instrucción a instrucción, etc.

Lo mejor es combinar las ventajas de uno y otro. Depurar y desarrollar un programa con un intérprete y una vez probado y visto que funciona adecuadamente, compilar para tener un programa ejecutable independiente sin necesidad de tener que traducirlo cada vez que se ejecute.

Bases de Datos.

Ver anexo

PARTE II. Representación de la Información

1. Introducción

El problema que nos ocupará en esta parte del tema es la representación de la información para su almacenamiento y procesamiento por un sistema informático. La información se codifica para su tratamiento de acuerdo con el medio de trabajo.

Por cuestiones de índole técnica, los circuitos electrónicos que forman un ordenador están capacitados para reconocer señales eléctricas de tipo digital (señales discretas), y lo más sencillo es manejar únicamente dos señales o estados diferentes. Por tanto, se hace necesario que los métodos de codificación internos tengan su origen en el sistema binario que únicamente utiliza dos símbolos, el 0 y el 1. El valor 0 o 1 es pues la mínima información que el ordenador maneja y la llamaremos **bit** (binary digit). El problema estriba en codificar la información a base de unos y ceros.

Veremos la codificación de dos tipos de información: la numérica y la textual. Previamente repasamos el sistema de codificación numérica que mejor conocemos, el sistema decimal.

2. El sistema decimal

La principal característica de este sistema es que dispone de diez símbolos que representan las cantidades del cero al nueve:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Con estos símbolos, repetidos o no, y con un determinado orden en su escritura, podemos representar cualquier cantidad.

Si leemos "92", un 9 seguido de un 2, sabemos que se representa la cantidad compuesta por 2 unidades y 9 decenas. Si le añadimos un 1 delante también le añadimos una centena. Pero si el 1 lo situamos tras el 2, ya tenemos 1 unidad, 2 decenas y 9 centenas.

El orden en que se expresan las cifras lleva asociado un peso de potencias de diez. Así:

$$91 = 9 \times 10 + 1 = 9 \times 10^1 + 1 \times 10^0$$

$192=1 \times l00+9 \times 10+2=1 \times 10^2+9 \times 10^1+2 \times 10^0$

$921 = 9 \times 100 + 2 \times 10 + 1 = 9 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$

Necesitamos la combinación del 1 y el 0 para representar la cantidad diez, no hay un único símbolo para esta cantidad base. El 0 a la derecha indica 0 unidades, pero el 0 a la izquierda no añade nada.

También se pueden representar cantidades fraccionarias, lo que llamamos decimales. El peso de las décimas es de 1/10, el de las centésimas es de 1/100, etc. Es decir, potencias negativas de la base:

$0{,}58 = 5 \times 10^{-1} + 8 \times 10^{-2}$

Todo esto que describimos es un código para la representación de información numérica.

Se dice que la razón de utilizar diez como cantidad base para el peso de los símbolos en este sistema consiste en que es algo "natural", dado que tenemos diez dedos que funcionan como un sencillo ábaco. Rompiendo la inercia cultural, con mayor o menor esfuerzo, se puede trabajar de hecho con cualquier cantidad como **base**.

3. Sistema de numeración. Teorema Fundamental de la Numeración

Supongamos, de manera equivalente, que sólo contamos con una mano, es decir, únicamente disponemos de cinco símbolos, del 0 al 4. Según esto, la cantidad "cinco" ya no se puede representar con un solo símbolo. Como en el sistema decimal, damos un peso al orden en que aparecen los símbolos en la codificación.

Así, la cifra de la derecha seguirá indicando las unidades, pero la inmediatamente a la izquierda representará grupos de cinco unidades, la siguiente, grupos de 25, etc. Por ejemplo:

veintisiete = $\mathbf{1} \times 5^2 + \mathbf{0} \times 5^1 + \mathbf{2} \times 5^0$

luego la cantidad "veintisiete" se codifica como 102 en un sistema que toma como **base** el 5.

Tanto en el sistema con base diez (decimal) como en el de base cinco, utilizamos un método común y generalizable a cualquier base, que viene enunciado por el Teorema Fundamental de la Numeración (por simplificar no vamos a tratar parte fraccionaria).

Sea:

- **B** la base (número de símbolos de que se compone el código).
- **e** número de dígitos menos uno, puesto que los dígitos se nombrarán desde A_0 hasta A_e, el más significativo.

Pues bien, dada una cantidad expresada en el sistema de numeración de base B y dada por los símbolos:

$$A_e \ldots A_1 A_0$$

su valor en base decimal viene dado por:

$$A_e \times B^e + \ldots + A_1 \times B^1 + A_0 \times B^0$$

Por ejemplo: supongamos que en un caso e = 3, B = 4 y $A_3 = 3$, $A_2 = 1$, $A_1=2$, $A_0 = 2$, entonces tenemos, expresada en base 4, la cantidad

$$3122_{(4}$$ (el subíndice indica la base en que se expresa la cantidad)

que, expresada en base decimal, sería:

$$3 \times 4^3 + 1 \times 4^2 + 2 \times 4^1 + 2 \times 4^0 =$$
$$= 3 \times 64 + 1 \times 16 + 2 \times 4 + 2 \times 1 =$$
$$= 192 + 16 + 8 + 2 = 218_{(10}$$

Dada una cantidad expresada en un sistema de base B, el teorema nos proporciona la forma de traducirlo a su expresión decimal. Pero también nos da el camino inverso, cómo representar cualquier cantidad decimal en otra base cualquiera B.

Este proceso, inverso al proporcionado por el teorema, consiste en obtener los sucesivos restos de dividir la cifra decimal entre la base B, que serían los

coeficientes A_i, hasta que el cociente sea menor que el divisor (la base). Este cociente es el dígito más significativo del resultado, A_e.

Ejemplo: Sea la cifra decimal 483 y queremos expresarlo en base 7:

483	I 7				
0	69	I 7	resto 0 = A_0	y	cociente 69
	6	9 I 7	resto 6 = A_1	y	cociente 9
		2 1	resto 2 = A_2	y	cociente 1 = A_3

luego en base 7 es $1260_{(7}$.

Para comprobarlo, no tenemos más que utilizar el teorema en el mismo sentido de su enunciado:

$$1260_{(7} = 7^3 + 2 \times 7^2 + 6 \times 7 = 343 + 98 + 42 = 483_{(10}$$

4. El sistema binario

En este apartado vamos a practicar especialmente con el sistema de numeración en base 2. La razón está ya descrita en la introducción del tema: nuestro medio es el de la electrónica digital binaria.

4.1. Codificación binaria

Si codificamos con un 1 cuando un biestable (elemento electrónico capaz de encontrarse en dos únicos estados diferentes) deja pasar la corriente, y con un 0 el caso opuesto, en varios biestables podemos almacenar un número expresado en el sistema base 2.

En este sistema sólo disponemos de dos dígitos, el 0 y el 1, de tal forma que la cantidad "dos" se debe representar ya con la combinación de los dos símbolos, sería el 10, el "tres" será el 11, el "cuatro" será el 100, etc. Evidentemente, se necesitan bastantes posiciones para representar una cantidad relativamente pequeña, que en codificación decimal ocupa mucho menos (una tercra parte

aproximadamente), pero no es ese el problema. Y la ventaja es el tratamiento electrónico de la información, nada menos. Veamos algunos ejemplos para familiarizarnos con este sistema.

La cantidad $1100111010_{(2}$, ¿qué forma tendrá en su expresión decimal?

Sabemos que el peso de cada posición viene dado por las sucesivas potencias de la base, en este caso 1, 2, 4, 8, etc. Por tanto, aplicando:

$$A_e \ldots A_{0(B} = A_e \times 2^e + \ldots + A_1 \times 2^1 + A_0 \times 2^0$$

En este caso quedaría:

$$11\ 0011\ 1010\ _{(2} = 2 + 8 + 16 + 32 + 256 + 512 = 826$$

Planteemos el problema de la forma inversa. ¿Cuál es la expresión binaria del decimal 826? Ya sabemos que la solución se encuentra por divisiones sucesivas por la base:

826 I 2		$A_0 = 0$
0 413 I 2		$A_1 = 1$
1 206 I 2		$A_2 = 0$
0 103 I 2		$A_3 = 1$
1 51 I 2		$A_4 = 1$
1 25 I 2		$A_5 = 1$
1 12 I 2		$A_6 = 0$
0 6 I 2		$A_7 = 0$
0 3 I 2		$A_8 = 1$
1 1		$A_9 = 1$

Luego la solución es (como ya sabíamos):

$$11\ 0011\ 1010_{(2}$$

No se puede pensar en otro sistema de numeración que necesite menos símbolos distintos; con ceros y unos representamos cualquier cantidad.

Cada posición ocupada por un dígito binario se llama **bit**. Y a la agrupación de 8 bits se le llama **byte** (o también **octeto**).

Así, si disponemos de esos 8 bits, de un byte, las cantidades que se pueden tratar recorren desde el 0 (los 8 bits a 0) hasta el $2^8 - 1 = 255$ (los 8 bits a 1) . Esto es, en total se pueden representar 256 valores distintos, y su rango de representación, del 0 al 255. Si disponemos de 16 bits, 2 bytes, el rango irá de 0 a $2^{16} - 1 = 65.535$, con 65.536 valores distintos.

El rango de representación cambia si se tiene en cuenta la posibilidad de codificar números negativos, como veremos en el apartado correspondiente de este mismo tema.

Lo mismo que ocurre con el sistema métrico decimal, se da nombres a cantidades clave. Así, a 1.024 bytes se le llama un **K**, a 1.024 K's se le llama un **mega**, a 1.024 megas se le llama un **giga**, etc.

5. El sistema hexadecimal

Hasta ahora sólo hemos tratado códigos de numeración de base inferior a la decimal. Pero ¿qué pasa si decidimos que la base sea mayor que 10? Entonces necesitaríamos más de 10 signos. Este es el caso del sistema de numeración en base 16 o hexadecimal; más tarde veremos cuál es su importancia.

En él existen 16 dígitos, del 0 al 9 normales, y 6 nuevos, que representaremos por las letras de la A a la F, del 10 al 15 decimales.

decimal:	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
hexadecimal:	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Las conversiones de hexadecimal a decimal y viceversa siguen los mismos algoritmos que utilizamos para los sistemas de numeración de bases inferiores a 10.

Ejemplo: Vamos a pasar el valor $1234_{(10}$ a hexadecimal dividiendo sucesivamente por la base 16 :

1234	I 16		$A_0 = 2$
114	77	I 16	$A_1 = D$
02	13	4	$A_2 = 4$

Luego

$$1234_{(10} = 4D2_{(16}$$

A la inversa, si queremos obtener el valor decimal de $45AB_{(16}$, hacemos el desarrollo de las potencias:

$$45AB_{(16} = 11 \times 16^0 + 10 \times 16^1 + 5 \times 16^2 + 4 \times 16^3 = 17.795_{(10}$$

Como se puede observar, una cifra expresada en hexadecimal ocupa menos posiciones, necesita menos dígitos, que en decimal. Y si comparamos con el binario, aún es mayor la diferencia.

Ejemplo: ¿Qué forma tiene en binario el valor $A5_{(16}$?

$$A5_{(16} = 5 \times 16^0 + 10 \times 16^1 = 165_{(10} = 10100101_{(2}$$

En este caso, en binario ocupamos justamente cuatro veces más de posiciones ($16 = 2^4$) que en hexadecimal. Además, observamos que se obtendría el mismo resultado **escribiendo cada cifra en binario**, pero **reservando para cada una cuatro posiciones binarias**:

$$A_{(16} = 1010_{(2} \quad \text{y} \quad 5_{(16} = 0101_{(2}$$

Luego

$$A5_{(16} = 1010\ 0101_{(2}$$

Esta es una propiedad fundamental del hexadecimal: su gran facilidad para la traducción a binario (y viceversa) y el hecho de ser más compacto.

Para pasar de binario a hexadecimal se agrupan los bits de cuatro en cuatro, empezando por las posiciones menos significativas, y se traduce a hexadecimal cada cuarteto.

Ejemplos:

- Pasar a hexadecimal el binario $111\ 1010\ 1011\ 0101_{(2}$

Traducimos cada cuarteto (de dcha. a izda.):

$0101 \rightarrow 5 \qquad 1011 \rightarrow B \qquad 1010 \rightarrow A \qquad 111 \rightarrow 7$

Luego el hexadecimal es 7AB5.

- A la inversa, el hexadecimal $6452_{(16}$ en binario sería:

$6 \rightarrow 0110 \qquad 4 \rightarrow 0100 \quad 5 \rightarrow 0101 \quad 2 \rightarrow 0010$

Luego el binario es 110 0100 0101 0010

Generalización:

Es una propiedad generalizable a aquellos sistemas de numeración cuyas bases son potencias de 2. Así, para el sistema octal, base 8 ($2^3 = 8$), se seguirán los mismos métodos de traducción octal-binario y viceversa que para el hexadecimal, con la salvedad de que ahora haríamos agrupaciones de tres dígitos binarios.

Ejemplos:

- Pasar a binario el octal $527_{(8}$

$5 \rightarrow 101 \qquad 2 \rightarrow 010 \qquad 7 \rightarrow 111$

Por tanto, el binario correspondiente es $101010111_{(2}$

- Pasar a octal el binario $10101110001_{(2}$

$001 \rightarrow 1 \qquad 110 \rightarrow 6 \qquad 101 \rightarrow 5 \qquad 10 \rightarrow 2$

Por tanto, el octal correspondiente es $2561_{(8}$

6. Codificación de la información en el ordenador

6.1 Números en Complemento a 2

Es uno de los métodos utilizados para resolver el problema de la representación del signo .

Para obtener el complemento a 2 de un número binario, se cambian los ceros por unos y los unos por ceros y a continuación se suma 1 (el acarreo si se produce, se desprecia). Es imprescindible conocer a priori el número de bits utilizados para el almacenamiento.

Ejemplo:

Utilizando 4 bits, el 3 en binario es

$0011_{(2}$

invirtiendo,

1100

y sumándole 1,

1101, que representa el -3

a la inversa, complementando el 1101, obtenemos

0010

y sumándole 1,

0011, que representa al 3 decimal

Ejemplo:

si disponemos de 8 bits para almacenamiento de números, 14 en binario es 0000 1110

1111 0001 + 1 = 1111 0010 que representa al -14

para comprobarlo, sumamos 14 y -14

```
   0000 1110
+  1111 0010
 1 0000 0000
```

que, despreciando el acarreo, se obtiene el 0.

El único problema con este sistema surge cuando se amplía el número de bits que contienen al valor. Por ejemplo, el valor 24 codificado en un byte es:

0001 1000

Para pasar ese valor a 2 bytes, sólo hay que añadirle 8 ceros a la izquierda,

0000 0000 0001 1000

pero si representamos -24

1110 1000

para ampliarlo a 2 bytes, conservando el valor, hay que añadirle ocho unos a la izquierda,

1111 1111 1110 1000

Luego los bits de ampliación repiten el bit de signo.

6.2. Coma flotante

Se divide el número de bits dedicados a almacenar la magnitud real en dos zonas, una para la mantisa y otra para el exponente. Esto parte de la posibilidad de representación en una forma normalizada mantisa-exponente, donde la mantisa contiene los dígitos significativos y el exponente el factor de escala, es decir, donde se sitúa la coma decimal, por eso se habla de coma flotante.

De la gran variedad de formatos de este tipo, vamos a ver el más generalizado, el llamado de simple precisión. En él se dedican 24 bits para la mantisa, el primero de ellos para el signo, y 8 bits para el exponente.

La mantisa aparece de forma normalizada, esto es, sobreentendiendo la coma justo antes del primer bit, "0,mantisa", y adecuando el exponente de acuerdo a esa operación. Así, la mantisa de 12,25 es 1225 (0,1225 de valor real), con el exponente 2.

$12{,}25 = 0{,}1225 \times 10^2 \rightarrow$ mantisa normalizada = 1225

exponente = 2

Pasando a binario la mantisa y el exponente:

1225 = 10011001001

2 = 10

el número, por tanto, sería:

0000 0000 0000 0100 1100 1001 0000 0010

6.3. Representación de texto.

El texto está compuesto de caracteres alfanuméricos, los alfabéticos, las cifras decimales, y los caracteres especiales, signos de puntuación y de control. Para su representación se asigna un código, un valor entero, a cada carácter diferente. Este código es (principalmente) el código ASCII.

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(espacio)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□