1. INTRODUCCIÓN

Conceptos: *Informática, Ordenador, Programa, Dato, Bit, Byte, Hardware, Software, Lenguaje de Programación, Código máquina, Lenguaje ensamblador, Lenguaje de alto nivel, Compilador, Intérprete, Ciclo de vida, Codificación, Álgebra booleana.*

Resumen: Este primer tema se dedica a los conceptos computacionales básicos más relacionados con la arquitectura de los ordenadores, como su estructura física y funcional, la representación de datos, la aritmética del ordenador y los principios algebraicos fundamentales del diseño de circuitos. Se hace especial hincapié en las características específicas de los ordenadores personales al ser éstos la herramienta fundamental de la mayoría de los futuros ingenieros. Además se analizan las herramientas lógicas que permiten la comunicación con los elementos físicos del ordenador, como son: los programas, los lenguajes de programación de bajo y alto nivel y los sistemas operativos.

Objetivos específicos. Al finalizar el tema, el alumno deberá ser capaz de:

a) Definir conceptos básicos informáticos (*Conocimiento*)
b) Describir la estructura de un ordenador (*Conocimiento*)
c) Definir el concepto de lenguaje de programación y describir el ciclo de vida del software (*Conocimiento*)
d) Describir los formatos de codificación de datos más significativos en informática (*Conocimiento*)
e) Convertir un valor numérico natural, entero o real a binario (y viceversa) (*Aplicación*)
f) Definir el álgebra booleana y decribir sus operaciones básicas (*Conocimiento*)

1.1. TERMINOLOGÍA BÁSICA

El término *informática* es un neologismo francés que viene de la contracción de *infor*mación auto*mática* y que se ha tomado como la traducción de la expresión original inglesa *Computer Science*.

La **informática** puede definirse como la disciplina que intenta dar un tratamiento científico a una serie de materias: el diseño de computadores, la programación de los mismos, el procesamiento automático de la información y el diseño de algoritmos para la resolución de problemas de diversa índole.

Un **computador** u **ordenador** es una máquina electrónica que bajo el control de uno o varios *programas*, de forma *automática* acepta y procesa *datos*, efectuando operaciones lógicas y aritméticas con ellos, y proporciona los resultados del proceso. De forma algo más vaga se podría decir que un ordenador es una máquina de propósito general destinada al procesamiento de información.

El término **automática** indica que no hay intervención humana.

Un **programa** es un conjunto ordenado de instrucciones que controlan el funcionamiento de un ordenador para llevar a cabo una tarea específica. Para Niklaus Wirth:

Programa = Algoritmos + Estructuras de Datos

Un **algoritmo** es una secuencia de reglas o pasos precisos que permiten obtener unos resultados a partir de unos datos. Por ejemplo: una receta. El algoritmo debe ser claro (no ambiguo) y finito en términos de recursos empleados. Asimismo debe ser independiente del lenguaje que se emplee para implementar cada paso.

Un **programa** es la traducción de un algoritmo a un lenguaje de programación; la receta (algoritmo) puede traducirse al francés o al español. Un programa puede estar formado por un conjunto de algoritmos, cada uno de los cuales, lleve a cabo una tarea específica. Por otro lado, existen diferentes **estructuras de datos** definidas por las relaciones o las formas en que pueden agruparse los datos que las constituyen. La estructura actual de los ordenadores no realiza distinción alguna entre datos y programas dada la naturaleza de éstos, como se verá más adelante.

En general y atendiendo al formato de codificación de los datos, los dispositivos que procesan datos pueden clasificarse en analógicos y digitales. Un dispositivo **analógico** almacena y procesa datos que están representados en términos de una variable continua, por ejemplo, en voltaje eléctrico. Son rápidos pero poco precisos y difíciles de programar. Sólo sirven para tareas sencillas. Un reloj de agujas o un termómetro de mercurio serían dispositivos analógicos. En un dispositivo **digital** la variable puede tomar un conjunto discreto de valores, por ejemplo, datos en formato binario: se emplean órganos o elementos con dos estados estables. Es mucho más flexible y preciso. La codificación viene condicionada por los elementos físicos empleados en la construcción del computador: un interruptor abierto o cerrado, un transistor conduciendo o no, una cinta magnética magnetizada en un sentido o en otro, una señal eléctrica caracterizada por un pulso o por su ausencia… Cuando en lo sucesivo se hable de *ordenador* será a este tipo de dispositivo al que se haga referencia.

Un **dato** es cualquier información codificada de forma que pueda ser aceptada y procesada por un computador. Como se verá más adelante, será un valor que toma una variable para cada uno de los elementos de un conjunto.

Para que pueda ser almacenada, transferida de una parte a otra y procesada por el ordenador, la codificación de la información se lleva a cabo en formato binario. Un dígito binario o **bit** (resultado de la contracción de la expresión ***bi****nary digit*) es la unidad más

pequeña y fundamental de información. Un **octeto o byte** es un conjunto de ocho bits y es la unidad práctica de información. Como el byte es una unidad de representación/almacenamiento de información relativamente pequeña para muchos usos, se suelen emplear múltiplos:

1 Kilobyte (o KB) = 2^{10} bytes = 1024 bytes ≈ 10^3 bytes
1 Megabyte (o MB) = 2^{10} KB = 2^{20} bytes = 1 048 576 bytes ≈ 10^6 bytes
1 Gigabyte (o GB) = 2^{10} MB = 2^{30} bytes = 1 073 741 824 bytes ≈ 10^9 bytes
1 Terabyte (o TB) = 2^{10} GB = 2^{40} bytes ≈ 10^{12} bytes
1 Petabyte (o PB) = 2^{10} TB = 2^{50} bytes ≈ 10^{15} bytes
1 Exabyte (o EB) = 2^{10} PB = 2^{60} bytes ≈ 10^{18} bytes

1.2. BREVE HISTORIA DE LA INFORMÁTICA

La operación de calcular con los dedos en la Prehistoria cedió su nombre al *cálculo digital*, sinónimo también del término actual *informática*. El intento de automatizar y acelerar las operaciones de cálculo aritmético marca ya desde el principio de su historia la evolución de la informática. El **ábaco** es el primer instrumento mecánico de cálculo conocido y está considerado el primer prototipo de máquina de calcular.

Ya en el siglo XVII aparecen los primeros dispositivos mecánicos construidos a base de engranajes. En 1653, Blaise **Pascal** (1623-1662) diseña la primera calculadora mecánica automática que suma y resta para ayudar a su padre que era recaudador de impuestos. En 1673, Gottfried Wilhem **Leibniz** (1646-1716) construye una máquina para multiplicar y dividir. En 1822, **Charles Babbage** (1791–1871) construye una *máquina diferencial*, que calcula diferencias hasta con ocho decimales para corregir unas tablas de logaritmos llenas de errores. Como sus predecesoras todavía era una máquina de uso específico para resolver un tipo muy determinado de problema. En 1833 concibe la *máquina analítica*, es decir, una máquina de uso general o universal en la que, para realizar cada tarea concreta, hay que programarla. Ya se establece la diferencia entre la entrada de datos, el programa y la salida de resultados.

George Boole (1815–1864), a mediados del siglo XIX, desarrolla el álgebra booleana, base del diseño de circuitos en ordenadores digitales. En 1894, **Hollerith** (1860-1929) construye una máquina capaz de leer tarjetas perforadas que se emplea para mecanizar el censo de los Estados Unidos de América. **Alan Turing** (1912–1954), a principios del siglo XX, define un modelo matemático de computador consistente en un ordenador teórico, completamente abstracto, que pudiera llevar a cabo cualquier cálculo realizable por un ser humano.

En 1944 aparece el primero de los dispositivos electromecánicos basado en relés: se le llamó **MARK-I**. Es una máquina universal construida en Harvard y financiada por IBM. Podía realizar todas las operaciones aritméticas básicas y calculaba logaritmos y senos. Medía 17 metros de longitud y 3 de altura. Contenía 760.000 piezas conectadas por 800 km de cables. Tardaba 10 segundos para multiplicar (comparar con los 10^{-9} s que tarda el CRAY-2) debido a la gran cantidad de partes mecánicas y electromecánicas que poseía. El **MARK-II**, ya totalmente eléctrico, aparece en 1947.

Después surgen los dispositivos eléctricos, basados en válvulas de vacío: el **ENIAC** (contracción de la expresión *Electric Numerical Integrator And Computer*) construido en Pennsylvania en 1946. Consumía 150.000 vatios, por lo que, disponía de un sistema de refrigeración y cuando se ponía en funcionamiento las luces de la ciudad tenían menos intensidad. Ocupaba 130 m^2 y pesaba 30 toneladas. Poseía 18.000 tubos de vacío, 70.000 resistencias y 10.000 condensadores, 6.000 interruptores y era 5.000 veces más rápido que el MARK-I. Empleaba todavía la aritmética decimal y no la binaria. Era capaz de hacer el

trabajo de 100 ingenieros durante un año en dos horas. Aproximadamente cada dos días el ordenador dejaba de funcionar porque uno de sus 18.000 tubos se fundía. A modo de anécdota, el término *debugging*, empleado actualmente para hacer referencia a la depuración o detección de errores en un programa, se acuñó en el momento en que una polilla (*bug*) provocó un cortocircuito al introducirse entre los tubos de vacío de la máquina.

En 1945 **John von Neumann** (1903–1957) pone las bases de la arquitectura básica de un ordenador: utilización de aritmética binaria y almacenamiento en memoria de las instrucciones del programa junto con los datos. Esto último aceleró las operaciones del ordenador. El uso de anillos magnéticos de ferrita resolvió la necesidad de utilizar elementos de memoria seguros, de bajo coste y rápidos. El primer ordenador de este tipo fué el *Torbellino*, utilizado por su rapidez en el control del tráfico aéreo.

Con el desarrollo de los transistores (contracción de la expresión *transfer resistor*) surgen los dispositivos electrónicos. Como el volumen físico y el precio se reducen considerablemente, ya son comercializa la serie 7.000 de IBM. Paralelamente, se aumenta la seguridad y la velocidad de proceso. Mas tarde la concentración de elementos electrónicos en circuitos integrados o *chips* inicia una nueva generación de dispositivos electrónicos. Aparecen los primeros miniordenadores (PDP-1). La siguiente generación se basa en la utilización de arquitecturas masivamente paralelas: Hipercubo de Intel, *Thinking Machines*,... Suelen distinguirse diversas generaciones de computadores dependiendo de los elementos físicos constituyentes:

- *1ª Generación*: Caracterizada por la tecnología del relé, el tubo de vacío y los núcleos de ferrita (1945-1958). La velocidad de proceso se mide en milésimas de segundo. Disipan una gran cantidad de energía calorífica. Los trabajos se realizan uno a uno (monoprogramación) y no existe sistema operativo. Se programa en lenguaje máquina directamente.
- *2ª Generación*: Caracterizada por la utilización de transistores y diodos (1957-1968). La velocidad de proceso se mide en millonésimas de segundo. Se utilizan núcleos de ferrita para las memorias. Aparecen los primeros lenguajes simbólicos y los sistemas operativos.
- *3ª Generación*: Caracterizada por la integración de circuitos o *chips* (1964-1977). La velocidad se mide en nanosegundos. Aparecen los lenguajes de alto nivel, la multiprogramación, el multiproceso y las bases de datos.
- *4ª Generación*: Caracterizada por la integración masiva de circuitos (1972-). El tiempo de proceso se mide en picosegundos. Surgen los lenguajes de cuarta generación, la inteligencia artifical y los sistemas expertos.

Tabla 1. Generaciones de ordenadores en función del número de circuitos integrados por mm^2

1960	Short Scale Integration	SSI	<100
1966	Medium Scale Integration	MSI	100-1000
1969	Large Scale Integration	LSI	1000-10000
1975	Very Large Scale Integration	VLSI	>10000

1.3. EL ORDENADOR

Un ordenador se compone de *hardware* y *software*. El ***hardware*** es el soporte físico del ordenador, es decir, los componentes físicos que lo constituyen: conjunto de circuitos electrónicos, cables, carcasas,... Por ejemplo, el diseño de circuitos y la electrónica son disciplinas relacionadas con el hardware.

El ***software*** es el soporte lógico, es decir, los programas que dirigen el funcionamiento del ordenador. Los programas van a ser el principal objeto de estudio de esta obra. La

construcción de algoritmos, la estructura de datos y la metodología para desarrollar programas son varios ejemplos de disciplinas relacionadas con el software.

1.3.1. Clasificación de los ordenadores

Los ordenadores pueden clasificarse de acuerdo con sus capacidades en:

a) **Microcomputadores**: *Ordenadores personales* con dispositivos de almacenamiento interno y capacidad limitada (Mb, Gb).
b) **Minicomputadores**: Multiusuario (decenas), almacenamiento medio (Gb, Tb).
c) ***Mainframes***: Mayores computadores, decenas o cientos de terminales, almacenamiento masivo,...
d) **Supercomputadores**: Desarrollo reciente, muy rápidos: optimizados para cálculo científico.

1.3.2. Estructura física de un ordenador

Cualquiera de los tipos de ordenador anteriores se componen de distintas partes, cada una de las cuales lleva a cabo una labor específica.

El **procesador** o unidad central de proceso (CPU) es el dispositivo donde se lleva a cabo el tratamiento de los datos. Los ordenadores de tamaño medio o grande pueden tener más de uno. Las funciones básicas del procesador son:

a) Manipulación de datos: operaciones elementales de tipo aritmético (sumas, desplazamientos de bits…) y lógico (operaciones del álgebra de Boole) llevadas a cabo por la Unidad Aritmético-Lógica (ALU).
b) Control de la ejecución de los programas: transferencias de datos con la memoria, control de la secuencia en la que se ejecuta el programa,... Se lleva a cabo en la Unidad de Control (UC). Contiene un reloj o generador de pulsos que sincroniza todas las operaciones elementales del ordenador.
c) Adicionalmente, dispone de una memoria propia limitada para el almacenamiento de resultados intermedios, información necesaria para el control de la ejecución del programa,...

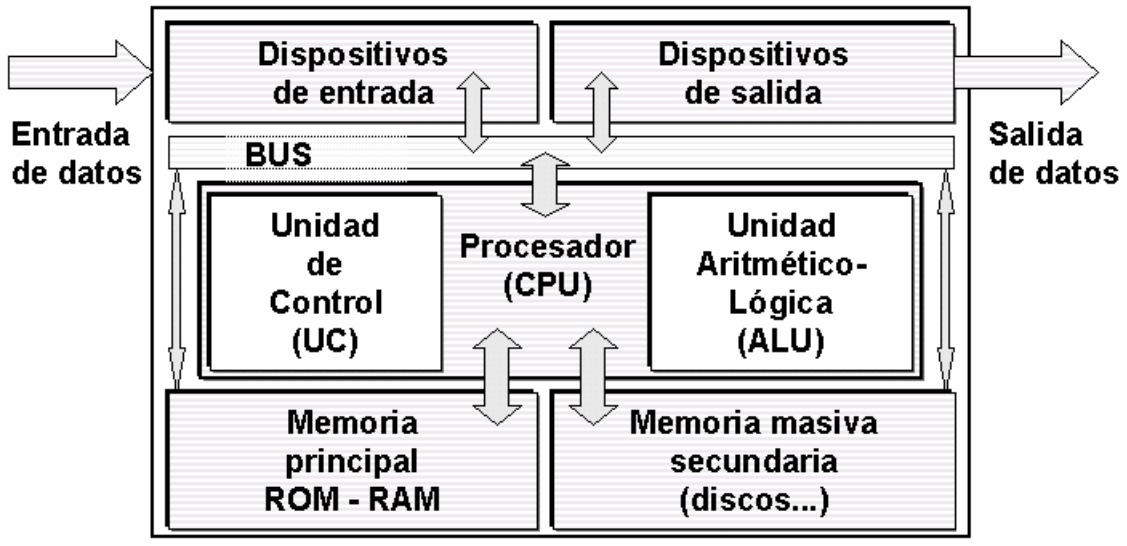

Figura 1. Esquema funcional de un ordenador

Las principales características de los procesadores son:

a) Tamaño en bits de los datos que maneja (registros internos).
b) Tamaño de los datos con los que se comunica al exterior.
c) Velocidad del reloj de sincronización (tiempo empleado en un ciclo elemental de máquina) o su inversa, la frecuencia de ciclo que se mide en kiloherzios (kHz) o Megaherzios (MHz).
d) Variedad del conjunto de instrucciones que es capaz de realizar.
e) Rendimiento global del procesador: número de instrucciones capaz de realizar por segundo (MIPS) o número de operaciones en coma flotante por segundo (MFLOPS).

Existen distintos tipos de procesadores que pueden clasificarse en:

a) CISC (*Complete Instruction Set Computers*): los habitualmente empleados en ordenadores personales.
b) RISC (*Reduced Instruction Set Computers*): disponen de un limitado número de instrucciones pero son capaces de ejecutarlas mucho más rápidamente que un procesador normal.
c) *Transputers*: Son computadores en un chip ya que incluyen memoria y controladores de puertos. Interesantes para el diseño de computadores masivamente paralelos.
d) Procesadores vectoriales: Capaces de realizar la misma operación simultáneamente sobre un cierto número de datos. Muy adecuados para operaciones matriciales.
e) Procesadores *pipeline*: Capaces de procesar varias instrucciones simultáneamente, no esperando a que cada instrucción se haya ejecutado completamente, sino comenzando con la siguiente cada vez que una parte del decodificador de instrucciones quede libre.

La **memoria**, *principal*, *central* o *interna*, es el dispositivo formado por circuitos electrónicos integrados donde se almacena, *se carga*, información -instrucciones y datos- antes, durante y después de su tratamiento por su procesador. Las características principales de la memoria son su tamaño (capacidad de almacenamiento de datos) y el tiempo de acceso (tiempo necesario para recuperar un dato). Para acceder a una información determinada, se emplea su dirección o posición en la memoria. La memoria puede clasificarse en:

a) Memoria ROM o *Read Only Memory* (memoria de sólo lectura). Se emplea para almacenar datos y programas necesarios para la operación del ordenador. Es una memoria no volátil, es decir, los datos no se pierden cuando se apaga el ordenador.
b) Memoria RAM o *Random Access Memory* (literalmente, memoria de acceso aleatorio). Permite la lectura y escritura de datos y es la empleada normalmente para el almacenamiento de la información que se está tratando en un momento dado. Es volátil, es decir, los datos se pierden cuando se apaga el ordenador.
c) Memoria caché: Parte de la memoria RAM normalmente de acceso más rápido donde se almacenan datos de uso más frecuente.
d) Memoria virtual: Memoria RAM residente físicamente en los sistemas de almacenamiento masivo y gestionada por el sistema operativo.

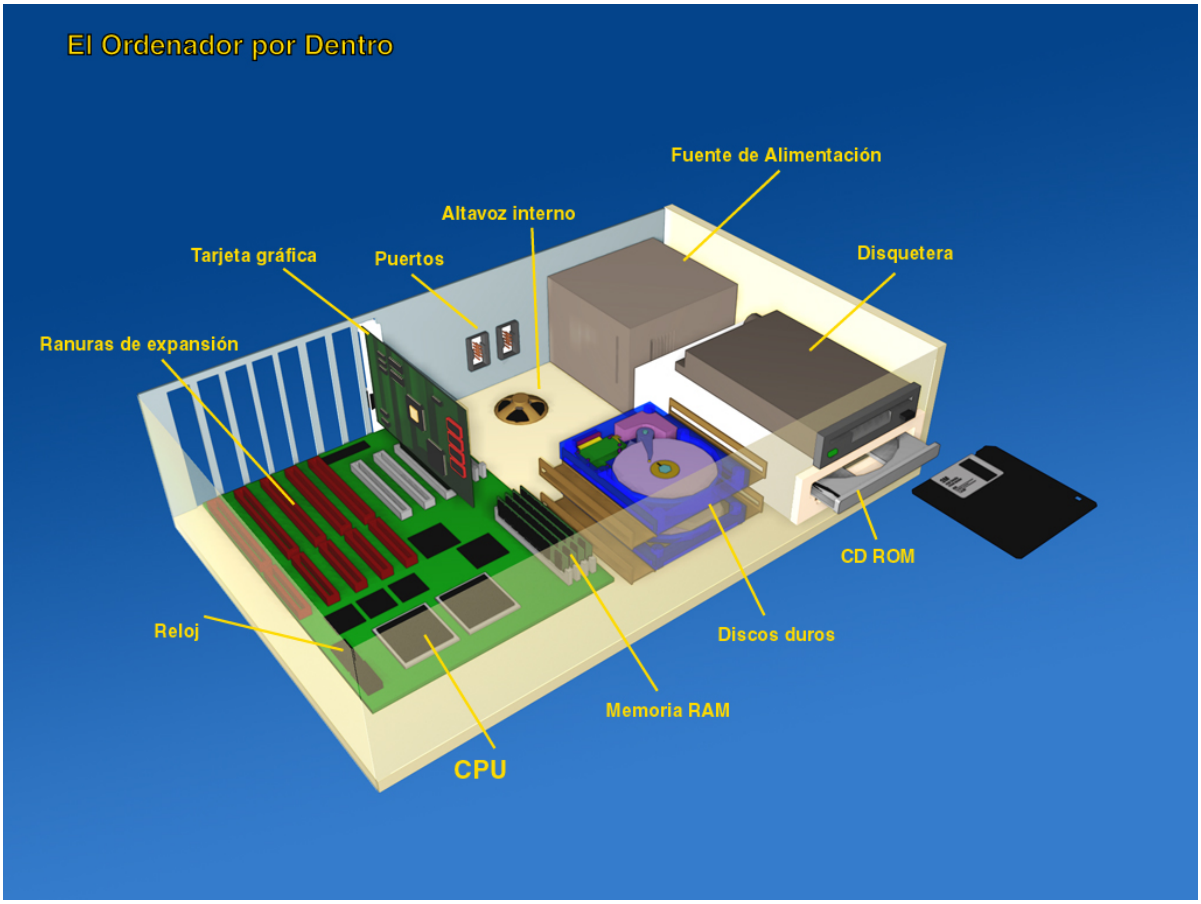

Figura 2. Esquema de un ordenador por dentro (cortesía de Julio Martín).

Las **memorias auxiliares** o **sistemas de almacenamiento masivo** son dispositivos donde se pueden almacenar grandes volúmenes de información (de acceso más infrecuente). Son no volátiles y de acceso más lento que la memoria interna. Su tamaño va desde centenares de Kb a centenares de Gb. Existen de muy diferentes tipos: cintas magnéticas, discos magnéticos, CD-ROM, DVD, discos ópticos...

Los **buses** permiten la transmisión de datos entre las distintas partes del *hardware* mientras que los **dispositivos de entrada y salida** de datos, junto con los **elementos de interconexión**, permiten el intercambio de información con el entorno (incluyendo otros sistemas). Existen distintos tipos: terminales (constan de pantalla y teclado), impresoras, *plotters* o trazadores, *scanners*, tarjetas de red, dispositivos de adquisición de datos,...

Los dispositivos de entrada de datos transforman las informaciones de entrada en señales binarias de naturaleza eléctrica. Un ordenador puede tener diferentes dispositivos de entrada: el teclado, un ratón, un escáner, una lectora de tarjetas de crédito.

Los dispositivos de salida de datos presentan los resultados de los programas ejecutados por el ordenador, transformado las señales eléctricas binarias en imágenes visualizadas o dibujadas o caracteres impresos o reproducidos acústicamente. Ejemplos de dispositivos de salida de datos son: un monitor, una impresora o un plotter.

Otros dispositivos como las tarjetas de red y los modems son, indistintamente, dispositivos de entrada y salida de datos.

Finalmente, las ranuras de expansión permiten la conexión del ordenador con otros dispositivos externos.

1.4. LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es un medio de comunicación entre el usuario y el ordenador formado por un conjunto de símbolos elementales (alfabeto), palabras (vocabulario) y reglas (semántica y sintaxis). Con un lenguaje se construyen cada una de las **órdenes** o **instrucciones** para el ordenador de la secuencia de instrucciones que constituyen un programa.

1.4.1. Clasificación de los lenguajes

Existen muchos lenguajes de programación que pueden clasificarse en tres grandes grupos:

a) **Lenguajes máquina**: son códigos binarios de instrucciones, es decir, sucesiones de ceros y unos (`1100 0010 1010 0101` ...) que componen un repertorio reducido de operaciones. Es el único lenguaje que entienden los circuitos electrónicos que forman la Unidad de Control del procesador y cada procesador tiene un lenguaje máquina específico. La programación con un lenguaje máquina es muy laboriosa para el programador a todos los niveles: escritura, interpretación, depuración y mantenimiento.

b) **Lenguajes ensamblador**: son códigos de instrucciones formados por grupos de palabras mnemotécnicas (`LOAD`=almacenar, `ADD`=sumar, `CMP`=comparar,...). Estos lenguajes sustituyen los códigos binarios por códigos mnemotécnicos mas fáciles de identificar y recordar Cercanos a los lenguajes máquina, también dependen de cada procesador. Los programas no son directamente interpretados por el procesador, necesitándose un programa traductor llamado *ensamblador*. A los lenguajes máquina y ensamblador también se les conoce como **lenguajes de bajo nivel**. Tienen como ventaja el que los programas ocupan poco espacio en memoria y tienen un tiempo mínimo de ejecución. Como inconvenientes principales estarían el que dependen del procesador, es necesario un conocimiento muy profundo del hardware y las operaciones que describen cualquier proceso son muy elementales. Se emplean especialmente en microprocesadores industriales.

c) **Lenguajes de alto nivel**: Son universales: no dependen del procesador ni de la estructura interna de cada ordenador. No se necesita conocer el hardware específico de dicha máquina. Están más próximos al lenguaje natural y son claros, simples, eficientes y legibles. Sus instrucciones equivalen a múltiples instrucciones en lenguaje máquina. Tiene el problema de que los programas son más lentos de ejecución y necesitan ser traducidos a lenguaje máquina para que el procesador los entienda. Como ejemplos de lenguajes de alto nivel pueden destacarse los siguientes: FORTRAN, ALGOL, COBOL, BASIC, PASCAL, C, LISP, JAVA...

1.4.2. Programas traductores

Como los procesadores sólo entienden su lenguaje máquina se necesitan programas para traducir los programas escritos a otros lenguajes a lenguaje máquina. Estos programas traductores pueden clasificarse en:

a) **Ensambladores**: traducen de lenguaje ensamblador a lenguaje máquina.

b) **Compiladores**: traducen las instrucciones de un programa escrito en un lenguaje de alto nivel a un programa escrito en lenguaje máquina. El **programa fuente** es el programa escrito en lenguaje de alto nivel. El **programa objeto** es el programa ya traducido en lenguaje máquina. En muchos casos el resultado de la compilación es sólo una parte o módulo del programa ejecutable que posteriormente construirá el

enlazador o **montador** (*linker*) a base de dichos módulos e incorporando código de las librerías o unidades utilizadas.

c) **Intérpretes**: traducen de un lenguaje de alto nivel a lenguaje máquina pero ejecutando cada instrucción conforme se va traduciendo todo el programa hasta su última instrucción. A diferencia de los compiladores, los intérpretes no generan un programa objeto.

d) **Preprocesadores**: traducen un programa escrito en un lenguaje de alto nivel a otro escrito en otro lenguaje de alto nivel.

1.5. CICLO DE VIDA DEL SOFTWARE

Se entiende por **ciclo de vida** de una aplicación software al periodo que comprende desde el momento en que se decide desarrollar el programa hasta que éste deja de estar en funcionamiento. Un producto o aplicación software trata de resolver un problema o necesidad y está formado por uno o varios programas, junto con sus datos y la documentación asociada. Su desarrollo no se reduce a codificar un programa en un determinado lenguaje de programación. La *ingeniería del software* es la disciplina que estudia los aspectos metodológicos relacionados con el diseño, desarrollo y mantenimiento de programas. En la secuencia de desarrollo de un programa es necesario tener en cuenta las siguientes fases:

a) **Análisis del problema**. Se analiza de forma clara qué problema debe resolver el programa. Se determinan cuáles deben ser los datos de entrada y cuáles son las de salida. Como resultado de esta fase se obtienen dos documentos: la **Definición del Problema** y el **Documento de Requisitos**. En esta fase deben intervenir tanto el usuario final del programa como el informático o grupo de informáticos especialistas.

b) **Diseño y Arquitectura**. En esta fase se decide qué estructuras de datos van a a emplearse (diseño de las estructuras de datos) y cómo se van a ejecutar las acciones (diseño de los procesos). Como resultado se obtendrá el pseudocódigo u organigrama del programa. La responsabilidad de esta fase es de los analistas informáticos junto con los usuarios finales del programa.

c) **Codificación**. Se traducen o *codifican* los algoritmos y estructuras diseñadas anteriormente al lenguaje de programación seleccionado. Como resultado se obtendrá el o los programas fuente que, una vez compilados y enlazados, darán lugar al programa ejecutable.

d) **Verificación y Validación**. Se comprueba que el programa funciona correctamente (verificación) y que cumple las especificaciones de requisitos definidos en la Fase 1 (validación). Para llevar a cabo estos objetivos se realizarán pruebas de cada módulo o unidad, de la integración de todos los módulos y de validación de las especificaciones funcionales.

e) **Instalación y Depuración**. El programa se instala en el sistema informático en el que debe funcionar y se realizan las correcciones necesarias. Cuando es conveniente se elimina el código redundante.

f) **Explotación y Mantenimiento**. Una vez a disposición del usuario, se mantiene un periodo de asistencia técnica o mantenimiento y se solucionan problemas, errores o dudas que aparezcan, se readapta a nuevas especificaciones o circunstancias o, simplemente, se mejora algún aspecto del programa.

g) **Retirada**. Con el paso del tiempo el programa puede quedar obsoleto, eliminándose del sistema en el que fue instalado.

Estas fases pueden desarrollarse de forma secuencial o solapada. Por otro lado, las actividades de planificación y documentación (descripción detallada de los resultados de

cada fase) forman parte de todas ellas. En concreto, la documentación resulta especialmente importante para facilitar el mantenimiento del software.

1.6. PROGRAMAS

Como ya se ha comentado anteriormente, un programa es un conjunto de instrucciones que controla el funcionamiento de un ordenador para llevar a cabo una tarea específica. Los programas pueden clasificarse en programas de **sistema operativo** y programas de **aplicación**.

Un **sistema operativo** está formado por un programa o un conjunto de programas que gestiona el funcionamiento del *hardware*, controla la ejecución de los programas y procesa las órdenes del usuario al ordenador. Ejemplos de sistemas operativos son: *Windows XP*, *Linux*, *MS-DOS*, *UNIX*, *Windows NT*, *Mac/OS*, *VMS*, ... Los sistemas operativos emplean un **lenguaje de control** específico cuyas instrucciones se denominan **comandos**. Por otro lado los sistemas operativos sirven de soporte a los programas de aplicación. El sistema operativo se carga en la memoria RAM y mas concretamente en las direcciones iniciales de memoria.

Las principales funciones de un sistema operativo son:

a) Gestión de disposición de E/S (escritura, lectura, control)
b) Ejecucion de programas (carga, ejecución, interrupciones)
c) Gestion de archivos (creacion, borrado, lectura)
d) Detección de errores (en memoria, en periféricos)
e) Asignación de recursos (CPU, memoria)
f) Protección (confidencialidad, no interferencia)
g) Contabilidad (carga de los recursos)

Los **programas de aplicación** sirven para que el ordenador lleve a cabo una tarea específica. Pueden clasificarse en: paquetes integrados de *software* (*Office*, *Works*,...), editores o procesadores de texto (*Word*, *WordPerfect*,...), bases de datos (*Access*, *Oracle*,...), hojas de cálculo (*Excel*, *Lotus 1-2-3*,...), aplicaciones de diseño gráfico (*AutoCad*, *Microstation*,...), etc.

1.7. CODIFICACIÓN DE LA INFORMACIÓN Y REPRESENTACIÓN DE DATOS

Los datos son abstracciones de la realidad necesarias para la resolución de una tarea. Deben estar representados de forma que sean procesables por un ordenador.Los circuitos electrónicos que componen los ordenadores emplean el formato binario para procesar la información, lo que implica la codificación correspondiente de ésta. Los datos aparecen representados mediante secuencias de ceros y de unos. Si bien el **bit** es la unidad más pequeña de información, el **byte** es la unidad usual de información. Un byte es una secuencia de ocho bits. Como cada uno de estos bits puede tomar dos valores, un byte podrá tomar 2^8 valores distintos. El ordenador puede trabajar con uno o varios bits a la vez, normalmente 8 (1 byte), 16 (2 bytes), 32 (4 bytes) o 64 (8 bytes). Una **palabra** es una cadena finita de bits que puede ser manipulada de forma simultánea como un conjunto por un ordenador en una operación. El **ancho de palabra** es el número de bits de la palabra.

Los **archivos** son estructuras de datos que residen en la memoria secundaria. Son conjuntos de informaciones estructuradas del mismo tipo y en número indeterminado. Pueden ser archivos de datos o de programas.

Tabla 2. Número de bits y posibles combinaciones

Nº de bits	Número de combinaciones
4	$2^4 = 16$
5	$2^5 = 32$
6	$2^6 = 64$
7	$2^7 = 128$
8 (1 byte)	$2^8 = 256$
10	$2^{10} = 1024$
15	$2^{15} = 32768$
16 (2 bytes)	$2^{16} = 65536$
20	$2^{20} = 1048576$
30	$2^{30} = 1073741824$
32 (4 bytes)	$2^{32} = 4294967296$
64 (8 bytes)	$2^{64} = 1.8447 \cdot 10^{19}$

En general, pueden considerarse dos tipos de datos: los datos **numéricos** y los datos **alfanuméricos** o caracteres. Una de las diferencias prácticas entre ellos es que ciertas operaciones están definidas para un tipo de dato y no para el otro. Tanto para codificar datos numéricos como para caracteres se emplean varios formatos alternativos.

1.7.1. Datos numéricos

Dependiendo de la naturaleza de los valores que se quieran representar, los datos numéricos pueden codificarse empleando diferentes formatos.

1.7.1.1 Formato BCD (*Binary Code Decimal*)

Se codifican los diez dígitos del sistema decimal por sus correspondientes representaciones binarias. Se necesitan cuatro bits para cada dígito decimal. Por ejemplo, el número 135 en formato BCD es 0001 0011 0101. El principal inconveniente de este formato es que desaprovecha seis combinaciones del total de posibilidades de representación.

1.7.1.2 Formato hexadecimal

Un dígito hexadecimal, expresado mediante uno de los diez dígitos decimales (0-9) o una de las seis primeras letras del alfabeto (A-F), representa cuatro dígitos binarios. Con cuatro dígitos binarios pueden obtenerse $2^4 = 16$ combinaciones distintas. Este formato es un compromiso razonable entre lo cercano a la máquina y lo práctico para el usuario. Por ejemplo, la secuencia binaria 0100 1010 0010 1111 equivale en formato hexadecimal a 4A2F.

1.7.1.3 Formato de punto fijo para números enteros (sin signo)

Para los enteros sin signo se emplea la representación en base 2 del número natural. Es una representación directa, es decir, cada bit indica el coeficiente de la expresión de dicho número como suma de potencias de base 2. Por ejemplo la siguiente secuencia de 6 bits:

$$100110 = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 32 + 0 + 0 + 4 + 2 + 0 = 38$$

En general, n bits permiten representar 2^n números naturales distintos. El intervalo de representación es de 0 a 2^n-1 Si se quiere transformar un entero decimal a binario, se divide

sucesivamente el valor y los correspondientes cocientes por 2 hasta obtener un cociente nulo. Los restos son los dígitos binarios buscados. El último resto corresponde al bit más significativo (el situado más a la izquierda) y el primer resto corresponde al bit menos significativo (el situado más a la derecha). Por ejemplo, para transformar el valor decimal 38 entero a binario:

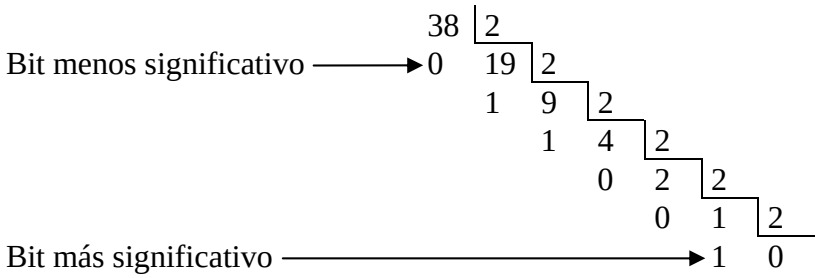

Figura 3. Proceso de conversión de entero decimal a binario

1.7.1.4 Formato de punto fijo para números enteros con signo

El problema surge cuando se quieren representar enteros con signo, ya que éste requiere un bit. Existen diferentes maneras de resolver la codificación de los enteros con signo. Los formatos más importantes son: signo-magnitud, complemento a 2 y exponente desplazado.

En el formato de **signo-magnitud** (SM), el bit situado más a la izquierda (el bit más significativo) representa el signo del entero (0 para positivo, 1 para negativo). Los demás bits representan su valor absoluto. El intervalo de representación es $-2^{n-1}+1 <= x <= 2^{n-1}-1$.

Por ejemplo:

$$100110 = -(0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0) = -(0+0+4+2+0) = -6$$

$$000110 = 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 0+0+4+2+0 = 6$$

El formato de **complemento a 2** (C2) es el habitualmente utilizado en los ordenadores para representar valores numéricos enteros con signo. El bit más a la izquierda representa el valor -2^{n-1} siendo *n* el número total de bits empleado. En este caso, el intervalo de representación es $-2^{n-1} <= x <= 2^{n-1}-1$. Por ejemplo:

$$000110 = 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 0+0+0+4+2+0 = 6$$

$$100110 = -1\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = -32+0+0+4+2+0 = -26$$

El valor opuesto de un entero representado en complemento a 2 se puede obtener cambiando ceros por unos y viceversa y sumando uno a la expresión en binario que resulta. Por ejemplo, partiendo que la secuencia 000110 representa el valor 6, si cambiamos los ceros por unos y viceversa (111001) y sumamos 1, obtenemos el 111010, que representa el valor opuesto –6.

$$111010 = -1\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = -32+16+8+0+2+0 = -6$$

El formato del **exponente desplazado** o **sesgado** considera el signo restando un valor constante al valor entero absoluto representado por una secuencia binaria. El valor constante restado suele ser 2^{n-1}, siendo n el número de bits de la secuencia. Por ejemplo:

$$100110 = 1\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 - 2^5 = 32 + 0 + 0 + 4 + 2 + 0 - 32 = 6$$

$$000110 = 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 - 2^5 = 0 + 0 + 0 + 4 + 2 + 0 - 32 = -26$$

Tabla 3. Conversión de los principales formatos para la representación de números enteros

Binario (4 bits)	BCD	Hexadecimal	Sin signo	Signo - Magnitud	Complemento a 2	Exponente Desplazado
0000	0	0	0	0	0	-8
0001	1	1	1	1	1	-7
0010	2	2	2	2	2	-6
0011	3	3	3	3	3	-5
0100	4	4	4	4	4	-4
0101	5	5	5	5	5	-3
0110	6	6	6	6	6	-2
0111	7	7	7	7	7	-1
1000	8	8	8	0	-8	0
1001	9	9	9	-1	-7	1
1010	-	A	10	-2	-6	2
1011	-	B	11	-3	-5	3
1100	-	C	12	-4	-4	4
1101	-	D	13	-5	-3	5
1110	-	E	14	-6	-2	6
1111	-	F	15	-7	-1	7

1.7.1.5 Formato de punto fijo para fracciones

El formato de punto o coma fija para fracciones consiste en la expresión del número como suma de potencias positivas y negativas de 2 y en el que el punto que separa las potencias negativas está en una posición determinada entre dos dígitos binarios. Por ejemplo:

$$1001{,}101 = 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = 8 + 1 + 0.5 + 0.125 = 9.625$$

Para números negativos menores que la unidad también se pueden emplear los formatos de signo-magnitud y de complemento a dos. En estos casos, si el bit más significativo vale 1, se toma, respectivamente, como valor de signo negativo o como el coeficiente del sumando -2^{m-1}, siendo m el número de bits a la izquierda de la coma. Por ejemplo:

$$\text{(SM)}\quad 0001{,}101 = 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = 1 + 0.5 + 0.125 = 1.625$$

$$\text{(SM)}\quad 1001{,}101 = -\left(0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3}\right) = -\left(1 + 0.5 + 0.125\right) = -1.625$$

$$\text{(C2)}\quad 0001{,}101 = 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = 1 + 0.5 + 0.125 = 1.625$$

$$\text{(C2)}\quad 1001{,}101 = -1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = -8 + 1 + 0.5 + 0.125 = -6.375$$

Para pasar un valor fraccionario decimal a binario se multiplica sucesivamente la parte fraccionaria del valor de partida y de los resultados obtenidos en los productos por 2.

El valor en binario se forma con las partes enteras (cero o unos) de los productos obtenidos. Por ejemplo:

0.15625	0.3125	0.625	0.25	0.5
x 2	x 2	x 2	x 2	x 2
0.3125	0.625	1.25	0.5	1.0

0.15625 = ,00101

0.6745	0.349	0.698	0.396	0.792	0.584	0.168	0.336
x 2	x 2	x 2	x 2	x 2	x 2	x 2	x 2
1.349	0.698	1.396	0.792	1.584	1.168	0.336	0.672

0.6745 = ,10101100...

La precisión está determinada por el número de dígitos empleado.

1.7.1.6 Formato de punto o coma flotante

Es la solución al problema del aumento geométrico del número de bits necesario para representar valores enteros o reales mayores. Una solución a este problema consiste en la utilización de la notación científica o notación de punto o coma flotante. Todo valor numérico se puede representar por una mantisa, una base y un exponente. Utilizando una base decimal:

$$54016 = 0.54016 \cdot 10^{5} = 0.54016\text{E} + 05$$

$$0.00374 = 0.374 \cdot 10^{-2} = 0.374\text{E} - 02$$

En el último formato, la letra E sustituye a la base 10 de la potencia. Pero, como ya se ha dicho los ordenadores emplean una base binaria, lo que queda implícito al codificar cualquier valor numérico mediante una mantisa y un exponente expresados como potencias negativas y positivas de 2. La mantisa puede representarse empleando un formato en punto fijo, mientras que para el exponente se puede utilizar cualquier formato de representación de un entero. Por ejemplo:

$$54016 = 2^{15} + 2^{14} + 2^{12} + 2^{9} + 2^{8} = (2^{-1} + 2^{-2} + 2^{-4} + 2^{-7} + 2^{-8}) \cdot 2^{16}$$

se convierte en formato binario en 011010011 (mantisa) y 010000 (exponente)

$$0.0032501 = 2^{-9} + 2^{-10} + 2^{-12} + 2^{-14} + 2^{-16} = (2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8}) \cdot 2^{-8}$$

se convierte en formato binario en 011010101 (mantisa) y 111000 (exponente)

Se dice que un valor binario en coma flotante está *normalizado* si el valor absoluto de la mantisa, m, cumple la siguiente condición: $0.5 \leq | m | < 1$. El valor binario del ejemplo anterior está normalizado ya que el valor numérico de la mantisa es 0.83203. Por el contrario, las secuencias binarias 001101010 111001 ó 100111010 010010 representan valores numéricos no normalizados.

Es muy habitual el empleo del formato en coma flotante con exponente desplazado en la que al valor entero del exponente tomado como entero positivo se le resta siempre 2^{n-1} siendo n el número de bits utilizado para la codificación del mismo exponente.

La precisión también está determinada por el número de dígitos empleados. Como se verá en la siguiente sección, los principales factores para seleccionar una codificación específica dependen del coste de la traducción, el coste del almacenamiento y el coste del procesamiento.

Las operaciones realizadas con este tipo de datos son relativamente lentas y se llevan a cabo empleando código o subrutinas específicas (*software*) o coprocesadores matemáticos (*hardware*).

1.7.2. Operaciones aritméticas

Para realizar operaciones con datos numéricos en formato binario hay que utilizar la aritmética binaria teniendo en cuenta, además, la limitación del número de bits para representar operandos y resultados. Este último punto determina la existencia de un intervalo de representación y una precisión para los datos numéricos a manipular. La forma en la que se realicen las operaciones dependerá del formato de representación de los valores numéricos.

Cualquiera que sea la forma de representación y el número de bits empleado hay un límite en el rango o intervalo de números que pueden representarse. Cuando el resultado de una operación excede de este intervalo se produce un desbordamiento u *overflow*.

Pero, entonces ¿por qué no se emplean, por ejemplo 100 bits o una cantidad incluso mayor, para codificar cualquier dato numérico? Sencillamente por tres motivos: el coste de la codificación (pruébese a codificar -45 en complemento a dos con 100 bits), el coste de almacenamiento (mayores necesidades en la capacidad de almacenamiento y ¡la memoria en un ordenador sale muy cara!) y el coste del tratamiento de la información (las operaciones aritméticas se complican).

Asimismo, esta limitación también incide en la precisión o resolución de representación. Si se están utilizando valores fraccionarios, a mayor número de bits mayor precisión. De nuevo aparece el problema de utilizar un número grande de dígitos para representar un valor.

Por otro lado, algunas operaciones se realizan en términos de otras. Las multiplicaciones se llevan a cabo mediante desplazamientos a la izquierda y sumas. Las divisiones mediante desplazamientos a la derecha y restas. El resultado puede tener un bit más que los sumandos. Si se excede del número de bits disponibles se produce en rebosamiento, desbordamiento u *overflow*. Por ejemplo, si se utilizan cuatro bits para representar valores numéricos enteros positivos y se intenta sumar los valores 6 (0110) y 12 (1100), el resultado será 2 (0010).

1.7.3. Datos alfanuméricos (caracteres)

Básicamente, los ordenadores solo trabajan con números. Pueden almacenar letras y otros caracteres asociandoles un número a cada uno de ellos. El carácter es la unidad de información a nivel del alfabeto humano. Hay caracteres alfabéticos, numéricos (dígitos decimales) y especiales (signos de puntuación, etcétera). En un principio lo habitual era emplear secuencias de seis, siete u ocho caracteres para representar los distintos caracteres. El formato binario más utilizado para la codificación de caracteres es el formato ASCII (correspondiente a las siglas de *American Standard Code for Information Interchange*). Utiliza 7 bits para 128 caracteres. Permite representar cuatro tipos de caracteres:

a) Caracteres alfabéticos, tanto para las mayúsculas como para las minúsculas: A, B, C,... Z, a, b, c,... z
b) Caracteres numéricos, es decir, los dígitos decimales: 0, 1, 2, 3,... 9
c) Caracteres especiales y de puntuación: incluyen algunos signos de puntuación : & #, *
d) Caracteres de control.

Posteriormente se amplía a 8 bits (ASCII extendido, ampliado o de 8 bits). El conjunto de caracteres que puede codificarse en un ordenador o que puede utilizar un

lenguaje es el *juego de caracteres del ordenador* o *del lenguaje*. Durante muchos años se empleó el código EBCDIC (*Extended Binary Coded Decimal Interchange Code*), diseñado por IBM en 1964. En el futuro probablemente se empleará el código UNICODE[1], que es un sistema de codificación de caracteres de 2 bytes (16 bits) y que permite la representación de 65536 caracteres diferentes.

1.8. LÓGICA BOOLEANA

La **lógica booleana** desarrollada por el matemático inglés George Boole (1810–1864) en 1847 es un conjunto de operaciones que manipulan datos o variables booleanas. Un dato o variable booleana sólo puede tener dos valores o estados: verdadero/falso, alto/bajo, 1/0,... El tratamiento último de los datos en la Unidad Aritmético-Lógica del procesador de un ordenador se lleva a cabo en términos de variables booleanas.

Las **operaciones lógicas** operan sobre datos o variables booleanas produciendo otro dato o variable booleana. Cada operación lógica se caracteriza por una tabla de la operación o **tabla de verdad**, que proporciona el valor de la variable resultante en función de todos los valores posibles de las variables de entrada. Cada operación tiene dos símbolos: el del circuito lógico o **puerta lógica** que realiza la operación y el del **álgebra booleana**. La puerta lógica es el dispositivo que realiza la operación lógica.

Las operaciones lógicas básicas son:

NOT (Negación): Tiene una variable de entrada y otra de salida. El valor de la variable de salida es el opuesto al de la variable de entrada. La operación se expresa como Q = ¬ P y la tabla de verdad de esta operación se muestra en la Tabla 1.4.

Tabla 4. Tabla de verdad de la negación lógica

P	Q = ¬ P
0	1
1	0

AND (Producto o conjunción lógica): Tiene dos variables de entrada y una de salida. La de salida es 1 si todas las variables de entrada son 1. La operación se expresa como R = P · Q y la tabla de verdad de esta operación se muestra en la Tabla 1.5.

Tabla 5. Tabla de verdad del producto lógico

P	Q	R = P · Q
0	0	0
0	1	0
1	0	0
1	1	1

La tabla de verdad para la operación AND puede extenderse para tres o más variables siguiendo las reglas establecidas.

OR (Suma o disyunción lógica): Tiene dos variables de entrada y una de salida. La de salida es 1 si alguna de las variables de entrada es 1. La operación se expresa como R = P + Q y la tabla de verdad de esta operación se muestra en la Tabla 1.6.

[1] http://www.unicode.org

Tabla 6. Tabla de verdad de la suma lógica

P	Q	R = P + Q
0	0	0
0	1	1
1	0	1
1	1	1

XOR (Suma exclusiva): Tiene dos variables de entrada y una de salida. La de salida es 1 si únicamente una de las variables de entrada es 1. La operación se expresa como R = P ⊕ Q y la tabla de verdad de esta operación se muestra en la Tabla 1.7.

Tabla 7. Tabla de verdad de la suma exclusiva lógica

P	Q	R = P ⊕ Q
0	0	0
0	1	1
1	0	1
1	1	0

Puede demostrarse que cualquier operación lógica con cualquier número de entradas puede realizarse mediante combinaciones de puertas NOT y AND.

Bibliografía básica

- **Bishop, P.** *Fundamentos de Informática*, Anaya Multimedia, Madrid, 1992
- **García-Beltrán, A., Martínez, R.** y **Jaén, J.A.** *Métodos Informáticos en TurboPascal*, Ed. Bellisco, 2ª edición, Madrid, 2002
- **Glenn Brookshear, J.** *Computer Science*, The Benjaming-Cummings Publishing Company Inc, 1985
- **Pressman, R.S.** *Ingeniería del Software – Un enfoque práctico*, Mc. Graw-Hill, cuarta edición, 1997
- **Ureña, L.A., Sánchez, A.M., Martín, M.T. y Mantas, J.M**. *Fundamentos de Informática*, RA-MA, 1997
- **Prieto, A., Lloris, A. y Torres, J.C**., Introducción a la Informatica, Editorial McGraw-Hill, 2001
- **De Mora, C.** y otros, Estructura y Tecnología de Computadores I, Editorial UNED, 2002