Tema 1

Introducción a los ordenadores

Versión: 22 de septiembre de 2012

Las palabras españolas **informática** y **ordenador** provienen de las francesas **informatique** y **ordinateur**. La palabra francesa **informatique** se construyó como una contracción de las palabras **INFORmation** y **autoMATIQUE**.

La palabra **computador** (o **computadora**, mayoritariamente utilizada en Sudamérica), proviene del término inglés **computer**. La ciencia que se ocupa de los ordenadores se denomina, en inglés, **Computer Science**, aunque también existe la palabra **Informatics**.

1.1 Componentes básicos de un ordenador

En un ordenador se distinguen principalmente dos aspectos:

- **HARDWARE**, el soporte físico, no modificable (placas, circuitos integrados, chips, módulos, cables, etc.), es decir la "maquinaria".
- **SOFTWARE**, el conjunto de programas que se ejecutan en el ordenador, *grosso modo* divididos en:
  - Software del sistema o sistema operativo, el conjunto de programas necesarios para que el ordenador tenga capacidad de trabajar (funcionamiento de la pantalla, del teclado, movimientos del ratón, etc.).
  - Software de aplicación que son los programas que maneja el usuario (tratamiento de textos, hojas de cálculo, bases de datos, etc).

El modelo básico de los ordenadores actuales se atribuye a Von Neumann[1]. Consta principalmente de la unidad central de proceso, la memoria y los dispositivos de entrada/salida, conectados entre sí como se muestra de forma esquemática en el diagrama mostrado en la Figura 1.1.

1.2 Unidad central de proceso

Unidad central de proceso o **CPU** (**C**entral **P**rocess **U**nit) es el "cerebro" del ordenador: ejecuta las instrucciones de los programas y controla el funcionamiento de los distintos componentes del ordenador. Suele estar integrada en un chip denominado *microprocesador*. Los dos componentes más importantes de la CPU son:

[1] John von Neumann (1903-1957), matemático húngaro-estadounidense, pionero de los modernos ordenadores, que en un informe sobre el ordenador EDVAC, propuso el concepto de programa almacenado en la memoria del ordenador, junto con los datos.

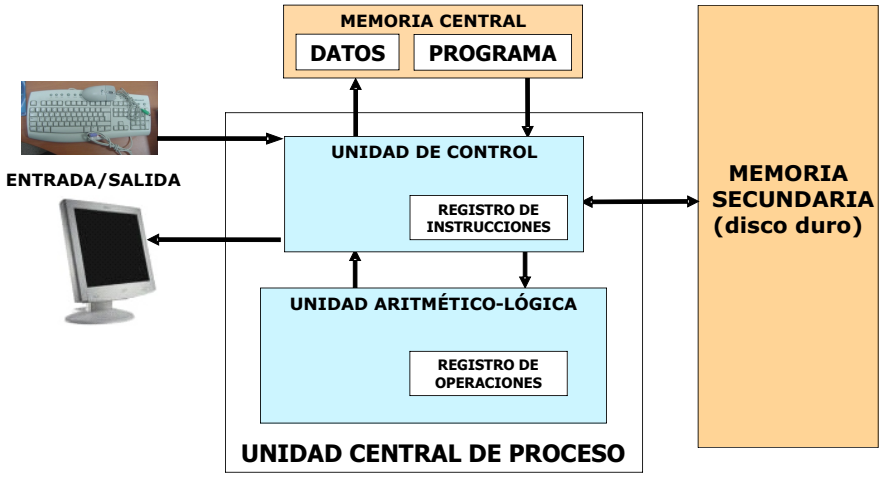


Figura 1.1: Componentes básicos de un ordenador

- La **unidad de control** o **CU** (**C**ontrol **U**nit), dirige y coordina la mayoría de las operaciones del ordenador. Interpreta cada instrucción enviada por un programa e inicia la acción apropiada para realizar esa instrucción. Para cada instrucción, la unidad de control repite un grupo de cuatro operaciones básicas, que constituye lo que se llama **ciclo de instrucción**:

  1. Leer ("fetch") - es el proceso de obtener una instrucción de un programa o datos de la memoria.
  2. Decodificar ("decode") - es el proceso de traducir la instrucción en comandos que el ordenador pueda ejecutar.
  3. Ejecutar ("execute") - es el proceso de llevar a cabo los comandos.
  4. Escribir ("writeback") - es el proceso de almacenar el resultado del paso de ejecución, "escribiéndolo" en la memoria.

- La **unidad aritmético-lógica** o **ALU** (**A**rithmetic **L**ogic **U**nit), que realiza las operaciones elementales que constituyen el programa (sumar, multiplicar, comparar, etc).

La velocidad de un procesador se suele medir en **hercios (Hz)**: número de operaciones por segundo que puede realizar.

$$\begin{array}{lll} \textbf{1 Megahercio (Mz)} & = 10^6 & \text{operaciones/segundo} \\ \textbf{1 Gigahercio (Gz)} & = 10^9 & \text{operaciones/segundo} \end{array}$$

1.3 Memoria

Son los componentes de hardware en los que se almacena la información procesada por el ordenador. Generalmente, se distinguen entre la **memoria central** y la **memoria secundaria** (véase más adelante). La principal característica de las memorias es su:

- **Capacidad**: indica la cantidad de datos que puede almacenar. Se mide en bits **bit** (**bi**nary uni**t**), o múltiplos del bit. Los bits suelen agruparse de ocho en ocho: **1 byte**= 8 bits. Las unidades de almacenamiento están recogidas en la Tabla 1.1.

bit	0 ó 1
byte (B)	8 bits
Kilobyte (KB)	2^{10} bytes = 1024 bytes
Megabyte (MB)	2^{10} KB = 1024 KB
Gigabyte (GB)	2^{10} MB = 1024 MB
Terabyte (TB)	2^{10} GB = 1024 GB

Tabla 1.1: Unidades de almacenamiento de información

1.3.1 Memoria central

Memoria central o principal, almacena tanto la secuencia de instrucciones del programa o programas que se esté ejecutando en cada momento como los datos que éste o éstos necesitan. Se distingue entre:

- Memoria **ROM** (**R**ead **O**nly **M**emory), (memoria de sólo lectura). Es permanente, esto es no se borra al apagar el ordenador y no se puede alterar (o se puede hacer difícilmente). Almacena códigos de programa grabados en fábrica necesarios para el funcionamiento del ordenador, como por ejemplo, la secuencia de instrucciones que hay que ejecutar cuando se enciende el ordenador (BIOS: Basic Input/Output System; POST: Power On Self Test).

- Memoria **RAM** (**R**andom **A**cces **M**emory), (memoria de acceso aleatorio). Almacena las instrucciones de los programas mientras que se ejecutan así como los datos que éstos necesitan o generen. Su contenido es volátil, es decir, se borra al apagar el ordenador. La denominación Acceso Aleatorio es antigua y se comenzó a utilizar para diferenciarla de otro tipo de memoria (cintas magnéticas) que era de acceso secuencial: para acceder a un dato había que recorrer todos los anteriores.

Con el objetivo de que el procesador pueda obtener los datos de la memoria central más rápidamente, la mayoría de los procesadores usan un tipo especial de memoria llamada **memoria caché** ó **RAM caché** (pequeña y de acceso muy rápido). En ella, además se almacenan los datos más recientes y las instrucciones inminentes.

La memoria central es un circuito que se puede imaginar como una enorme tabla que almacena información en cada una de sus **celdas** o **posiciones de memoria** (véase Figura 1.2).

Cada celda es una agrupación de bytes que se denomina **palabra**. Dependiendo del ordenador, existen palabras de 4 bytes (32 bits), 8 bytes (64 bits). Las palabras están numeradas correlativamente, comenzando desde 0. El número de una palabra es la **dirección** de esa palabra. La información almacenada en una palabra es su **contenido**.

1.3.2 Memoria secundaria

Como hemos mencionado, toda la información en la memoria central es accesible directamente en un tiempo muy corto; por ello dicha memoria es cara.

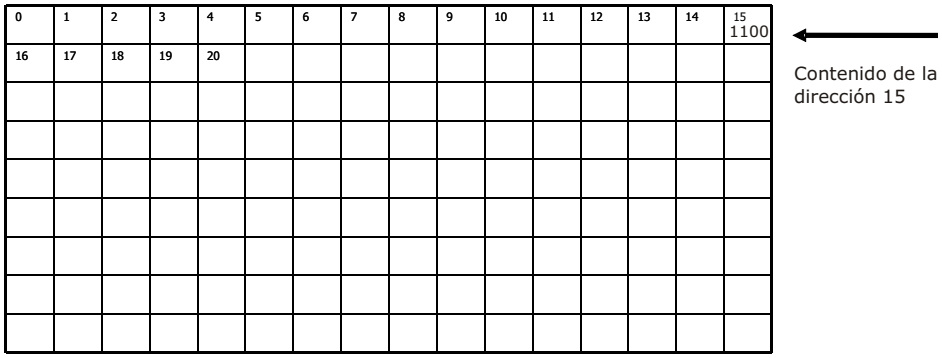


Figura 1.2: Memoria central de un ordenador

Memoria secundaria es una memoria más barata donde, a cambio de mayores tiempos de acceso, se puede conservar gran cantidad de información (disco duro, CD (Compact Disk), DVD (Digital Versatile Disk), memoria flash, etc). En ella se almacena todo lo que la unidad central de procesos (véase la Sección 1.2) no necesita urgentemente. La principal característica de este tipo de memorias es que su contenido no se pierde al apagar el ordenador.

1.4 Dispositivos de entrada/salida

Dispositivos de **entrada/salida** son todos los componentes de hardware anexos que permiten la comunicación entre el ordenador y el usuario. Los dispositivos de entrada son, por ejemplo, teclados, ratones, lectores de discos, micrófonos, etc. Los dispositivos de salida son, por ejemplo, monitores, impresoras, etc.

1.5 Sistema operativo

El **sistema operativo** es el software más importante que se ejecuta en un ordenador. Se trata de un conjunto de mecanismos y reglas básicas de funcionamiento que hace posible una utilización eficiente de los recursos de un ordenador. Ejemplos de sistemas operativos son WINDOWS, MacOS, UNIX, Linux. Se compone de un paquete integrado de programas de dos tipos:

- **Programas de control**: gestionan el hardware y software, por ejemplo colas de impresión etc.
- **Utilidades del sistema**: programas de ayuda al usuario, por ejemplo editores de texto, gestión de correo, etc.

1.6 Lenguajes

Un **lenguaje de programación** es un conjunto de símbolos y reglas sintácticas y semánticas que sirven para describir las órdenes que controlan el comportamiento físico y lógico de una máquina.

Los procesadores de las máquinas sólo son capaces de entender y obedecer programas escritos en **lenguaje-máquina**, cuyas instrucciones son cadenas binarias (formadas por 0 y 1) que se pueden

"cargar" directamente en la memoria central, sin necesidad de traducción. Sin embargo, el lenguaje-máquina es específico de cada tipo de ordenador, por lo que los programas escritos en dicho lenguaje no son portables en general de una máquina a otra. Además, debido a su representación totalmente numérica, son muy difíciles de escribir, leer y corregir.

El lenguaje **ensamblador** facilita (sólo un poco) esas tareas, ya que permite la escritura de las instrucciones básicas "entendibles" por la CPU en una forma más legible para el programador. [2]

A modo de ejemplo, una instrucción básica ejecutable por la CPU podría ser:

```
copiar el contenido de la celda 164 en la celda 183.
```

En lenguaje-máquina esta instrucción podría escribirse (sólo es un ejemplo)

$$\underbrace{01101100}_{\texttt{COPIAR}} \quad \underbrace{10100100}_{\texttt{164}} \quad \underbrace{10110101}_{\texttt{183}}$$

mientras que en lenguaje ensamblador se podría escribir de forma parecida a:

```
CP   164   183
```

Usualmente hay una correspondencia uno a uno entre las instrucciones simples de un código ensamblador y las de un código máquina. Por ello, el ensamblador sigue siendo un lenguaje de *bajo nivel*: por un lado el programador necesita conocer en profundidad la arquitectura de la máquina, por otro los programas escritos en ensamblador no son portables.

Los **lenguajes de alto nivel**, por el contrario, no obligan al programador a conocer los detalles del ordenador que utiliza. Las instrucciones se escriben en un formato flexible y más "humano". Además, se pueden escribir, de forma sencilla, instrucciones mucho más complicadas: cada instrucción en un lenguaje de alto nivel corresponde a varias (incluso muchas) de lenguaje-máquina. La instrucción del los ejemplos anteriores, se podría escribir:

$$A = B$$

El ordenador sólo "comprende" los programas escritos en lenguaje-máquina. Cualquier otro debe ser **traducido**.

La traducción de un programa escrito en ensamblador a lenguaje-máquina la realiza un programa específico denominado también **ensamblador**.

Un programa escrito en un lenguaje de alto nivel se denomina **programa fuente**. Para traducirlo al lenguaje-máquina se puede usar:

- Un **compilador**: es un programa informático que traduce un programa fuente a un programa equivalente escrito en lenguaje-máquina, que se denomina **programa objeto**. El programa objeto puede ser almacenado como archivo en la memoria secundaria del ordenador para ser ejecutado posteriormente sin necesidad de volver a realizar la traducción.
- Un **intérprete**: traduce el código fuente instrucción a instrucción y la ejecuta en el instante. No se crea un archivo o programa objeto, de modo que hay que volver a traducir cada vez que usemos el programa fuente correspondiente.

[2] Fue usado ampliamente en el pasado para el desarrollo de software, pero actualmente sólo se utiliza en contadas ocasiones, especialmente cuando se requiere la manipulación directa del hardware o se pretenden rendimientos inusuales del los equipos.

1.7 Representación de números en el ordenador

El **bit** es la unidad mínima de información empleada en informática o en cualquier dispositivo digital. Un bit es un (diminuto) dispositivo electrónico capaz de tener dos estados: "apagado", que se asimila al cero y "encendido", que se asimila al uno.

Así, con un bit, sólo pueden representarse dos valores. Para representar más cantidad de valores es necesario usar más bits. Si se usan 2 bits se pueden representar $4 = 2^2$ valores distintos: 00, 01, 10, 11. Si se usan 4 bits se pueden representar $16 = 2^4$ valores distintos: 0000, 0001, 0010, 0011, ...etc. En general, con n bits se pueden representar 2^n valores distintos.

Los bits se suelen agrupar en grupos de 8, formando un **byte**. Para almacenar datos, a su vez, se suelen considerar agrupaciones de 4 u 8 bytes (32 ó 64 bits), a las que se llama **palabra**. Estas agrupaciones determinan el conjunto de valores que se pueden almacenar en ellas: en 4 bytes (32 bits) se pueden almacenar 2^{32}=4.294$_1$967.296 valores distintos, mientras que en 8 bytes (64 bits) se pueden almacenar $2^{64} > 18 \times 10^{18}$ valores distintos.

Como consecuencia de lo anterior es claro que el tamaño de la palabra determina el cardinal (número de elementos) del conjunto de datos de un determinado tipo. Por ejemplo, no se podrán almacenar en el ordenador más de 2^{32} números enteros distintos: **el conjunto de números con los que se puede trabajar en un ordenador es finito**. A los números (enteros o reales) que sí se pueden representar en el ordenador se les suele llamar **números-máquina**.

Debido a que cada bit sólo dispone de dos estados posibles, los ordenadores utilizan para representar datos numéricos el **sistema binario** de numeración, en el que sólo hay dos **dígitos**: **0** y **1**. Esto significa, básicamente, que los números se representan en el ordenador en su expresión **en base 2**. La forma concreta de codificarlos depende del tipo de dato: entero, real,...

1.7.1 Representación de números enteros

Para un número entero N se tiene:

- **Representación decimal:**

$$\begin{aligned} N_{(10)} &= a_k\, a_{k-1} \ldots a_1\, a_0 \\ &= a_k \cdot 10^k + a_{k-1} \cdot 10^{k-1} + \cdots + a_1 \cdot 10^1 + a_0 \cdot 10^0, \quad a_i \in \{0, 1, 2, \ldots, 9\} \quad \forall i \end{aligned}$$

EJEMPLO:
$N = 3459 = 3 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0$

- **Representación binaria:**

$$\begin{aligned} N_{(2)} &= b_q\, b_{q-1} \ldots b_1\, b_0 \\ &= b_q \cdot 2^q + b_{q-1} \cdot 2^{q-1} + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0, \quad b_i \in \{0, 1\} \quad \forall i \end{aligned}$$

EJEMPLO:
$N_{(2)} = 10111 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 23_{(10)}$

Para pasar de un número entero N del sistema decimal a binario se dividen sucesivamente dicho número y los cocientes sucesivos por 2, hasta llegar a un cociente igual a 1. Entonces se tiene $N_{(2)} = C_n R_n R_{n-1} \dots R_2 R_1$, con $C_n = 1$, siendo R_k el resto de la k-ésima división, como se muestra en la Figura 1.3.

:2	N	
	C_1	R_1
	C_2	R_2
	C_3	R_3
	...	...
	C_n	R_n

Figura 1.3: Representación binaria de un número entero N.

EJEMPLO:

$N = 77_{(10)} = 1001101_{(2)}$

:2	77	
	38	1
	19	0
	9	1
	4	1
	2	0
	1	0

1.7.2 Representación de números reales

La parte entera de un número real se representa en base 2 igual que cualquier número entero. Para la parte fraccionaria se tiene:

- **Representación decimal:**

$$\begin{aligned} R_{(10)} &= 0.d_1\, d_2\, d_3\, \dots\, d_n \dots \\ &= d_1 \cdot 10^{-1} + d_2 \cdot 10^{-2} + \dots + d_n \cdot 10^{-n} + \dots \\ &= 0.d_1 + 0.0d_2 + \dots + 0.00 \dots d_n + \dots\,, \quad d_i \in \{0, 1, \dots, 9\} \quad \forall i \end{aligned}$$

EJEMPLO:

$R_{(10)} = 0.325 = 3 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$

- **Representación binaria:**

$$\begin{aligned} R_{(2)} &= 0.c_1c_2c_3\ldots c_nc_{n+1}\ldots \\ &= c_1\cdot 2^{-1}+c_2\cdot 2^{-2}+\cdots+c_n\cdot 2^{-n}+\ldots, \quad c_i\in\{0,1\} \quad \forall i \end{aligned}$$

EJEMPLO:

$R_{(2)} = 0.111 = 1\cdot 2^{-1}+1\cdot 2^{-2}+1\cdot 2^{-3} = 0.875_{(10)}$

El procedimiento (iterativo) para obtener la representación binaria de un número con parte fraccionaria es:

(a) Poner $R_0 = R$. Calcular $2R_0$ y tomar $c_1 = [2R_0]$ (parte entera de $2R_0$).

(b) Poner $R_1 = 2R_0 - c_1$. Calcular $2R_1$ y tomar $c_2 = [2R_1]$ (parte entera de $2R_1$).

(c) ...

(d) Poner $R_k = 2R_{k-1} - c_k$. Calcular $2R_k$ y tomar $c_{k+1} = [2R_k]$ (parte entera de $2R_k$), $\quad k \geq 1$.

EJEMPLO:

$R = 0.23_{(10)} = 0.0011101\ldots{}_{(2)}$

$$\begin{array}{lll} R_0 = 0.23\,; & 2R_0 = 0.46\,; & c_1 = 0 \\ R_1 = 2R_0 - c_1 = 0.46\,; & 2R_1 = 0.92\,; & c_2 = 0 \\ R_2 = 2R_1 - c_2 = 0.92\,; & 2R_2 = 1.84\,; & c_3 = 1 \\ R_3 = 2R_2 - c_3 = 0.84\,; & 2R_3 = 1.68\,; & c_4 = 1 \\ R_4 = 2R_3 - c_4 = 0.68\,; & 2R_4 = 1.36\,; & c_5 = 1 \\ R_5 = 2R_4 - c_5 = 0.36\,; & 2R_5 = 0.62\,; & c_6 = 0 \\ R_6 = 2R_5 - c_6 = 0.62\,; & 2R_6 = 1.24\,; & c_7 = 1 \end{array}$$

Obsérvese en el ejemplo cómo un número decimal con parte fraccionaria finita en base 10 puede tener una parte fraccionaria infinita en base 2.

Lo contrario no es verdad: si $0.c_1c_2c_3\ldots c_n$ es la expresión en base 2 de un número fraccionario, se tiene:

$$0.c_1c_2c_3\ldots c_n = c_1\cdot 2^{-1}+c_2\cdot 2^{-2}+\cdots+c_n\cdot 2^{-n} = \frac{c_1}{2}+\frac{c_2}{2^2}+\cdots+\frac{c_n}{2^n}$$

donde los c_i valen 0 ó 1. Por tanto la expresión anterior no es más que una suma **finita** de términos de la forma $\frac{1}{2^k} = (\frac{1}{2})^k = 0.5^k$, que tiene siempre un número finito de cifras decimales.

1.7.3 Almacenamiento de números enteros

Los números enteros se almacenan en palabras de 32 bits, distribuidos de la siguiente manera [3]:

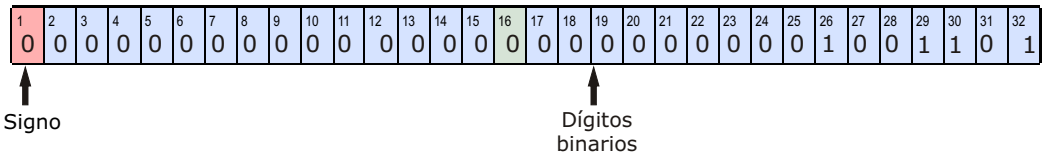
- 1 bit para el signo del número (0 si es positivo, 1 si es negativo)

[3] Esta explicación está simplificada. La codificación interna real es algo más complicada.

- los 31 bits restantes son para almacenar los dígitos binarios del número sin signo.

EJEMPLO:

$N = 77_{(10)} = 1001101_{(2)}$



En consecuencia, se tiene lo siguiente:

- El mayor entero positivo que se puede almacenar con este sistema es:

$$\underbrace{11\ldots11}_{31} = 2^{30} + 2^{29} + \cdots + 2^1 + 2^0 = 2^{31} - 1 = 2.147.483.647$$

- El menor entero negativo es: $-2.147.483.647$.
- El total de números enteros distintos representables: (doble representación para el cero) es:

$$2 \times 2^{31} - 1 = 2^{32} - 1 = 4.294.967.295.$$

1.7.4 Almacenamiento de números reales

Los números reales, habitualmente se representan en **coma flotante** (en inglés, *floating point*). Se trata de un modo "normalizado" de representación de números con parte fraccionaria que se adapta al orden de magnitud de los mismos, permitiendo así ampliar el rango de números representables.

Consiste en trasladar la coma (o punto) decimal hasta la posición de la primera cifra significativa (la primera, comenzando por la izquierda, que no es nula) a la vez que se multiplica el número por la potencia adecuada de la base de numeración.

EJEMPLO:
$17.02625 = 0.1702625 \times 10^2$ (en numeración decimal)
$0.0000000234 = 0.234 \times 10^{-7}$ (en numeración decimal)
$101.11000101 = 0.10111000101 \times 2^3$ (en numeración en base 2)

Sea, pues, un número R dado por la siguiente representación binaria en coma flotante:

$$R = (\pm)0.m_1m_2m_3\ldots m_k \cdot 2^{\pm e},$$

donde $m_1m_2m_3\ldots m_k$ se llama la **mantisa** y el número e (que escribimos aquí en forma de entero decimal) es el **exponente**, es decir, el número de lugares que hay que trasladar la coma a derecha o izquierda para obtener el número inicial. Obsérvese que m_1 tiene que ser siempre igual a 1, ya que es, por definición, la primera cifra no nula empezando por la izquierda.

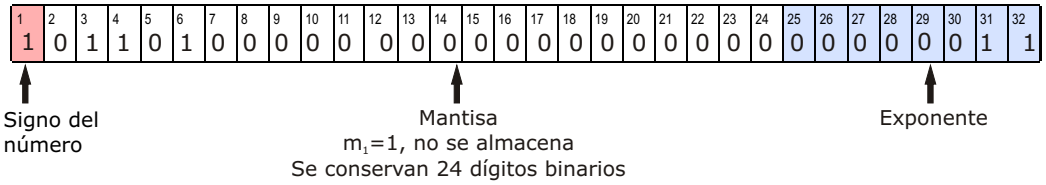
EJEMPLO:
$R = 5.625_{(10)} = 101.101_{(2)} = 0.101101 \cdot 2^3$

Para almacenar dicho número, los 32 bits de una palabra se distribuyen [4] de la siguiente manera:

- 1 bit para el signo del número (0 si es positivo, 1 si es negativo).
- 23 bits para la mantisa. Como m_1 es siempre igual a 1 se utiliza el pequeño " truco" de no almacenarlo. Así, aunque se almacenan 23 cifras binarias se "conocen" en realidad 24.
- 8 bits para el exponente con su signo (se guarda codificado como un número entero).

EJEMPLO:

$R = -5.625_{(10)} = -0.101101 \cdot 2^3, \ 3_{10} = 11_2$



Se tiene:

- El mayor exponente positivo que se puede almacenar: $1111111 = 2^6+2^5+\cdots+2^0 = 2^7-1 = 127$.
- El menor exponente negativo: -127.
- El número de mayor magnitud: $\approx 2^{127} \approx 10^{38}$.
- El número de menor magnitud: $\approx 2^{-127} \approx 10^{-38}$.

1.7.5 Overflow y underflow

Es claro que usando palabras de 32 bits sólo podemos representar una cantidad finita de números, en consecuencia podemos encontrar los siguientes fenómenos:

- **Overflow**: fenómeno que se produce cuando el resultado de un cálculo es un número real con exponente demasiado grande (en el ejemplo anterior > 127).
  - Cuando se produce un overflow durante la ejecución de un programa, los ordenadores modernos generan un "evento" de error y como resultado devuelven el símbolo `Inf`.
- **Underflow**: fenómeno que se produce cuando el resultado de un cálculo es un número real con exponente demasiado pequeño (en los ejemplos arriba mencionados < -127).
  - Cuando se produce durante la ejecución de un programa, normalmente se sustituye el número por cero.

[4] Esta distribución puede variar ligeramente de un ordenador a otro.

1.7.6 Errores

Al realizar cálculos en el ordenador, es inevitable cometer errores. Grosso modo, estos errores pueden ser de tres tipos:

1. **Errores en los datos de entrada**: vienen causados por los errores al realizar mediciones de magnitudes físicas.

2. **Errores de truncamiento (o discretización)**: En la mayoría de las ocasiones, los algoritmos matemáticos que se utilizan para calcular un resultado no son capaces de calcular su valor exacto, limitándose a calcular una **aproximación** del mismo. Este error es inherente al algoritmo de cálculo utilizado.

3. **Errores de redondeo**: aparecen debido a la cantidad finita de números que podemos representar en un ordenador. Por ejemplo, consideramos las dos siguientes mantisas-máquina consecutivas que podemos almacenar en 23 bits (recordar que el primer dígito es siempre 1 y por tanto no se guarda):

$$R_1 = 0.\underbrace{10000000000000000000000}_{24 \text{ dígitos}}$$

$$R_2 = 0.\underbrace{10000000000000000000001}_{24 \text{ dígitos}}$$

Ningún número real entre R_1 y R_2 es representable en un ordenador que utilice 23 bits para la mantisa. Tendrá que ser aproximado bien por R_1, bien por R_2, cometiendo con ello un error E que verifica:

$$E \leq \frac{|R_1 - R_2|}{2} = \frac{2^{-24}}{2} = 2^{-25} \simeq 10^{-7}$$

puesto que $|R_1 - R_2| = 0.00000000000000000000001 = 0.1 \times 2^{-23} = 2^{-24}$. De esta forma, en el almacenamiento de la mantisa se comete un error menor o igual que 10^{-7} o, dicho de otra forma, sólo los 7 primeros dígitos significativos de un número decimal almacenado en el ordenador se pueden considerar exactos.

EJEMPLO:

$$\begin{aligned} R = 0.1_{(10)} &= 0.\widehat{00011}_{(2)} \\ &= 0.00011001100110011\ldots \\ &= = 0.\underbrace{11001100110011001100}_{24}1100\cdots \times 2^{-3} \end{aligned}$$

R no es un número-máquina (su mantisa binaria tiene infinitos dígitos).

La aproximación de R por redondeo es:

$$\overline{R} = 0.\underbrace{11001100110011001101}_{24 \text{ dígitos}} \times 2^{-3}.$$

Aritmética en coma flotante

Para llevar a cabo una operación aritmética en un ordenador, hay que tener en cuenta que los números con los que trabajamos pueden no ser exactamente los de partida, sino sus aproximaciones por redondeo. Incluso aunque los números sean números-máquina (representables en palabras de 32 bits), los resultados que obtengamos puede que no lo sean. A continuación vamos a ver algunos ejemplos de cálculos efectuados en la aritmética de coma flotante, en el caso de un ordenador (hipotético) que trabaje con números reales en base 10, con 7 dígitos para la mantisa, y con aproximación por redondeo.

EJEMPLO:
Para sumar dos números en coma flotante, primero se representan con el mismo exponente, luego se suman y el resultado se redondea:

$$\begin{aligned} x_1 &= 0.1234567 \cdot 10^3 = 0.123456700000 \cdot 10^3 \quad (= 123.4567) \\ x_2 &= 0.3232323 \cdot 10^{-2} = 0.000003232323 \cdot 10^3 \quad (= 0.003232323) \end{aligned}$$

$$\begin{array}{rll} & 0.123456700000 & \cdot 10^3 \\ + & 0.000003232323 & \cdot 10^3 \\ \hline & 0.12345993232 3 & \cdot 10^3 \quad \text{(sólo se almacenan los 7 primeros dígitos)} \\ & 0.1234599 & \cdot 10^3 \quad \text{(resultado-máquina de } x_1 + x_2\text{)} \end{array}$$

Obsérvese que las 5 últimas cifras de x_2 no contribuyen a la suma.

EJEMPLO:
Para multiplicar (la división se hace por algoritmos más complicados) dos números en coma flotante se multiplican las mantisas y se suman los exponentes. El resultado se normaliza:

$$\begin{aligned} x_1 &= 0.4734612 \cdot 10^3 \\ x_2 &= 0.5417242 \cdot 10^5 \end{aligned}$$

$$\begin{array}{rll} & 0.4734612 & \cdot 10^3 \\ \times & 0.5417242 & \cdot 10^5 \\ \hline & 0.25648538980104 & \cdot 10^8 \\ & 0.2564854 & \cdot 10^8 \quad \text{(resultado-máquina de } x_1 \cdot x_2\text{)} \end{array}$$