Representación de la información en las computadoras

En esta unidad se estudian los aspectos relacionados con la representación de la información en el interior de las computadoras. Se considera tanto la representación de los **datos numéricos** como los **no numéricos**. También se analizan los aspectos prácticos de los sistemas de numeración desde el punto de vista de su aplicación a la Informática, incluyendo las operaciones básicas booleanas binarias, así como las operaciones aritméticas básicas en notación entera y exponencial.

1 Introducción

Una computadora es una máquina que procesa información. Más concretamente, la ejecución de un programa implica la realización de unos tratamientos; según especifica un conjunto ordenado de instrucciones (es decir, el programa) sobre unos datos. Obviamente, para que la computadora ejecute un programa es necesario darle dos tipos de informaciones: las instrucciones que forman el programa y los datos con los que debe operar ese programa.

Dos de los aspectos más importantes que se presentan en Informática relacionados con la información es cómo *expresarla*, cómo materializarla ó *registrarla físicamente* y cómo *transmitirla*. Los tres problemas están íntimamente relacionados y el primero de ellos es el objetivo fundamental de este capítulo. El segundo se resuelve con soportes de información (discos y cintas magnéticas, etc.). El tercero será motivo de discusión en el resto de las unidades de trabajo.

Normalmente la información se da a la computadora en la forma usual escrita que utilizan los seres humanos; es decir, con ayuda de un alfabeto o conjunto de símbolos que denominamos caracteres.

Los caracteres que constituyen el alfabeto suelen agruparse en cuatro categorías, aparte de los caracteres gráficos:

Caracteres alfabéticos: Son las letras mayúsculas y minúsculas del abecedario inglés:

A,B,C,...,X,Y,Z, a,b,c,.,.,x,y,z

Caracteres numéricos: Están constituidos por las diez cifras decimales:

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

El cero suele marcarse con una raya inclinada para evitar posibles confusiones con la O mayúscula.

Caracteres especiales: Son los símbolos no incluidos en los grupos anteriores, entre otros, los siguientes:

) (. * / : ; + Ñ ñ ¡ ! ¿ ? , = " & > # <] Ç [SP

Con SP representamos el carácter o espacio en blanco, tal como el que separa dos palabras.

Caracteres de control: Representan órdenes de control, como el carácter indicador de fin de línea o el carácter indicador de sincronización de una transmisión o de que se emita un pitido en un terminal, etc. Muchos de los caracteres de control son generados e insertados por la propia computadora. No tienen representación gráfica.

Este capítulo se dedica principalmente a los tres primeros tipos de caracteres, a veces denominados caracteres-texto. Al conjunto de los dos primeros tipos se le denomina conjunto de caracteres alfanuméricos.

De la misma forma que una máquina de escribir no suele tener teclas para todos los caracteres posibles (unas tienen unos caracteres especiales y otras otros), un sistema de procesamiento de la información (periférico o procesador) puede no reconocer algunos caracteres (las letras minúsculas o la ñ, por ejemplo). Toda comunicación con una computadora convencional se realiza según los caracteres que admitan sus dispositivos de E/S, y con ellos los usuarios tienen que expresar cualquier dato o instrucción. Es decir, toda instrucción o dato se representará por un conjunto de caracteres tomados del alfabeto definido en el sistema a utilizar. A continuación se va a ver cómo se pueden representar los caracteres en las computadoras.

El diseño de un sistema informático resulta *más fáci*l, su realización *menos compleja* y su funcionamiento *muy fiable* si se utilizan sólo dos valores o estados posibles para las variables físicas que representan los caracteres en los soportes de información o en el interior de la computadora. Estos valores conceptualmente se representan por cero (0) y uno (1) y corresponden a dos niveles de tensión (0 voltios y 3,5 voltios, por ejemplo), dos valores de corriente, dos situaciones de una lámpara (apagada y encendida), etc. En otras palabras, la información es retenida por medio de dos valores de una magnitud física (bit: **Binary digIT**) que conceptualmente se pueden representar por ceros y unos.

Al tener que traducir toda la información suministrada a la computadora a ceros y unos es necesario establecer una correspondencia entre el conjunto de todos los caracteres.

$$\alpha \equiv \{A, B. C, ..., Z, a, b, c, ..., z, 0, 1, 2, 3. ..., 9, /, +. (,), ... \}$$

y el conjunto binario

$$\beta \equiv \{0, 1\}^n;$$

es decir, es necesario hacer una codificación o representación de los elementos de un conjunto (en este caso α) mediante los de otro (β) de forma tal que a cada elemento de α le corresponda un elemento distinto de β (n bits).

Éstos códigos de transformación se denominan **Códigos de Entrada/Salida** (E/S) o códigos externos y pueden definirse de forma arbitraria. No obstante existen códigos de E/S normalizados que son utilizados por diferentes constructores de computadoras; estos códigos se estudiarán un poco más adelante.

Las operaciones aritméticas con datos numéricos se suelen realizar en una representación más adecuada para este objetivo que la obtenida con el código de E/S. Por ello, en la propia computadora se efectúa una transformación entre códigos binarios, obteniéndose una representación fundamentada en el sistema de numeración en base dos, sistema que puede considerarse como una codificación en binario, pero que al ser una representación numérica posicional es muy apta para realizar operaciones aritméticas.

2 Sistemas de numeración usuales en informática

Las computadoras suelen efectuar las operaciones aritméticas utilizando una representación para los datos numéricos basada en el sistema de numeración base dos; que, por abreviar, denominaremos binario natural, o, cuando no se preste a confusión con códigos binarios de E/S o texto, **binario**, sin más.

También se utilizan los sistemas de numeración, preferentemente el **octal** y **hexadecimal**, para obtener códigos intermedios. Un número expresado en uno de estos dos sistemas puede transformarse directa y fácilmente a binario, y viceversa. Con ellos se simplifica la trascripción de números binarios y están más próximos al sistema que utilizamos usualmente (el sistema decimal), por lo que a veces se utilizan como paso intermedio en las transformaciones de decimal a binario, y viceversa.

2.1 Representación posicional de los números

Un sistema de numeración en base b utiliza para representar los números un alfabeto compuesto por b símbolos o cifras. Así todo número se expresa por un conjunto de cifras, contribuyendo cada una de ellas con un valor que depende:

a) de la cifra en sí,

b) de la posición que ocupa dentro del número.

En el sistema de numeración decimal (o sistema en base 10), que es el que habitualmente se utiliza (b = 10), el alfabeto está constituido por diez símbolos, denominados también cifras decimales:

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

y, por ejemplo, el número 3278.52 puede obtenerse como suma de:

3000 + 200 + 70 + 8 + 0.5 + 0.02 = 3278.52

es decir, se verifica que:

$3278.52 = 3 \times 10^3 + 2 \times 10^2 + 7 \times 10^1 + 8 \times 10^o + 5 \times 10^{-1} + 2 \times 10^{-2}$

Cada posición, por tanto, tiene un peso y un nombre específicos:

posición	0	peso b°	unidades	(en el ejemplo: 8)
posición	1	peso b^1	decenas	(7)
posición	2	peso b^2	centenas	(2)
posición	3	peso b^3	millares	(3)

.....

Generalizando, se tiene que la representación de un número en una base b:

$$N \equiv \ldots n_4\, n_3\, n_2\, n_1\, n_0\, .\, n_{-1}\, n_{-2}\, n_{-3} \ldots$$

es una forma abreviada de expresar su valor, que es:

$$N \equiv \ldots n_4 b^4 + n_3 b^3 + n_2 b^2 + n_1 b^1 + n_0 b^0 + n_{-1} b^{-1} + n_{-2} b^{-2} + n_{-3} b^{-3} \ldots$$

No estamos hablando ni más ni menos que del ***Teorema fundamental de la numeración***.

Para representar un número, por un lado, resulta más cómodo que los símbolos (cifras) del alfabeto o la base de numeración sean los menos posibles, pero, por otra parte, cuanto menor es la base, mayor es el número de cifras que se necesitan para representar una cantidad dada.

En base ocho, el alfabeto que se suele utilizar es:

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

y b=8.

El valor decimal del número octal 175.372 será:

$$175.372_{(8} = 1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 + 3 \times 8^{-1} + 7 \times 8^{-2} + 2 \times b\text{-}3^2 =$$

$$= 125.4882812_{(10}$$

2.2 Sistema de numeración en base dos

Según se indicó anteriormente, las operaciones aritméticas se suelen realizar utilizando una representación para los datos y resultados en binario natural. A pesar de que el cambio del código binario de E/S a la representación en binario natural lo realiza automáticamente la computadora, es conveniente recordar aquí una serie de cuestiones relativas al sistema de numeración binario y a las transformaciones entre él y el sistema decimal.

2.2.1 Definición del sistema binario

En el sistema de numeración binario es b = 2, y se necesitan tan sólo dos elementos para representar cualquier número:

{0, 1}

Los elementos de este alfabeto se denominan cifras binarias o bits. En la siguiente tabla se muestran los números enteros binarios que se pueden formar con 3 bits, que corresponden a los decimales de 0 a 7.

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

2.2.2 Transformaciones entre bases binaria y decimal

Se puede transformar un número binario a decimal sin más que tener en cuenta las expresiones del *Teorema fundamental de la numeración* para el caso de b = 2.

Para transformar un número decimal a binario:

La parte entera del nuevo número (binario) se obtiene dividiendo por 2 (sin obtener decimales en el cociente; es decir, se realizar la división entera) la parte entera del número decimal de partida, y de los cocientes que sucesivamente se vayan obteniendo. Los residuos (restos) de estas divisiones y el último cociente (que serán siempre ceros o unos) son las cifras binarias. El último cociente será el bit más significativo (MSB; "Most Significative Bit") y el primer residuo será el bit menos significativo (LSB; "Less Significative Bit").

La parte fraccionaria del número binario se obtiene multiplicando por 2 sucesivamente la parte fraccionaria del número decimal de partida y las partes fraccionarias que se van obteniendo en los productos sucesivos. El número binario se forma con las partes enteras (que serán ceros o unos) de los productos obtenidos de izquierda a derecha en el orden en el que se van obteniendo.

Se observa que, en ocasiones, un número decimal con cifras fraccionarias puede dar lugar a un número de cifras fraccionarias mucho mayor o incluso infinito. Si el número binario se almacena con un número prefijado de bits se producirá en la representación binaria un error de truncamiento.

3 Operaciones aritméticas y lógicas con variables binarias

Una variable binaria puede representar, entre otras cosas, una cifra de un número en el sistema de numeración de base dos o una variable booleana (o variable de conmutación; la que comúnmente, en el argot, denominamos "switch").

Las operaciones aritméticas básicas son la suma, resta, multiplicación y división. Al ser la representación en binario natural una notación ponderada estas operaciones son análogas a las realizadas en decimal, pero hay que utilizar para cada posición las siguientes tablas, según la operación aritmética a realizar.

Suma aritmética	Resta aritmética	Producto aritmético	División aritmética
0 + 0 = 0	0 – 0 = 0	0 x 0 = 0	0 : 0 = indeterminado
0 + 1 = 1	0 – 1 = 1 y debo 1	0 x 1 = 0	0 : 1 = 0
1 + 0 = 1	1 – 0 = 1	1 x 0 = 0	1 : 0 = ∞
1 + 1 = 0 y me llevo 1	1 – 1 = 0	1 x 1 = 1	1 : 1 = 1

Se observa que multiplicar por $10_{(2}$ (es decir, por 2 en decimal) equivale a añadir un cero a la derecha, siendo esto similar a multiplicar por $10_{(10}$, un número decimal. De la misma forma dividir por $2_{10} = 10_{(2}$ se hace desplazando el punto decimal a la izquierda, o eliminando ceros a la derecha.

Las operaciones booleanas u operaciones lógicas con variables binarias (consideradas éstas como variables de conmutación) más importantes son la suma lógica (+) (también denominada unión o función OR u O), el producto lógico (·) (también denominado intersección o función AND o Y) y la complementación (¯ ó ¬) (o inversión o negación o función NOT o NO). Estas operaciones se rigen según las siguientes tablas:

Suma lógica	Producto lógico	Complementación
0 + 0 = 0	0 · 0 = 0	
0 + 1 = 1	0 · 1 = 0	¬0 = 1
1 + 0 = 1	1 · 0 = 0	¬1 = 0
1 + 1 = 1	1 · 1 = 1	

Es frecuente también la utilización de operaciones combinadas como NAND (AND con NOT) y NOR (OR con NOT).

NAND			NOR		
A B	A· B	¬(A· B)	A B	A + B	¬(A + B)
0 0	0	1	0 0	0	1
0 1	0	1	0 1	1	0
1 0	0	1	1 0	1	0
1 1	1	0	1 1	1	0

4 Representación en complementos

Para representar un número negativo se puede utilizar el complemento de ese número a la base o a la base menos uno del sistema de numeración utilizado. De esta forma como se verá más adelante, las sumas y restas quedan reducidas a sumas, independientemente de los signos de los operandos. Este sistema de representación es de interés en el caso de las computadoras, ya que al utilizarlo se reduce la complejidad de los circuitos de la unidad aritmético-lógica (no son necesarios circuitos específicos para restar).

- **El complemento a la base menos uno** de un número N es el número que resulta de restar cada una de las cifras de N a la base menos 1 del sistema de numeración que se esté utilizando.

Se puede restar dos números sumando al minuendo el complemento a la base menos uno del sustraendo. La cifra que se arrastra del resultado (*acarreo*) se descarta y se suma al resultado así obtenido.

Fácilmente se observa que para transformar un número binario, N, a complemento a 1 basta con cambiar en N los unos por ceros y los ceros por unos.

- **El complemento a la base de un número**, N, es el número que resulta de restar cada una de las cifras del número N a la base menos uno del sistema que se esté utilizando y posteriormente sumar uno a la diferencia obtenida; o lo que es lo mismo, sumar 1 a la base menos uno de un número.

Se puede restar dos números sumando al minuendo el complemento a la base del sustraendo y despreciando, en su caso, el acarreo del resultado.

Observamos que para transformar un número binario, N a complemento a 2 basta con cambiar los ceros por unos y los unos por ceros de N y sumar 1 al resultado.

Como resumen de las definiciones dadas y generalizando se tiene que:

Complemento a la base, b, de un número N de n cifras: $(b^n - N)$

Complemento a la base menos uno, b –1, de un número N de n cifras. $(b^n - N - 1)$.

5 Códigos intermedios

Los códigos intermedios se fundamentan en la facilidad de transformar un número en base 2 a otra base que sea una potencia de 2 ($2^2 = 4$; $2^3 = 8$; $2^4 = 16$, etc.), y viceversa.

Usualmente se utilizan como códigos intermedios los sistemas de numeración en base 8 (u octal) y en base 16 (o hexadecimal).

5.1 Base octal

En la base octal, b=8 y el conjunto de símbolos utilizado es:

{0, I, 2, 3, 4, 5, 6, 7}

Un número octal puede pasarse a binario aplicando los algoritmos anteriores y utilizando como base intermedia la base decimal. No obstante, al ser $b=8=2^3$, puede hacerse la conversión fácilmente en la forma que se indica a continuación.

- Para transformar un número binario a octal se forman grupos de tres cifras binarias a partir del punto decimal hacia la izquierda y hacia la derecha. Posteriormente se efectúa directamente la conversión a octal de cada grupo individual.

Como en la práctica sólo es necesario hacer la conversión a octal (o decimal) de cada uno de los grupos de tres cifras, basta con memorizar la representación en binario de los números del 0 al 7 para poder realizar rápidamente la conversión.

$$10\ 001\ 101\ 100.110\ 10\ _{(2} = 2154.64\ _{(8}$$

$$1\ 101\ _{(2} = 15\ _{(8}$$

De octal a binario se pasa sin más que convertir individualmente a binario (tres bits) cada cifra octal, manteniendo el orden del número original.

$$537.24\ _{(8} = 101\ 011\ 111.\ 010\ 100\ _{(2}$$

$$175.22\ _{(8} = 1\ 111\ 101.\ 010\ 010\ _{(2}$$

Para transformar un número de octal a decimal no hay truco, se aplica lo visto en las secciones anteriores con b = 8. Para pasar un número entero de decimal a octal, igualmente, se hacen sucesivas divisiones enteras del número y los subsiguientes cocientes por ocho (tal como en binario se hacía por 2). Para transformar la parte fraccionaria de un número decimal a octal se hacen sucesivas multiplicaciones por ocho (como para pasar a binario se hacía por 2). Es decir, se actúa como para cualquier otra base.

5.2 Base hexadecimal

Para representar un número en base hexadecimal (esto es, b = 16) es necesario disponer de un conjunto o alfabeto de 16 símbolos. Se suele utilizar el siguiente conjunto:

{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Al ser $b = 16 = 2^4$, se pueden hacer las conversiones de binario a hexadecimal, y viceversa, en forma análoga al sistema octal. Ahora bien, aquí utilizaremos grupos de 4 bits en lugar de grupos de 3 bits.

$$10\ 0101\ 1101\ 1111.1011\ 101\ _{(2} = 25DF.BA\ _{(16}$$

$$111\ 1111\ 1011\ 1100\ 0010\ _{(2} = 7FBC2\ _{(16}$$

De la misma forma que manualmente, es muy fácil convertir números de binario a octal, y viceversa, y de binario a hexadecimal, y viceversa. También resulta sencillo efectuar esta operación electrónicamente o por programa, por lo que a veces la computadora utiliza este tipo de

notaciones intermedias internamente o como entrada/salida. Pero sobre todo su utilidad radica en reducir las ristras de 1's y 0's para evitar errores de trascripción.

Para transformar un número de hexadecimal a decimal se aplica el teorema fundamental de la numeración con b = 16. Para pasar un número de decimal a hexadecimal se hace de forma análoga a los casos binario y octal: la parte entera se divide por 16, así como los cocientes enteros sucesivos, y la parte fraccionaria se multiplica por 16, así como las partes fraccionarias de los productos sucesivos. Es decir, actuaremos en ambos casos como para cualquier otra base.

6 Códigos de Entrada/Salida

Ya se indicó en la introducción de esta unidad que los códigos de entrada/salida (E/S) o códigos externos son códigos que asocian a cada carácter (alfabético, numérico o especial) una determinada combinación de bits. En otras palabras, un código de E/S es una correspondencia entre los conjuntos:

$$\alpha \equiv \{A, B. C, ..., Z, a, b, c, ..., z, 0, 1, 2, 3. ..., 9, /, +. (,), ... \}$$

y el conjunto binario

$$\beta \equiv \{0, 1\}^n;$$

El número n de elementos del conjunto β depende del dispositivo o sistema informático o codificación de E/S que se esté utilizando.

Supóngase que se utiliza un número fijo, n, de bits para codificar los símbolos de α. El valor mínimo de n dependerá del número m de elementos de α. Así:

Con 2 bits (n = 2) se pueden hacer 4 combinaciones distintas con lo que se pueden codificar 4 símbolos (m = 4) distintos.

Con 3 bits (n = 3) se pueden hacer 8 combinaciones distintas, con lo que se pueden codificar hasta 8 símbolos (m = 8) distintos.

En general, con n bits se pueden codificar $m = 2^n$ símbolos distintos.

En otras palabras, se verifica que para codificar m símbolos distintos se necesitan n bits; siendo,

$$n \geq \log_2 m$$

Como en la práctica n debe ser entero, n es el menor número entero que verifica esta relación.

Uno de los códigos usuales, ASCII, suele utilizar unos 95 caracteres (es decir, m = 95) por lo que el número de bits mínimos para codificarlos es 7 ya que

$$64 = 2^6 < 95 < 2^7 = 128$$

Para los 10 símbolos decimales, se podrían establecer códigos de E/S de forma totalmente arbitraria y con un número de bits n = 4 según se ha visto con la fórmula anterior. Obviamente existen códigos normalizados y que suelen ser utilizados por los constructores de computadoras como son el **código ASCII** (American Standard Code for Information Interchange) de 7 bits, utilizado habitualmente para transmisión de datos, **el código ASCII extendido** de 8 bits, propio de los PC´s, y el código **EBCDIC** (Extended Binary Coded Decimal Interchange Code); utilizado ampliamente en mainframes de IBM.

7 Representación de números reales

Cuando operamos con números muy grandes es frecuente utilizar la notación exponencial para representarlos. Así, por ejemplo el número 50467894.1235 podría expresarse como

$50467894.1235 = 504678941235 \times 10^4 = 5.04678941235 \cdot 10^7 = 0.504678941235 \cdot 10^8 = ...$

Es decir, que todo número N lo podemos expresar de la forma: $N = M \times B^{exp}$

donde en el ejemplo anterior la base B = 10, M = 0.504678941235 representa la mantisa y $^{exp} = 8$ el exponente. Este tipo de representación de los números se denomina notación exponencial, notación científica o notación en punto o coma flotante (ya que parece como si el punto flotase de derecha a izquierda al cambiar el valor del exponente).

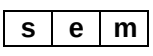
Podemos transformar la representación de N, conservando su valor, cambiando el exponente ^{exp} y reajustando adecuadamente la mantisa M:

- si dividimos M por B, aumentaremos ^{exp} en una unidad.
- si multiplicamos M por B, disminuiremos ^{exp} en una unidad.

El ordenador emplea siempre esta notación exponencial para representar y manejar los números reales, siendo responsabilidad del hardware del computador o de los traductores de lenguajes el conseguirlo. Los microprocesadores potentes contienen internamente, en el mismo chip, los circuitos para operar en coma flotante y, en otros casos, se dispone de circuitos integrados (**coprocesadores aritméticos**) para realizar específicamente estas operaciones. Si el hardware no dispone de circuitería para coma flotante, y un lenguaje de programación dispone de este tipo de datos, será el traductor correspondiente el que descomponga las operaciones en coma flotante en términos de las operaciones que presenta el lenguaje máquina, obteniéndose en este caso un rendimiento (velocidad) mucho menor en la ejecución de los programas; es decir, se realiza una emulación de las tareas del coprocesador aritmético.

Hasta la década de los años ochenta puede decirse que cada fabricante de computadores utilizaba su sistema propio para la representación de números reales; pero es de gran importancia que existan sistemas normalizados, ya que ello posibilita que los matemáticos construyan bibliotecas de programas de alta calidad, los diseñadores de computadores puedan construir unidades aritmético-lógicas muy eficientes, y los fabricantes de circuitos integrados puedan construir aceleradores y coprocesadores aritméticos estándar. Debido a ello, de 1977 a 1985, la asociación IEEE desarrolló un sistema normalizado de representación, denominado Normalización IEEE 754 que es el que hoy día tiene mayor aceptación. Estudiaremos a continuación dicha norma:

- Todo número se expresa en la forma $N = M \times B^{exp}$ siendo el exponente entero.
- La base del exponente está predeterminada y es B = 2, por lo que no es necesario almacenarla.
- El orden de almacenamiento es:

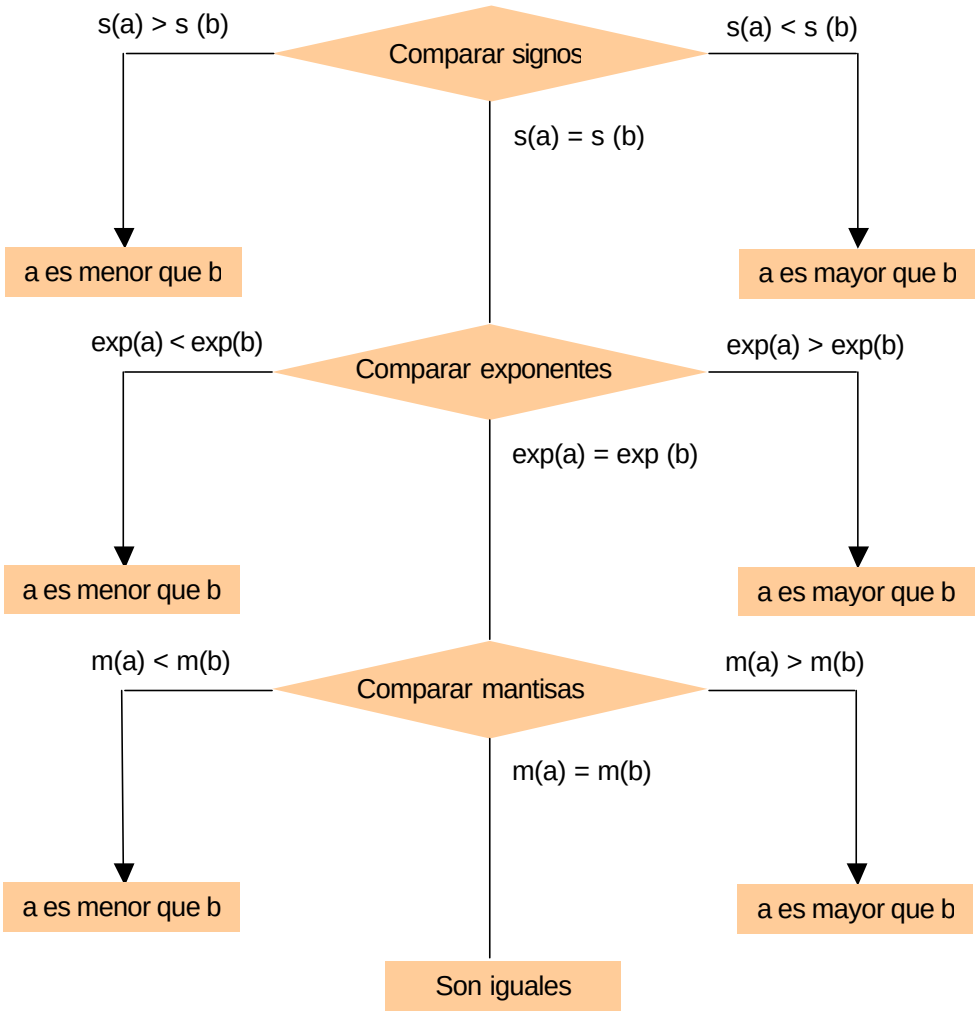


- **s**.- Es el campo del signo (ocupa 1 bit) e indica el signo del número: vale 0 para los números positivos y 1 para los negativos.

- **e**.- Es el campo del exponente o característica (ocupa ne bits, incluyendo el signo del exponente)

- **m**.- Es el campo de la mantisa (ocupa nm bits)

Ese orden de almacenamiento posibilita que los algoritmos de comparación entre números enteros (hallan si un número es mayor, igual o menor que otro) sean también válidos para la representación de números reales.



- El exponente se almacena en la forma de entero sesgado, es decir, el campo del exponente e se obtiene siempre sumando al exponente del número (E) un sesgo (valor constante) S

$$S = 2^{ne-1} - 1 \Rightarrow e = S + exp = 2^{ne-1} - 1 + exp$$

De esta forma en los ne bits reservados para el exponente se pueden incluir exponentes positivos o negativos sin utilizar un bit explícito de signo.

- El número se almacena normalizado, es decir, que se ajusta el exponente de forma tal que el 1 más significativo de la mantisa se encuentre en la posición de las unidades.

Ejemplos: $1001.1100\ 110 \times 2^{-5}$ normalizado sería $1.0011\ 1001\ 10 \times 2^{-2}$

$0.0000\ 1101\ 101 \times 2^{34}$ normalizado sería $1.1011\ 01 \times 2^{28}$

- El campo de la mantisa se obtiene almacenando sólo la parte fraccionaria del número normalizado. Esto se hace así porque todos los números normalizados empiezan siempre por un uno y un punto, por lo que se ahorra espacio de memoria no almacenando esta cabecera y se dice que el uno y el punto está implícito o que el número se encuentra empaquetado. Obviamente cuando la ALU (o el traductor) realice cualquier operación, debe restituir el «1.» haciéndolo explícito; es decir, debe desempaquetar el dato.

  En otras palabras, la mantisa M normalizada debe verificar

  $$1 \le M < 2$$

  o sea,

  $$M = [1.m]$$

  donde m es el valor que se almacena y 1. el valor explícito, que no se almacena.

Las normalizaciones son necesarias para no perder precisión (cifras significativas) en operaciones sucesivas. Esto se debe a que el número de cifras binarias de la mantisa (nm) es fijo y conviene, por tanto, normalizar la mantisa de forma que las cifras significativas ocupen las posiciones de mayor peso. Veamos las especificaciones de la norma IEEE 754:

	nm (bits de precisión)	ne (bits del exponente)	Exponente máximo	Exponente mínimo	Sesgo del exponente
Simple	23	8	127	-126	127
Doble	52	11	1023	-1022	1023

Ejemplo:

Hallar el valor del siguiente número almacenado en formato IEEE 754 simple precisión:

1	0011 1110	0011 1100 0000 0000 0000 000

El signo es negativo puesto que s = 1

$$e = 0011\ 1110_{(2} = 3E_{(16} = 62 \Rightarrow \quad exp = e - S = 62 - 127 = -65$$

$$m = 0011\ 110....0 \Rightarrow M = 1.0011\ 110 =$$

$$= 1 + 0.125 + 0.0625 + 0.03125 + 0.015625 = 1.234375$$

Por tanto

$$N = -M \times 2^{exp} = -1.234375 \times 2^{-65} = -3.345780142 \times 10^{-20}$$

Situaciones especiales:

- Cuando el campo del exponente toma su valor mínimo (e=0), el 1 más significativo de la mantisa no se encuentra implícito, y entonces la mantisa se almacena denormalizada. En este caso el sesgo es

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

es decir, el valor del exponente correspondiente a los números denormalizados es:

$$E = e - S = -(2^{ne-1} - 2)$$

- El número N = 0 se representa con todos los bits del campo del exponente y del campo de la mantisa a cero.

- Si todos los bits del campo del exponente son 1 (*exp* adquiere su valor máximo), el dato representa:

  - Si m = 0, más o menos infinito (el resultado de dividir por 0, por ejemplo).

  - Si m ≠ 0, representa un **NaN** (*Not a Number*, o sea, un "no número"). Estos patrones de bits se utilizan para almacenar valores no válidos (resultados de operaciones tales como la raíz cuadrada de un número negativo, 0 x ∞, ∞ x ∞, etc.).

Ejemplo

$$N = -0.0025 = -(0.0025 \times 10^{2}) \times 10^{-2} = -0.25 \times 10^{-2} = -(0.25 \times 2^{-6.643856189775}) =$$

$$= -(0.25 \times 2^{-0.643856189775}) \times 2^{-6} = -0.16 \times 2^{-6} = -0.28F5C28_{(16} \times 2^{-6} =$$

$$= -0.0010\ 1000\ 1111\ 0101\ 1100\ 0010\ 1000_{(2} \times 2^{-6} = -1.0100\ 0111\ 1010\ 1110\ 0001\ 0100_{(2} \times 2^{-9}$$

Característica = Sesgo + Exponente = 127 - 9 = 118 = 0111 0110

Resultado: 1 0111 0110 0100 0111 1010 1110 0001 010 = BB23D70A$_{(16}$

Ejemplo:

$$N = 325496.623 = 4F778.9F7CED91_{(16} = 0100\ 1111\ 0111\ 0111\ 1000.1001\ 1111\ 0111_{(2}$$

$$= 1.0011\ 1101\ 1101\ 1110\ 0010\ 100 \times 2^{18}$$

$$\text{Característica} = 127 + 18 = 145 = 91_{(16}$$

Repres. Interna : 0 1001 0001 0011 1101 1101 1110 0010 100 = 48 9E EF 14

- Vamos a hallar cuál es el error absoluto cometido:

$$\exp = 145 - 127 = 18$$

$$\text{Matisa} = 1.0011\ 1101\ 1101\ 1110\ 0010\ 100_{(2} = 1001111011101111000.10100_{(2} = 325496 + 1/2 + 1/8 =$$

$$= 325496.625 \qquad \text{Error absoluto} = |325496.623 - 325496.625| = 0.002$$

Ejemplo

$$N= - 0.00625 = 0.01999999_{(16} = 0.00000001\ 1001\ 1001\ 1001\ 1001_{(2} =$$

$$= 1.\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001_{(2} \times 2^{-8}$$

$$\text{Característica} = 127 - 8 = 119 = 77_{(16}$$

$$\text{Repres. Interna} = 1011\ 1011\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101 = \text{BBCCCCCD}_{(16}$$

Lo visto hasta ahora nos permitirá fácilmente reconocer el formato que, por ejemplo, utiliza C++ Builder para almacenar la información numérica: