

TEMA 2

LA INFORMACIÓN Y SU REPRESENTACIÓN

Introducción

Sistemas de numeración usuales en informática

- Representación posicional de los números
- Sistema de numeración binario
- Operaciones aritméticas y lógicas con variables binarias

Códigos intermedios y conversiones

- Octal
- Hexadecimal

Representación de números enteros

- Módulo y signo
- Complemento a C-1
- Complemento a C-2
- Exceso a 2^{n-1}
- Importancia de la representación en complementos.

Representación de números con punto fijo

- Binario puro
- Decimal codificado en binario (BCD)
- Decimal desempquetado
- Decimal empaquetado

Representación en coma flotante

Códigos de entrada / salida

- Código BCD de intercambio normalizado
- Código EBDIC
- Código ASCII

LA INFORMACIÓN Y SU REPRESENTACIÓN

INTRODUCCIÓN

Como vimos, un computador es una máquina que procesa información , (un conjunto de *instrucciones* que se ejecutan sobre un conjunto de *datos*.)

El hombre suministra información a la máquina mediante símbolos (**caracteres**) : Estos podemos dividirlos en cinco categorías:

- Caracteres *alfabéticos*: { a,b,...,z,A,B,...,Z }.
- Caracteres *numéricos*: { 0,1,...,9 }.
- Caracteres *especiales*: { (,) ,*,+,-, ?, ... }.
- Caracteres *de control*: { fin de línea , carácter de sincronización, avance página, pitido, ... }.
- Caracteres *gráficos*: { ␣, ♣, ♠, (, ... }

A todos los caracteres de los grupos primero y segundo se les denomina *caracteres alfanuméricos* y a los de los grupos primero, segundo y tercero, *caracteres de texto*.

Un ordenador, debido a su construcción basada fundamentalmente en circuitos electrónicos digitales, trabaja con el sistema binario (1 = 5V, 0 = 0V), usando una serie de códigos que permiten su funcionamiento.

Este es el motivo que nos obliga a transformar internamente todos nuestros datos, tanto numéricos como alfanuméricos, a una representación binaria para que la máquina sea capaz de procesarlos. Además del sistema de numeración binario, el ordenador también trabaja para la codificación numérica con los sistemas de numeración octal y hexadecimal.

Definimos **sistema de numeración** como el conjunto de símbolos y reglas que se utilizan para representar cantidades o datos numéricos.

Tienen como característica una *base* a la que referencian y que determina el diferente número de símbolos que lo componen. Nosotros utilizamos el sistema de numeración en base 10, compuesto por diez símbolos diferentes (del 0 al 9).

Los sistemas de numeración que utilizamos son sistemas posicionales, es decir, el valor relativo que cada símbolo representa quedará determinado por su valor absoluto y la posición que ocupe dicho símbolo en un conjunto.

Todos los sistemas posicionales están basados en el **Teorema Fundamental de la Numeración (TFN)**, que sirve para relacionar una cantidad expresada en cualquier sistema de numeración con la misma cantidad expresada en el sistema decimal.

Viene dado por la fórmula siguiente: donde X es el valor absoluto del dígito en cuestión, i es la posición que ocupa el dígito con respecto al punto decimal y B es la base.

$$\sum X_i * B^i$$

SISTEMAS DE NUMERACIÓN USUALES EN INFORMÁTICA

En la representación interna se utiliza un código **binario natural** que es distinto del código de E/S.

También se utilizan los códigos octal y hexadecimal como **códigos intermedios** (los cuales son una simplificación por la que se representan secuencias de ceros y unos abreviadamente, que son más próximos a nuestro sistema decimal, y que permiten traducir rápidamente a y desde binario).

Representación posicional de los números

Un **sistema de numeración posicional** en base b usa un alfabeto de b símbolos distintos (o *cifras*), y cada posición tiene un peso específico. Así, cada número se representará como una secuencia de cifras, contribuyendo cada una de ellas con un valor que dependerá de:

- La cifra en sí.
- La posición de la cifra dentro de la secuencia.

$$N \equiv \dots n_4 n_3 n_2 n_1 n_0 . n_{-1} n_{-2} n_{-3} n_{-4} \dots$$

(Número expresado como secuencia de cifras, donde cada n_i pertenece al conjunto de símbolos).

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

(Valor numérico del número N interpretado en base b).

Ejemplo

Supongamos que la base b es 10. El conjunto de símbolos será: $\{0, \dots, 9\}$. Vemos que el número 3278,52 puede verse como:

$$3278,52 = 3000 + 200 + 70 + 8 + 0,5 + 0,02 = 3 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

La base 10 es la que estamos acostumbrados a utilizar. Pero puede utilizarse cualquier b .

Nosotros, en particular, estaremos interesados en las siguientes bases, sobre todo:

- Base 2 ($b=2$): **Sistema binario natural**. El alfabeto de símbolos será $\{ 0,1 \}$
- Base 8 ($b=8$): **Sistema octal**. El alfabeto de símbolos será $\{ 0,\dots,7 \}$
- Base 10 ($b=10$): **Sistema decimal**. El alfabeto de símbolos será $\{ 0,\dots,9 \}$
- Base 16 ($b=16$): **Sistema hexadecimal**. El alfabeto de símbolos será $\{ 0,\dots,9,A,\dots,F \}$

Sistema de numeración binario

Utilizará la base $b=2$ y, por tanto, el alfabeto de símbolos será $\{ 0,1 \}$. Veamos a continuación como pasar de binario a decimal y viceversa:

Conversión de binario a decimal:

Simplemente aplicaremos la fórmula vista anteriormente, tomando $b=2$.

Ejemplo

Obtener el valor decimal de $N=111000101,0011_2$ (Siendo ese 2 la base; utilizaremos a menudo esta notación de subíndice para indicar en qué base debemos interpretar la secuencia de números).

Cogemos la parte entera y de derecha a izquierda se determinan las posiciones:

$$N_{\text{entero}} = 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 45310.$$

Después cogemos la parte decimal:

$$N_{\text{decimal}} = 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0,187510.$$

El número final obtenido es la concatenación de las dos partes:

$$N = 453,187510.$$

Conversión de decimal a binario:

a) Parte entera : Simplemente vamos dividiendo por la base el número original, sin decimales (parte entera), y vamos repitiendo el procedimiento para los cocientes que vamos obteniendo.

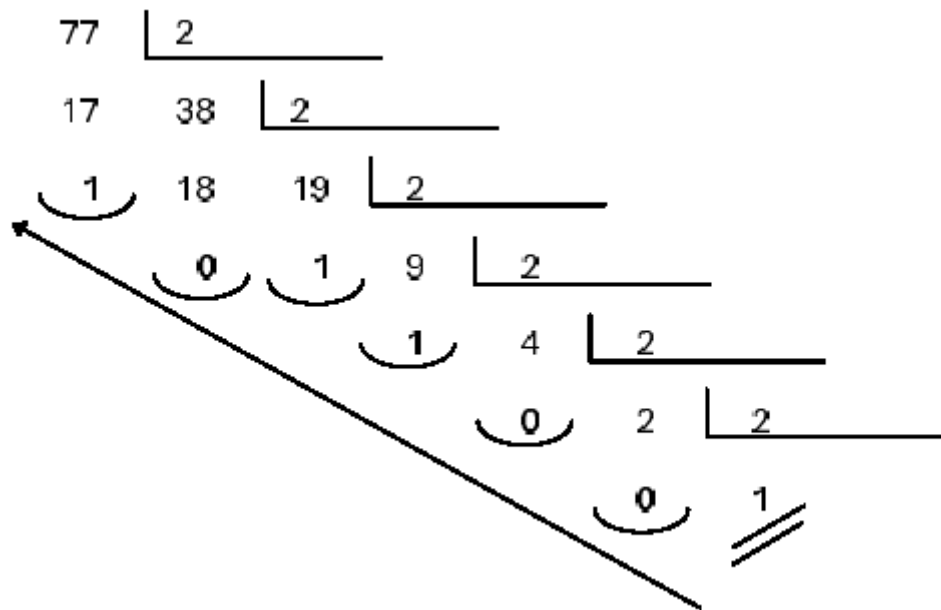
Los restos de estas divisiones y el último cociente son las cifras buscadas (observar que siempre deberán estar entre 0 y $b-1$). El último cociente es el dígito más significativo, y el primer resto el menos significativo.

b) Parte fraccionaria : Vamos multiplicando por la base la parte fraccionaria del número original, y sucesivamente repetimos el procedimiento con las partes fraccionarias de los números obtenidos. La secuencia de dígitos que vamos obteniendo es la representación en base b buscada de la parte fraccionaria del número.

Ejemplo

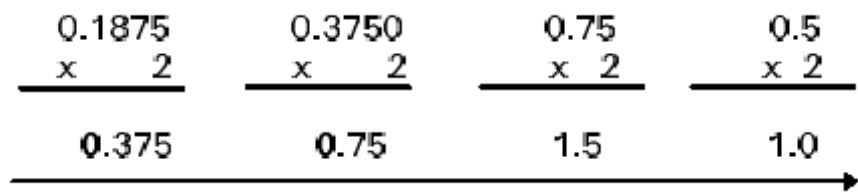
Vamos a pasar el número 77.1875_{10} a base 2 :

a) Cogemos la parte entera , vamos dividiendo por la base, y tenemos en cuenta los dígitos el resto, así como el último cociente:



Luego la parte entera sería $N_{entera} = 77_{10} = 1001101_2$

b) Ahora cogemos la parte decimal , la vamos multiplicando por la base y vamos teniendo en cuenta los dígitos enteros resultantes. Para el paso siguiente, obviamente, sólo tendremos en cuenta la parte fraccionaria resultante.



Se puede comprobar fácilmente que un número decimal con cifras fraccionarias puede dar lugar a un número binario con mayor número de cifras detrás de la coma decimal. Incluso infinitas cifras, como se observa en el siguiente ejemplo :

$$\begin{array}{ccccc}
 \begin{array}{r} 0.6 \\ \times 2 \\ \hline 1.2 \end{array} &
 \begin{array}{r} 0.2 \\ \times 2 \\ \hline 0.4 \end{array} &
 \begin{array}{r} 0.4 \\ \times 2 \\ \hline 0.8 \end{array} &
 \begin{array}{r} 0.8 \\ \times 2 \\ \hline 1.6 \end{array} &
 \begin{array}{r} 0.6 \\ \times 2 \\ \hline \dots \end{array} \\
 \hline
 & & & & \longrightarrow
 \end{array}$$

Es decir, $0.6_{10} = 0.1001_2$.(periódico) Vemos, pues, como un número con un sólo dígito de parte fraccionaria en base diez da lugar a un número con infinitos dígitos fraccionarios en base dos. Si un número binario se almacena en un computador con un número prefijado de bits, y por tanto finito, deberíamos recortar el número de cifras con la que lo representemos, a fin de guardarlo en dicho tamaño finito.

El error que se comete al hacer dicho recorte (por ejemplo, almacenar sólo 0.10011001 para representar un 0.6 en base diez, porque sólo reserváramos 8 bits para las partes fraccionarias de los números) se denomina **error de truncamiento**. Con una apariencia o con otra, dependiendo del tipo de representación elegida, estaremos cometiendo errores de este tipo cuando intentemos almacenar valores reales en el computador. En realidad, es un problema análogo al que hay en las calculadoras convencionales, que sólo permiten 8 ó 10 dígitos significativos, por ejemplo, y nos obligan a trabajar con números aproximados.

Operaciones aritméticas y lógicas con variables binarias

Estas son las tablas correspondientes a las **operaciones aritméticas** básicas:

Suma aritmética	Resta aritmética	Multiplicación	División
0 + 0 = 0	0 - 0 = 0	0 * 0 = 0	0 : 0 = indeterminado
0 + 1 = 1	0 - 1 = 1 (y debo 1)	0 * 1 = 0	0 : 1 = 0
1 + 0 = 1	1 - 0 = 1	1 * 0 = 0	1 : 0 = infinito
1 + 1 = 0 (y llevo 1)	1 - 1 = 0	1 * 1 = 1	1 : 1 = 1

Y aquí van algunos ejemplos de la mecánica de las operaciones aritméticas con valores expresados en binario. Observar que las operaciones están correctamente realizadas, cuando pasamos tanto los operandos como los resultados a decimal:

$ \begin{array}{r} 1011101 \text{ (93)} \\ + 1000101 \text{ (69)} \\ \hline 10100010 \text{ (162)} \end{array} $	$ \begin{array}{r} 1011101 \text{ (93)} \\ - 1000101 \text{ (69)} \\ \hline 0011000 \text{ (24)} \end{array} $	$ \begin{array}{r} 1101010 \text{ (106)} \\ - 1010111 \text{ (87)} \\ \hline 0010011 \text{ (19)} \end{array} $	$ \begin{array}{r} 1101010 \text{ (106)} \\ \times 101 \text{ (5)} \\ \hline 1101010 \\ 0000000 \\ 1101010 \\ \hline 1000010010 \text{ (530)} \end{array} $
---	---	--	--

$$\begin{array}{r}
 1101.01 \quad (13.25) \\
 - 101 \\
 \hline
 00110 \\
 - 101 \\
 \hline
 00110 \\
 - 101 \\
 \hline
 001...
 \end{array}
 \quad
 \begin{array}{r}
 101 \quad (5) \\
 \hline
 10.101... \quad (2.625) \text{ (inexacto, puesto que} \\
 \text{faltan dígitos.)}
 \end{array}$$

Códigos intermedios

Facilitan la labor de programación (trabajar en binario es muy laborioso y puede llevar a errores por cualquier cambio en la secuencia de ceros y unos). Se basan en la facilidad de transformar un número en base dos a otra base mayor que es potencia de dos. De esta manera, cada dígito de la nueva base se traducirá inmediatamente a una secuencia concreta de ceros y unos, pero nos servirá para expresarnos más abreviadamente. Existen dos tipos de códigos intermedios: octal y hexadecimal. Los estudiaremos con detenimiento:

Base octal ($b=8$) dígitos = $\{0,1,2,\dots,7\}$. Un dígito octal por cada tres binarios.

Conversión de binario a octal

Dado un número N , en binario, lo dividimos en ternas de tres valores; si el número de cifras no fuese un múltiplo entero de 3, las cifras que no formen parte de la terna se “rellenarían” con ceros hasta formar una terna, hacia la izquierda, si es en la parte entera (izquierda de la coma), o hacia la derecha, si es en la parte fraccionaria (derecha de la coma).

Ejemplo

$$N = 10110010,11100001_2 = \underset{2}{010} \underset{6}{110} \underset{2}{010}, \underset{7}{111} \underset{0}{000} \underset{2}{010}_2$$

Ahora, a cada terna le corresponde un dígito octal, aquel que resulta de interpretar cada terna en binario natural:

$$N = 262,702_8$$

Conversión de octal a binario

Se hace en sentido inverso: Cada dígito octal se sustituye por tres binarios:

Ejemplo

$$N = 376,3_8 = 011 \ 111 \ 110, 011_2$$

Conversión de octal a decimal

Se hace como se explicó antes, utilizando la fórmula de conversión.

Ejemplo

$$N = 376,3_{(8)} = 3 \cdot 8^2 + 7 \cdot 8^1 + 6 \cdot 8^0 + 3 \cdot 8^{-1} = 254,375_{(10)}$$

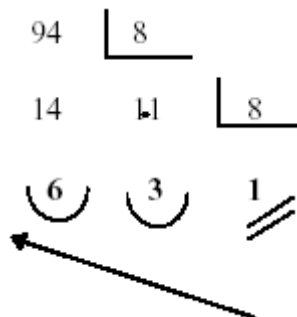
Conversión de decimal a octal

Se hace de la misma manera que cuando pasábamos de decimal a binario pero dividiendo y multiplicando ahora por la base $b=8$.

Ejemplo:

$$N=94,8125_{(10)}$$

La parte entera es $N_{\text{entera}}=94$



La parte decimal sería $N_{\text{decimal}}=0,8125$

$$\begin{array}{r} 0.8125 \\ \times 8 \\ \hline 6.5 \end{array} \qquad \begin{array}{r} 0.5 \\ \times 8 \\ \hline 4.0 \end{array}$$

Luego $N=136,64_{(8)}$

Conversión de binario a hexadecimal

Ahora, en vez de coger una terna, cogemos cuatro cifras tomando como punto de referencia la coma decimal (si la hay).

Ejemplo:

$$N = 10110111001101,101011 = 0010\ 1101\ 1100\ 1101, 1010\ 1100_{(2)}$$

Según la tabla que vemos a continuación, y que se deduce directamente de escribir en binario natural con cuatro dígitos los valores decimales 0 a 15 (o, lo que es lo mismo, en hexadecimal, 0 a F), sería $N = 2DCD,AC_{(16)}$.

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

De hexadecimal a binario

Seguimos el proceso inverso, sustituyendo cada dígito hexadecimal por sus cuatro binarios correspondientes:

Ejemplo:

$$N=27CB.0A_{(16)} = \underset{2}{0010} \underset{7}{0111} \underset{C}{1100} \underset{B}{1011}, \underset{0}{0000} \underset{A}{1010}_{(2)}$$

De hexadecimal a decimal

Se hace igual que en la base octal, y que en cualquier otra base : Siguiendo la fórmula.

Ejemplo:

$$N=23C,A_{(16)} = 2 \cdot 16^2 + 3 \cdot 16^1 + 12 \cdot 16^0 + 10 \cdot 16^{-1} \text{ (la A y C se sustituyen por su equivalente decimal).}$$

$$N=572,625_{(10)}$$

De decimal a hexadecimal: También de modo análogo a como lo hicimos en el octal:

Ejemplo:

$$N=572,6250.$$

La parte entera sería $N_{\text{entera}} = 572$

$$\begin{array}{r} 572 \quad | \quad 16 \quad \underline{\hspace{1cm}} \\ 92 \quad 35 \quad | \quad 16 \\ \hline \textcircled{12} \quad \textcircled{3} \quad \textcircled{2} \\ \swarrow \end{array}$$

La parte decimal $N_{\text{decimal}} = 0,6250$

$$\begin{array}{r} 0,6250 \\ \times 16 \\ \hline 10.0 \end{array}$$

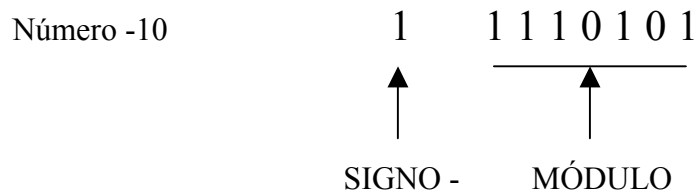
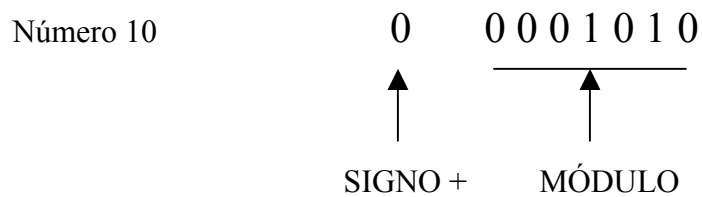
Es decir, el resultado final en hexadecimal es: $N=23C,A_{(16)}$

COMPLEMENTO A 1 (C-1)

Este sistema de representación utiliza el bit de más a la izquierda para el signo, correspondiendo el 0 para el signo + y el 1 para el signo -.

Para los números **positivos**, los n-1 bits de la derecha representan el módulo (igual que en el caso anterior).

El **negativo** de un número positivo se obtiene complementando todos sus dígitos (cambiando 0 por 1 y viceversa), incluido el bit de signo.



COMPLEMENTO A 2 (C-2)

Este sistema de representación utiliza el bit más a la izquierda para el signo, correspondiendo el 0 para el signo + y el 1 para el -.

En el caso de los números positivos, los n-1 bits de la derecha representan el módulo (igual en en los dos casos anteriores).

El negativo de un número se obtienen en dos pasos:

- Se complementa el número positivo en todos sus bits (cambiando los ceros por 1 y viceversa), incluido el bit de signo, es decir se realiza el complemento a 1.
 - Al resultado obtenido anteriormente ase le suma 1 (en binario) despreciando el último acarreo si existe.
-

La representación en C-2 de los números 10 y -10 sería:

Número 10 0 0 0 0 1 0 1 0
 ↑ ↑
 SIGNO + MÓDULO

Número -10

Primer paso 1 1 1 1 0 1 0 1

Segundo paso 1 1 1 1 0 1 0 1
 + 1

 1 1 1 1 0 1 1 0
 ↑ ↑
 SIGNO - MÓDULO

EXCESO A 2^{n-1}

Este método de representación no utiliza ningún bit para el signo, con lo cual todos los bits representan un módulo o valor.

Este valor se corresponde con el número representado más el exceso, que para n bits viene representado por 2^{n-1}

Por ejemplo, para 8 bits (n= 8) el exceso es de $2^{8-1} = 2^7 = 128$, con lo cual el número 10 vendrá representado por $10 + 128 = 138$ (en binario)

Para el caso del número -10 tendremos $-10 + 128 = 118$ (en binario)

Número 10: 1 0 0 0 1 0 1 0
 Número -10 0 1 1 1 0 1 1 0

IMPORTANCIA DE LA REPRESENTACIÓN EN COMPLEMENTOS

Se utiliza para que dentro de las máquinas no se tenga que tener más circuitería de la necesaria. La ventaja clara que se tiene es que las restas se realizarán como sumas, y por tanto nuestra ALU no tendrá que incorporar un restador (con un circuito sumador nos bastará para realizar tanto sumas como restas).

Complemento a la base menos uno (C1, si la base es 2):

Dado un número N, el complemento a la base menos uno 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.

Podremos entonces restar un número de otro simplemente sumando al minuendo el complemento a la base menos uno del sustraendo. La cifra que se arrastra del resultado se descarta, y se suma al resultado obtenido.

Ejemplo:

Supongamos que queremos restar 30-25. En base 2 (b=2) le haremos el C1 (esto es, la base menos uno) al número 25 (11001):

$$\begin{array}{r} 11111 \\ - \underline{11001} \\ \hline \end{array}$$

C1(11001) = 00110

Como lo que queremos es realizar una resta, una vez hallado el C1 del sustraendo, se le suma éste último al minuendo, y el último acarreo (en caso de que exista) se le suma al resultado final.

$$\begin{array}{r} 11110 \quad (30) \\ + \underline{00110} \quad (-25) \\ \hline \end{array}$$

$$1 \quad 00100$$

$$\underline{\quad +1}$$

$$00101 \quad (5)$$

Ahora le sumamos el acarreo (en negrita)

Resultado que coincide con el que hubiésemos obtenido con la resta normal.

Si no se hubiese producido el acarreo final, entonces no hay que sumar nada al resultado, pero lo que ocurre es que éste será negativo, y por tanto debemos complementarlo a uno para interpretarlo.

Por ejemplo, si quisiésemos haber hecho 25-30:

$$C1(11110) = 00001$$

$$\begin{array}{r} 11001 \quad (25) \\ + 00001 \quad (-30) \\ \hline 11010 \end{array}$$

Ahora no se produce acarreo final. Hallamos, pues, el C1 del resultado para interpretarlo como negativo

C1(11010) = 00101 (5) Esto es, el resultado de nuestra operación es -5.

Complemento a la base (C2, si la base es dos):

Dado un número N, el complemento a la base sería el número que resulta de restar cada una de las cifras del número N a la base menos uno y posteriormente sumar 1 a la diferencia obtenida. Esto es, es lo mismo que obtener el complemento a la base menos uno y después sumar uno al resultado.

Ejemplo:

El procedimiento para restar es muy parecido al anterior: Calcular el C2 (suponiendo b=2) del sustraendo, y añadirlo al minuendo:

$$\begin{array}{r} C2(11001) : \quad 11111 \\ \quad \quad \quad - \underline{11001} \\ \quad \quad \quad 00110 \\ \text{Sumar 1 a lo anterior :} \quad + \underline{\quad 1} \\ \quad \quad \quad 00111 \end{array} \quad \rightarrow \quad \text{Este es el complemento a 2 de 11001.}$$

Para realizar la resta 30-25 procedemos a sumar el C2(25) :

$$\begin{array}{r} 11110 \quad (30) \\ + \underline{00111} \quad (-25) \\ \hline 1 \quad 00101 \end{array}$$

Ahora simplemente descartamos el acarreo (no como antes, que había que sumarlo).

DECIMAL CODIFICADO EN BINARIO

En el sistema BCD cada cifra de un número decimal se representa por un conjunto de 4 bits, siendo la tabla de equivalencias entre ambos la siguiente

DECIMAL	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Por ejemplo para representas el 15 en BCD sería

1	5
0001	0101

Con lo que el 15 decimal sería el 00010101 en BCD

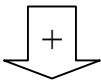
DECIMAL DESEMPAQUETADO

Un número decimal se repretetna de forma que cada una de sus cifras ocupa un octeto o byte.

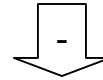
Cada uno de los octetos lleva en su cuarteto de la izquierda cuatro unos (1111) denominados bits de zona y en el cuarteto de la derecha, la codificación de la cifra en BCD, denominándose bits de dígito.

El cuarteto de la izquierda de la última cifra (cifra de la derecha) representa el [signo del número](#), siendo 1100 para el signo positivo (+) y 1101 para el signo negativo (-).

Ejemplo: número 1994

1111 0001	1111 1001	1111 1001	 1100	0100
_____	_____	_____	_____	_____
1	9	9	4	

Número -1994

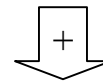


1111 0001	1111 1001	1111 1001	1101 0100
-----	-----	-----	-----
1	9	9	4

DECIMAL EMPAQUETADO

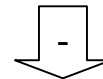
En este sistema de codificación se representa cada cifra decimal de un cuarteto (se eliminan los bits de zona), salvo el primer octeto de la derecha en el que su cuarteto también de la derecha lleva el signo con las mismas consideraciones que en el caso anterior).

Ejemplo: número 1994



0001	1001	1001	0100 1100
-----	-----	-----	-----
1	9	9	4

Número - 1994



0001	1001	1001	0100 1101
-----	-----	-----	-----
1	9	9	4

REPRESENTACIÓN EN COMA FLOTANTE

Para representar los números en coma flotante se utiliza la notación científica o exponencial matemática en la que una cantidad se representa de la forma:

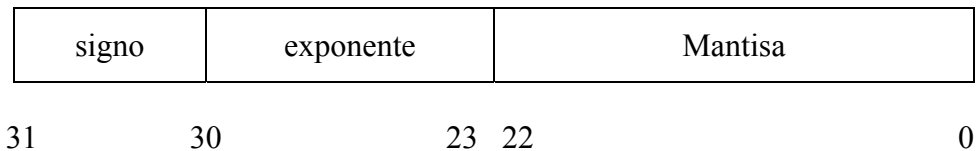
$$N^{\circ} = \text{mantisa} * \text{base de exponenciación}^{\text{exponente}}$$

Por ejemplo en base 10 la representación del número 25,4 sería:

$$0,254 \times 10^2$$

Tomaremos como norma que la mantisa no tiene parte entera y el primer dígito de la derecha es significativo (diferente de 0).

En este sistema de representación se divide los bits disponibles de la palabra del ordenador entre la mantisa, exponente y signo, como por ejemplo



EXPONENTE

Se representa en módulo y signo o en exceso a 2^{N-1} siendo siempre un número entero.

MANTISA

Es un número real con punto decimal implícito a la izquierda de sus bits, siendo representada generalmente en módulo y signo o complemento a 1 o complemento a 2.

BASE DE EXPONENCIACIÓN

Es una potencia de 2 determinada por el fabricante del ordenador.

Ejemplo:

Un ordenador utiliza el siguiente formato para representar los números en coma flotante:

- Los bits del 23 al 30 se utilizan para representar el exponente en exceso a 128 (2^7)
- Los bits del 0 al 22 se utilizan para representar la mantisa normalizada a C-1
- El bit 31 se utiliza para representar el signo de la mantisa (0 para +)
- La base de exponenciación es 2
- El 0 se representa con todos los bits a 0

Representar el número 12 en ese formato.

Primeramente tendremos que normalizar el número 12.

Tendremos que ir dividiendo entre las sucesivas potencias de 2 hasta que la parte entera sea 0.

$$12 = 12 \times 2^0 = 6 \times 2^1 = 3 \times 2^2 = 1,5 \times 2^3 = 0,75 \times 2^4$$

Por lo tanto tenemos que el exponente es 4

La representación del exponente en exceso a 128 es $(128 + 4) = 132$ que en binario (con 8 bits que es lo que tenemos del 23 al 30) es 10000100

La mantisa 0,75 la pasaremos a binario (recordar como se pasa la parte fraccionaria a binario multiplicando por 2) y nos queda 0,11.

Calculamos el complemento a 1 con lo que nos queda igual pero con el signo 0

Como tenemos 23 bits tendremos que rellenar por la derecha

11000000000000000000000

Exponente: 10000100

Mantisa: 11000000000000000000000

Signo: 0

0	10000100	11000000000000000000000
----------	-----------------	--------------------------------

Para el -12 sería similar:

$$-12 = -12 \times 2^0 = -6 \times 2^1 = -3 \times 2^2 = -1,5 \times 2^3 = -0,75 \times 2^4$$

El exponente en exceso a 128 será el mismo $(128+4) = 132 = 10000100$

La mantisa se pasa a complemento a 1 que como es negativo recordamos que había que complementar los bits (cambiar 1 por 0 y viceversa) y el signo será negativo (1)

$$0,75 = 11000000000000000000000$$

Complementándolo a 1 queda: 00111111111111111111111

1	10000100	00111111111111111111111
----------	-----------------	--------------------------------

CÓDIGOS DE E/S

Los sistemas de codificación alfanumérica sirven para representar una cantidad determinada de símbolos, en binario. A cada símbolo le corresponderá una combinación de un número de bits.

La asignación de códigos es arbitraria, y por tanto cada fabricante podría asignar una combinación diferente al mismo carácter. Para combatir el caos que ello provocaría, se crean códigos que normalicen esta situación, y que se aceptan entre toda la comunidad informática como estándares.

Veremos los más importantes :

Código BCD de intercambio normalizado (Standard Binary Code Decimal Interchange Code)

Utiliza 6 bits, por lo que puede asignar $2^6 = 64$ valores (no incluye letras minúsculas). A veces se añade un bit de paridad impar, para comprobar errores, con lo que se tiene una longitud de 7 bits, pero sólo 64 valores válidos .

El formato que siguen las palabras de éste código es el siguiente:

Bit de Paridad	Bits de Zona		Bits de Posición			
C	B	A	8	4	2	1
bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- **Bit de paridad (o de verificación)** : (Opcional) Sirve para detectar errores en los caracteres.
- **Bits de zona**: Sirve para distinguir entre un número y otro carácter (00 para los caracteres numéricos) .
- **Bits de posición**: para los valores numéricos, se codifica en código binario natural (excepto el cero, que se codifica como un 10, esto es, 1010_2).

Ejemplo:

Suponer un teclado en el que se pulsaran estos caracteres :

754.32 BEATRIZ

La secuencia de códigos que se generará

**0000111 1000101 1000100 0111011 1000011 0000010 1000000 0110010 1110101
0110001 0010011 0101001 1111001 0011001**

Código EBDIC (Extended Binary Code Decimal Interchange Code)

En este caso $n=8$, luego como $2^8=256$, hay 256 posibles combinaciones de ceros y unos (se codifican mayúsculas, minúsculas, números e incluso caracteres de control).

Formato :

Bits de Zona		Bits de Posición					
bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

- **Bits de zona:**
 - Si valen 00 : El carácter es de control.
 - Si valen 01 : Se trata de un carácter especial (no letra ni número).
 - Si valen 10 : Es un carácter en minúscula.
 - Si valen 11 : Es un carácter en mayúscula o numérico.
- **Bits de posición:**
 - Si los bits 2 y 3 valen 00 : Carácter en el rango A-I (o a-i).
 - Si los bits 2 y 3 valen 01 : Carácter en el rango J-R (o j-r).
 - Si los bits 2 y 3 valen 10 : Carácter en el rango S-Z (o s-z).
 - Si los bits 2 y 3 valen 11 : Carácter numérico.

Código ASCII (American Standard Code for Information Interchange)

Es el más ampliamente utilizado. Tiene una longitud de siete bits ($n=7$), a la que a veces se añade, en algunos sistemas, otro bit más (bien para comprobar errores mediante paridad, o bien para doblar el número de caracteres representables de 128 a 256, y así añadir un amplio conjunto de caracteres gráficos, por ejemplo, como es el caso del PC).

Otros códigos

Existen otros muchos códigos de E/S, si bien comienzan, afortunadamente, a caer en desuso. Como ejemplo tenemos el código **FIELDATA** que usaban los computadores Sperry-Univac de la serie 1100, hoy día Unisys.

Código ASCII de impresión de 8 bits (con caracteres gráficos) (PC)

izda. → dcha. ↓	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000 0	NUL 0		SP 32	0 48	@ 64	P 80	` 96	p 112	Ç 128	É 144	á 160	⋮ 176	L 192	⌚ 208	α 224	≡ 240
0001 1		DC1 17	! 33	l 49	A 65	Q 81	a 97	q 113	ü 129	æ 145	í 161	⋮ 177	⌚ 193	⌚ 209	β 225	± 241
0010 2		DC2 18	“ 34	2 50	B 66	R 82	b 98	r 114	é 130	Æ 146	ó 162	⋮ 178	⌚ 194	⌚ 210	Γ 226	≥ 242
0011 3	♥ 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115	â 131	ô 147	ú 163	 179	⌚ 195	⌚ 211	Π 227	≤ 243
0100 4	♦ 4	DC4 20	\$ 36	4 52	D 68	T 84	d 100	t 116	ä 132	ö 148	ñ 164	⌚ 180	— 196	⌚ 212	Σ 228	∫ 244
0101 5	♣ 5	§ 21	% 37	5 53	E 69	U 85	e 101	u 117	à 133	ò 149	Ñ 165	⌚ 181	⌚ 197	⌚ 213	σ 229	∫ 245
0110 6	♠ 6	& 22	& 38	6 54	F 70	V 86	f 102	v 118	ö 134	û 150	▲ 166	⌚ 182	⌚ 198	⌚ 214	μ 230	÷ 246
0111 7	BEL 7		' 39	7 55	G 71	W 87	g 103	w 119	ç 135	ù 151	∞ 167	⌚ 183	⌚ 199	⌚ 215	τ 231	≈ 247
1000 8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120	ê 136	ÿ 152	ı 168	⌚ 184	⌚ 200	⌚ 216	φ 232	° 248
1001 9	HT 9	EM 25) 41	9 57	I 73	Y 89	i 105	y 121	ë 137	ÿ 153	⌚ 169	⌚ 185	⌚ 201	⌚ 217	θ 233	· 249
1010 A	LF 10		* 42	: 58	J 74	Z 90	j 106	z 122	è 138	Û 154	⌚ 170	⌚ 186	⌚ 202	⌚ 218	Ω 234	· 250
1011 B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123	ï 139	ç 155	½ 171	⌚ 187	⌚ 203	⋮ 219	δ 235	√ 251
1100 C	FF 12	FS 28	' 44	< 60	L 76	\ 92	l 108	 124	î 140	£ 156	¼ 172	⌚ 188	⌚ 204	■ 220	∞ 236	η 252
1101 D	CR 13		- 45	= 61	M 77] 93	m 109	} 125	ì 141	¥ 157	ı 173	⌚ 189	— 205	⌚ 221	φ 237	² 253
1110 E	SO 14		· 46	> 62	N 78	^ 94	n 110	~ 126	Ä 142	Ŕ 158	« 174	⌚ 190	⌚ 206	⌚ 222	ε 238	■ 254
1111 F	SI 15		/ 47	? 63	O 79	_ 95	o 111	 127	À 143	f 159	» 175	⌚ 191	⌚ 207	■ 223	∩ 239	SP 255