Sistemas de representación de números[1]

Examinando la representación de enteros en el sistema de numeración decimal, podemos observar un principio importante: el principio de la *notación posicional*.

Consideremos el número 824. El uso de la notación posicional puede ser visto fácilmente si leemos este número en voz alta: "ochocientos veinticuatro". La posición de cada dígito en un momento determina su valor: el dígito menos significativo es el dígito situado más a la derecha y el más significativo es el situado más a la izquierda. Un examen indica que el número escrito **824** se representa, o es una forma abreviada para:

$$(8 * 100) + (2 * 10) + (4*1)$$

o bien

$$(8 * 10^2) + (2 * 10^1) + (4 * 10^0)$$

Análogamente, **98642** es una forma compacta de representar

$$(9 + 10^4) + (8 * 10^3) + (6 * 10^2) + (4 * 10^1) + (2 * 10^0)$$

La idea de utilizar la posición de un dígito para determinar su valor en un número escrito es bastante ingeniosa. Los primeros sistemas de numeración no seguían este principio: I es "uno" en números romanos, V es "cinco", X es "diez" y XVI es "dieciséis". MCXVI es "mil ciento dieciséis". Sólo reglas secundarias, tales como escribir IV para "cuatro" y VI para "seis" utilizan la posición para afectar el valor.

Nuestro sistema de representación de números está basado completamente en la utilización de la posición para determinar el valor. Los procedimientos que empleamos para sumar, restar, multiplicar y dividir números decimales están basados todos ellos en el uso de este principio, y la simplicidad y claridad que resultan de la notación posicional, más la gran eficiencia de estos procedimientos, han impuesto tan completamente la notación posicional, que apenas notamos su existencia.

El sistema de representación para enteros decimales, utilizando notación posicional, está basado en la regla de que el número escrito $a_m a_{m-1} \ldots a_2 a_1 a_0$ representa la suma:

$$(a_m * 10^m) + (a_{m-1} * 10^{m-1}) + \ldots + (a_2 * 10^2) + (a_1 * 10^1) + (a_0 * 10^0)$$

Así para **824** tenemos $8 = a_2$, $2 = a_1$ y $4 = a_0$; esto representa:

1 Bartee, Thomas: Introducción a los computadores. Capítulo 2: Representación de la información en un computador digital.

$$(8 * 10^2) + (2 * 10^1) + (4 * 10^0)$$

que es igual a:

$$(8 * 100) + (2 * 10) + (4 * 1)$$

De forma similar, **32963** tiene $3 = a_4$, $2 = a_3$, $9 = a_2$, $6 = a_1$ y $3 = a_0$; 32963 es la representación en notación posicional para

$$(3 * 10^4) + (2 * 10^3) + (9 * 10^2) + (6 * 10^1) + (3 * 10^0)$$

Este "desarrollo en suma de potencias" es denominado *desarrollo literal* del número. Así el desarrollo literal de **54** es

$$(5 * 10^1) + (4 * 10^0)$$

En la notación de un número decimal, cada dígito está escogido de un conjunto de diez símbolos, los cuales escribimos como 0, 1, 2, ..., 9, y la existencia de 10 diferentes dígitos o símbolos es el origen de la expresión *sistema de numeración decimal*. Sin embargo, un número mayor o menor de símbolos pueden ser usados en cada posición y, para cada elección, tendremos un sistema distinto de representación de números.

El más simple de estos sistemas es el sistema de *numeración binario*, el cual tiene sólo dos posibles dígitos en cada posición; los designamos como 0 y 1. La regla para la formación de un entero binario es escribir $a_m a_{m-1} \ldots a_1 a_0$, como abreviación de

$$(a_m * 2^m) + (a_{m-1} * 2^{m-1}) + \ldots + (a_1 * 2^1) + (a_0 * 2^0)$$

donde cada a es 0 o 1. Por ejemplo, el número binario 101 representa la suma de $(1 * 2^2) + (0 * 2^1) + (1 * 2^0)$, el cual es representado por el dígito 5 en el sistema decimal. De forma análoga, el número binario 11101 representa la suma

$$(1 * 2^4) + (1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)$$

que es igual a

$$(1 * 16) + (1 * 8) + (1 * 4) + (0 * 2) + (1 * 1)$$

y es representado por 29 en el sistema decimal.

Debido a que los símbolos 0 y 1 son usados en ambos sistemas, binario y decimal, números como 10 pueden ser binarios o decimales. Un sistema de notación comúnmente utilizado es

distinguir entre si números binarios y decimales, consiste en escribir la base del número como un subíndice después de él: así 10_2 es "2 en binario" y 10_{10} es "10 en decimal".

Estas son las representaciones binarias y decimales para los primeros veinte enteros:

$0_2 = 0_{10}$	$111_2 = 7_{10}$	$1110_2 = 14_{10}$
$1_2 = 1_{10}$	$1000_2 = 8_{10}$	$1111_2 = 15_{10}$
$10_2 = 2_{10}$	$1001_2 = 9_{10}$	$10000_2 = 16_{10}$
$11_2 = 3_{10}$	$1010_2 = 10_{10}$	$10001_2 = 17_{10}$
$100_2 = 4_{10}$	$1011_2 = 11_{10}$	$10010_2 = 18_{10}$
$101_2 = 5_{10}$	$1100_2 = 12_{10}$	$10011_2 = 19_{10}$
$110_2 = 6_{10}$	$1101_2 = 13_{10}$	$10100_2 = 20_{10}$

El sistema de notación posicional utilizado para enteros decimales se extiende fácilmente a las fracciones decimales y, de forma similar, a las fracciones de otras bases.

La fracción **0.264** tiene como desarrollo literal la suma

$$(2 * 10^{-1}) + (6 * 10^{-2}) + (4 * 10^{-3})$$

La fracción **0.23** tiene como desarrollo la suma de

$$(2 * 10^{-1}) + (3 * 10^{-2})$$

En general, la fracción decimal $0.a_{-1}a_{-2} \ldots a_{-m}$ tiene como desarrollo literal la suma

$$(a_{-1} * 10^{-1}) + (a_{-2} * 10^{-2}) + \ldots + (a_{-m} * 10^{-m})$$

Así para **0.12637**, tenemos $a_{-1}=1$, $a_{-2}=2$, $a_{-3}=6$, $a_{-4}=3$, $a_{-5}=7$, el desarrollo literal es

$$(1 * 10^{-1}) + (2 * 10^{-2}) + (6 * 10^{-3}) + (3 * 10^{-4}) + (7 * 10^{-5})$$

El mismo principio puede ser utilizado para fracciones binarias. En su desarrollo literal aparecerán potencias negativas de 2, en vez de potencias negativas de 10.

Por ejemplo, **0.1011** tiene como desarrollo literal la suma

$$(1 * 2^{-1}) + (0 * 2^{-2}) + (1 * 2^{-3}) + (1 * 2^{-4})$$

Esto es

$$(1 * 1/2) + (0 * 1/4) + (1 * 1/8) + (1 * 1/16)$$

en decimal. La fracción binaria **0.11101** tiene como desarrollo literal la suma de

$$(1 * 2^{-1}) + (1 * 2^{-2}) + (1 * 2^{-3}) + (0 * 2^{-4}) + (1 * 2^{-5})$$

En general, la fracción binaria $0.a_{-1}a_{-2} \ldots a_{-m}$ tiene como desarrollo literal la suma

$$(a_{-1} * 2^{-1}) + (a_{-2} * 2^{-2}) + \ldots + (a_{-m} * 2^{-m})$$

Así, para la fracción binaria **0.1011** tenemos $a_{-1}=1$, $a_{-2}=0$, $a_{-3}=1$ y $a_{-4}=1$, y el desarrollo literal es

$$(1 * 2^{-1}) + (0 * 2^{-2}) + (1 * 2^{-3}) + (1 * 2^{-4})$$

(Esto, en decimal, tiene el valor 1/2 + 1/8 + 1/16 = 11/16).

Señalamos las siguientes correspondencias:

$0.1_2 = 0.5_{10}$ $\qquad$ $0.11_2 = 0.75_{10}$

$0.01_2 = 0.25_{10}$ $\qquad$ $0.001_2 = 0.125_{10}$

Operaciones aritméticas en el sistema binario

En esta sección pondremos nuestra atención en las operaciones aritméticas en el sistema de numeración binario, ya que es en este sistema que la mayoría de los computadores realizan sus operaciones.

Las operaciones de adición, sustracción, multiplicación y división, son inmediatas en el sistema de numeración binario. Para la adición basta aprender la suma de sólo cuatro combinaciones de ceros y unos; éstas son:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, más un acarreo de 1

La adición de números binarios puede ser efectuada columna por columna, como en el sistema decimal, llevando los acarreos en la misma forma.

101_2	1100_2	110101_2
$+ 1011_2$	$+ 1110_2$	$+ 1110_2$
10000_2	11010_2	1000011_2

Para comprobar la primera suma anterior, nótese que $101_2 = 5_{10}$ y $1011_2 = 11_{10}$ y que la suma de éstos dos enteros es $10000_2 = 16_{10}$. Las otras dos sumas pueden ser comprobadas en forma similar.

Análogamente, existen cuatro declaraciones para la sustracción:

0 - 0 = 0

0 - 1 = 1 con un descuento de 1, del próximo dígito más significativo en el minuendo

1 - 0 = 1

1 - 1 = 0

Aquí hay cuatro ejemplos de sustracciones. El "descuento" es manejado como con los decimales. Análogamente, las diferencias pueden ser comprobadas por conversión a números decimales.

101.1_2	110.1_2	111.01_2	1010.10_2
$- 100.1_2$	$- 101.0_2$	$- 11.00_2$	$- 1010.11_2$
1.0_2	1.1_2	100.01_2	-0.01_2

Las siguientes cuatro reglas serán suficientes para la multiplicación de dígitos binarios.

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

La multiplicación puede ser realizada utilizando el mismo procedimiento que para el sistema decimal. Sin embargo, para el binario simplemente se copia el multiplicando si el dígito multiplicador es 1, tal como se deduce de las reglas indicadas anteriormente.

101_2 *
110_2
000
101
101
11110_2

110_2 *
111_2
110
110
110
101010_2

110.101_2
11.01_2
110101
000000
110101
110101
10101.10001_2

La división puede realizarse de la manera tradicional que se hace con los decimales. En este caso, sin embargo, cada dígito cociente puede ser solamente 0 o 1, lo que simplifica mucho el proceso.

11110_2	$\vert 100_2$
100	111
111	
100	
110	
100	
10	

1111101.01_2	$\vert 110.1_2$
1101	100.01
10101	
1101	
10000	
1101	
11100	
1101	
1111	

Representación de números decimales codificados en binario

Los componentes de los circuitos electrónicos empleados para construir computadoras digitales son, por lo que respecta a su modo de operar, intrínsecamente binarios, como lo son los elementos de memoria utilizados para almacenar información y, por tanto, el sistema de numeración binario es el más lógico para un computador.

Por otro lado, el sistema decimal es el que utiliza la gente, y existe una resistencia natural a razonar los cálculos en un sistema de numeración binario. Asimismo, como los cheques, facturas, impuestos, precios, etc. figuran todos ellos en sistema decimal, los valores de la mayor parte de los datos de entrada deben ser convertidos del sistema decimal, antes de que empiecen los cálculos. Análogamente, los datos binarios deben ser convertidos a decimal, en casi todos los casos, antes de que puedan ser impresos o enviados a la salida.

Por estas y otras razones, la mayor parte de las primera máquinas trabajaban en sistemas de numeración decimal codificado en binario. En estos sistemas un grupo codificado de bits (*binary digit*), es utilizado para representar cada uno de los 10 dígitos decimales. Por ejemplo, un código obvio y natural es el simple "código binario ponderado", que se muestra en la tabla 2.1

Tabla 2.1 Código BCD (8,4,2,1)

Código binario	Dígito decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Este tipo de representación no es conveniente, ya que cada dígito decimal requiere de 4 bits. Así, representar el valor decimal 123 requiere de 12 dígitos binarios, y el valor 6541 requerirá de 16 bits.

Por otro lado, con 4 bits es posible obtener (2^4) 16 valores posibles, y se estarán utilizando solo 10. Utilizar 3 bits tampoco es una opción, ya que solo tendremos 8 valores posibles (2^3).

Sistemas numéricos octal y hexadecimal

Otros dos tipos numéricos ampliamente conocidos y utilizados en la ciencia de las computadoras son, el sistema octal y el hexadecimal.

El sistema octal es de base 8, utilizándose ocho símbolos distintos para representar números: normalmente éstos son: 0, 1, 2, 3, 4, 5, 6 y 7.

Los primeros dieciocho números octales y sus equivalentes decimales se muestran en la tabla 2.2

Tabla 2.2 Números en octal

Octal	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
10	8

Octal	Decimal
11	9
12	10
13	11
14	12
15	13
16	14
17	15
20	16
21	17

Para convertir un número octal a número decimal, utilizamos el mismo tipo de expresión literal que en el caso binario, excepto que ahora la base es 8 en vez de 2.

Por consiguiente, **1213** en octal es

$$(1 * 8^3) + (2 * 8^2) + (1 * 8^1) + (3 * 8^0)$$

que es igual a:

$$512 + 128 + 8 + 3 = 651$$

en decimal.

Asimismo, **1.123** en octal es equivalente a

$$(1 * 8^0) + (1 * 8^{-1}) + (2 * 8^{-2}) + (3 * 8^{-3})$$

o igual a:

$$1 + 1/8 + 2/64 + 3/512 = 1\frac{83}{512}$$

en decimal.

Existe un truco simple para convertir un número binario en octal. Consiste en agrupar simplemente los dígitos binarios en conjuntos de tres, partiendo del punto octal, y leer cada grupo de acuerdo con la tabla 2.3

Vamos a convertir el número binario 011101. Primero lo dividimos en grupos de tres (así 011 101), luego convertimos cada grupo de tres dígitos binarios; obtenemos 35 en octal. Por tanto 011101 binario = 35 octal.

Tabla 2.3 Conversión binario-octal

Tres dígitos binarios	**Dígito octal**
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Ejemplos:

$110110101_2 = 665_8$

$11011_2 = 33_8$

$1001_2 = 11_8$

$10101.11_2 = 25.6_8$

$1100.111_2 = 14.7_8$

$1011.1111_2 = 13.74_8$

Un empleo importante para el octal está en el listado de programas y para vaciados de memoria en máquinas binarias, con lo que tenemos listados más compactos. También es preferible emplear caracteres octal a emplear caracteres binarios para seguir la secuencia del contenido de los diferentes registros, y en el contenido oral del contenido de un registro binario.

El sistema numérico hexadecimal es útil por razones análogas. En gran parte de computadoras, las memorias están organizadas en "*bytes*" consistentes de 8 dígitos binarios. Cada byte es utilizado como una individualidad para representar un solo carácter alfanumérico o es dividido en dos grupos de 4 bits.

Cuando los bytes son manejados en dos grupos de 4 bits, estos son medio-bytes de 4 bits denominados frecuentemente "*nibbles*", y el programador tiene la opción de declarar cada carácter de 4 bits como una parte de un número binario o como dos cifras decimales-codificadas-en-binario. Por ejemplo, el byte 00011000 puede ser declarado como un número binario, en cuyo caso es equivalente a 24 en decimal, o dos caracteres decimal-codificado-en-binario, en cuyo caso representa el número 18.

Tabla 2.5 Dígitos Hexadecimales

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

Cuando el computador maneja números en binario, pero en grupos de 4 dígitos, es conveniente tener un código para representar cada uno de estos conjuntos de 4 dígitos. Como pueden ser representados 16 posibles números diferentes, necesitamos un sistema numérico de base 16: éste es denominado sistema numérico hexadecimal. Como los dígitos del 0 al 9 no son suficientes se utilizan también las letras A, B, C, D, E y F (véase la tabla 2.5).

Para pasar de binario a hexadecimal, simplemente se divide en número binario en grupos de 4 dígitos y convertimos cada grupo de 4 bits de acuerdo al código precedente. Así, 101110112 = BB16, 100101012 = 9516, 110001112 = C716, y 100010112 = 8B16.

La conversión de hexadecimal a decimal es directa, pero algo lenta. Por ejemplo, **BB** representa

$$(B * 16^1) + (B * 16^0) = (11 * 16) + (11 * 1) = 176 + 11 = 187.$$

Análogamente,

$AB6_{16}$ $= (A * 16^2) + (B * 16^1) + (6 * 16^0)$

$= (10 * 256) + (11 * 16) + (6 * 1)$

$= 2560 + 176 + 6 = 2742$