Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Formación Básica
Dpto. de Matemática
Carrera de : Ingeniería Civil, Electricista, Electrónica, Industrial, Mecánica y Agrimensura

INFORMATICA I

Autor: Jorge Di Marco

Sistemas de Numeración - Representación Interna

Sistemas de Numeración: Sistemas Numéricos Posicionales

La necesidad de representar conjuntos de objetos ha llevado a las distintas culturas a adoptar diversas formar de simbolizar su valor numérico.

Una primer manera de representar el número de elementos que constituyen un cierto conjunto, es establecer una correspondencia con un número igual de símbolos.

Esto lo hacemos cuando lo contamos con los dedos, como ser los días de la semana, dibujamos igual número de trazos: **I I I I I I I**

Tal sistema sería "*unario*", pues utiliza un sólo tipo de símbolo. Su desventaja es que no permite simbolizar cómoda y rápidamente conjuntos con muchos elementos.

Para poder simbolizar conjuntos de muchos elementos será necesario definir mayor cantidad de símbolos. A su vez, a fin de minimizar esta cantidad, se dedinen una cantidad de operaciones implícitas entre los mismos.

Los romanos utilizaron un sistemas de signos de valor creciente: **I, V, X, L, C, D, M,** etc., que se agrupaban de derecha a izquierda, sumándose o restándose entre sí, según siguieran o no el orden creciente:

CXVII = cien + diez + cinco + uno + uno

MCMV = mil + (mil – cien) + cinco

Esta codificación requería nuevos símbolos cuando se agotaban los de mayor valor, a la par que los cálculos convenía realizarlos con ábacos.

Fueron los pueblo orientales y americanos (mayas) los que desarrollaron los sistemas **posicionales**, basados en un conjunto limitado y constante de símbolos, entre los cuales se encontraba el *cero*, para indicar ausencia de elementos.

En estos sistemas, cada símbolo, además del número de unidades que representa considerado en forma aislada, tiene un significado o peso distinto según la posición que ocupa en el grupo de caracteres del que forma parte.

De esta manera es posible representar sistemáticamente cualquier número, empleando en forma combinada un conjunto limitado de caracteres.

Relacionado con los diez dedos, el sistema posicional tendría 10 elementos diferentes (que forman una sucesión monótona creciente 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) y se denomina "sistema posicional decimal" o "sistema base diez". Los caracteres (elementos) se denominan "***dígitos***", y constituyen piezas de información digital.

Los sistemas digitales actúan bajo el control de variables discretas, entendiéndose por éstas, las variables que pueden tomar un número finito de valores. Por ser de fácil realización los componentes físicos con dos estados diferenciados.

Tanto si se utilizan en procesamiento de datos como en control industrial, los sistemas digitales han de realizar operaciones con números discretos. Los números pueden representarse en diversos sistemas de numeración, que se diferencian por su base.

La base de un sistema de numeración es el número de símbolos distintos utilizados para la representación de las cantidades del mismo. El sistema de numeración utilizado en la vida cotidiana es el de base diez, en el cual existen diez símbolos distintos, del 0 al 9.

En sistemas digitales se trabaja con un sistema de numeración de base dos, denominado sistema binario.

Representación de los números

En un sistema de base ***b***, un número ***N*** cualquiera se puede representar mediante un polinomio de potencias de la base, multiplicados por un símbolo perteneciente al sistema.

En general tendremos:

$$N = a_n.b^n + a_{n-1}.b^{n-1} + ... + a_i.b^i + ... + a_0.b^0 + a_{-1}.b^{-1} + ...a_{-p}.b^{-p} \qquad (1.1)$$

Siendo ***b*** la base del sistema de numeración y a_i un número perteneciente al sistema y que, por tanto, cumple con la siguiente condición: $0 \le a_i < b$

donde ***n+1*** y ***p*** representan respectivamente el número de dígitos enteros y fraccionarios.

En nuestra práctica habitual, representamos los distintos números enteros o reales *(en realidad racionales, pues sólo consideramos una cantidad finita de cifras)* mediante **el sistema de numeración decimal** o de base 10.

Este sistema emplea diez cifras distintas o dígitos: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, a los que pone en correspondencia con el cero y los primeros nueve números naturales.

De acuerdo con todo esto, cualquier otro número puede representarse mediante una determinada cadena de dígitos, donde la contribución de cada uno depende no sólo de su propio valor, sino de la posición que ocupa en la cadena, de manera que el dígito está multiplicado por una potencia de la base (10 en este caso).

Por ejemplo:

$$623 = 6 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

$$87{,}54 = 8 \times 10^1 + 7 \times 10^0 + 5 \times 10^{-1} + 4 \times 10^{-2}$$

formalizando esto, obtenemos las siguientes expresiones:

Valores Enteros	**Valores Reales**
$i = s \times \sum_{k=1}^{q} w_k \times r^{k-1}$ donde: ***i*** es el valor entero ***q*** es el número de dígitos	$x = s \times b^e \times \sum_{k=1}^{p} f_k \times b^{-k}$ donde: ***x*** es el valor real ***s*** en el signo (+1 o –1)

r w_k es un número no negativo menor que r	***b*** es la base (entero mayor que 1) ***p*** en el número de dígitos de la mantisa (entero mayor que 1)

Otros Sistemas

De acuerdo a lo expuesto en las ecuaciones anteriores puede entonces definirse sistemas de numeración distintos del decimal, cuya base sea un número natural cualquiera >1.

Sistema Binario

Como ya hemos expuesto, este sistema utiliza solamente dos símbolos distintos que se representan gráficamente por **0** y **1** y reciben el nombre de ***bit***. La utilización casi exclusiva de este sistema de numeración en los equipos de cálculo y control automático es debida a la seguridad y rapidez de respuesta de los elementos físicos que poseen dos estados diferenciados y a la sencillez de las operaciones aritméticas en este *sistema (para representar una misma cantidad)* que en los sistemas cuya base es mayor de dos.

Sistema Hexadecimal

El sistema hexadecimal es el de base dieciséis, es decir para la representación de las cantidades utiliza dieciséis símbolos diferentes que son los dígitos del **0** al **9** y las letras del alfabeto de la **A** a la **F**. El interés de este sistema, es debido a que 16 es una potencia de 2 (2^4=16) y por lo tanto resulta muy sencilla la conversión de los números del sistema binario al hexadecimal y viceversa, permitiendo una representación compacta de cantidades binarias.

Tabla 1 - Equivalencia en los distintos Sistemas

Sistema Decimal	Sistema Binario	Sistema Hexadecimal	Sistema Decimal	Sistema Binario	Sistema Hexadecimal
0	0 0 0 0	0	**8**	1 0 0 0	8
1	0 0 0 1	1	**9**	1 0 0 1	9
2	0 0 1 0	2	**10**	1 0 1 0	A
3	0 0 1 1	3	**11**	1 0 1 1	B
4	0 1 0 0	4	**12**	1 1 0 0	C
5	0 1 0 1	5	**13**	1 1 0 1	D
6	0 1 1 0	6	**14**	1 1 1 0	E
7	0 1 1 1	7	**15**	1 1 1 1	F

Conversión a otro Sistema

Base *b* a decimal

De acuerdo con lo dicho hasta ahora, resulta inmediato hallar la representación decimal de un número expresado en otra base cualquiera ***b***.

Binario a Decimal, ej.:

$$10110_2 = 1 \text{ x } 2^4 + 0 \text{ x } 2^3 + 1 \text{ x } 2^2 + 1 \text{ x } 2^1 + 0 \text{ x } 2^0 = 22_{10}$$

$$11.01_2 = 1 \text{ x } 2^1 + 1 \text{ x } 2^0 + 0 \text{ x } 2^{-1} + 1 \text{ x } 2^{-2} = 3.25_{10}$$

Hexadecimal a decimal, ej.:

$$1AF_{16} = 1 \text{ x } 16^2 + 10 \text{ x } 16^1 + 15 \text{ x } 16^0 = 431_{10}$$

$$2B.E_{16} = 2 \text{ x } 16^1 + 11 \text{ x } 16^0 + 14 \text{ x } 16^{-1} = 43.0625_{10}$$

Hexadecimal a Binario y Binario a Hexadecimal

Ya se aclaró que el interés sobre el sistema hexadecimal es debido a que 16 (su base) es una potencia de 2 ($2^4=16$), y por lo tanto resulta muy sencilla la conversión de los números del sistema binario al hexadecimal y viceversa.

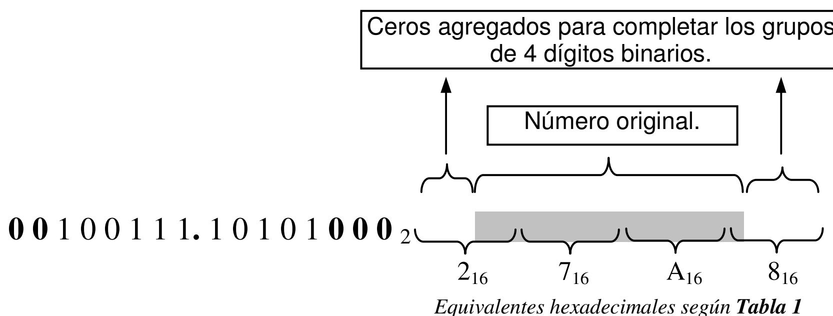
Para convertir un número del sistema hexadecimal al binario se sustituye cada símbolo por su equivalente en binario indicado en la **Tabla 1**. Sea por ejemplo el número $9A7E_{16}$. El equivalente a cada símbolo es:

$$8_{16} = 1\,0\,0\,0_2$$
$$1_{16} = 0\,0\,0\,1_2$$
$$A_{16} = 1\,0\,1\,0_2$$
$$E_{16} = 1\,1\,1\,0_2$$

Por lo tanto resulta: $81AE_{16} = \underbrace{1\,0\,0\,0}_{8}\,\underbrace{0\,0\,0\,1}_{1}\,\underbrace{1\,0\,1\,0}_{A}\,\underbrace{1\,1\,1\,0}_{E}{}_2$

La conversión de un número del sistema binario al hexadecimal se realiza a la inversa, agrupando los bits enteros y fraccionarios en grupos de cuatro a partir del punto decimal y convirtiendo cada grupo independientemente. Para completar el último grupo se añaden los ceros que sean necesarios.

Sea por ejemplo el número 100111.10101_2 ; debemos completarlo con dos ceros a la izquierda y tres a la derecha:



*Equivalentes hexadecimales según **Tabla 1***

Resultando por lo tanto: $1\,0\,0\,1\,1\,1.\,1\,0\,1\,0\,1_2 = 27.A8_{16}$

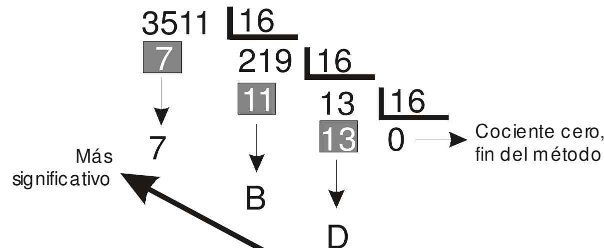
Otro ejemplo: $-1\,1.\,0\,1_2 = -\underbrace{0\,0\,1\,1}_{3}.\underbrace{0\,1\,0\,0}_{4}{}_2 = -3.4_{16}$

Decimal a otro sistema

El pasaje de un número decimal a otra base se realiza analizando por separado la parte entera (prescindiendo del signo) y la parte fraccionaria.

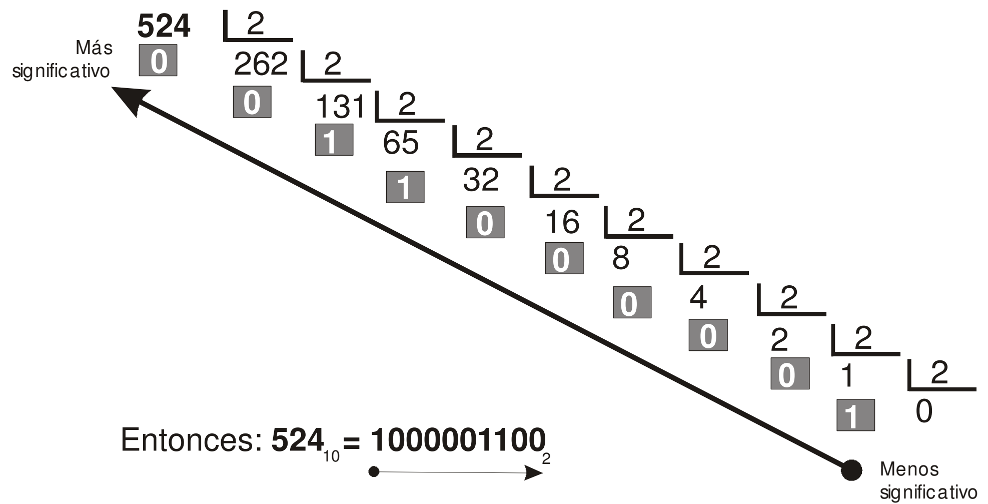
<u>Parte Entera:</u> Se divide el número por la base a la que se desea convertir (división entera); luego se divide el cociente obtenido por la base, y así sucesivamente hasta que el cociente sea cero. Los restos enteros de cada división, que son siempre menores que la base, son los dígitos de la nueva representación del número, ordenados del menos significativo al más significativo.

<u>Ejemplo 1:</u> Representar en Hexadecimal el número 3511_{10}



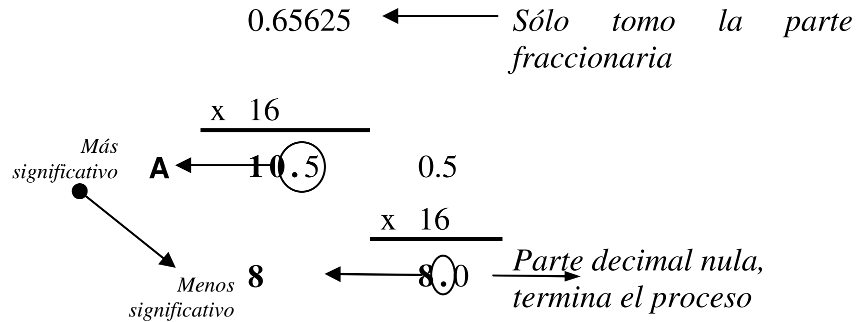
Entonces: $\mathbf{3511_{10} = DB7_{16}}$

<u>Ejemplo 2:</u> Representar en Binario el número 524_{10}



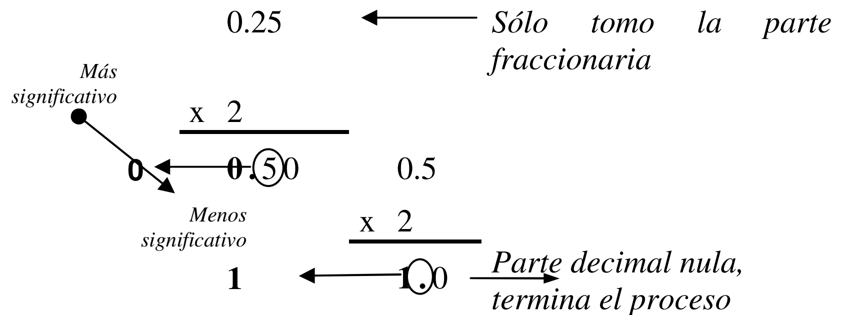
Parte Fraccionaria: Se multiplica la parte fraccionaria por la base a la que se desea convertir. La parte entera del resultado, que siempre es menor que la base, será el primer dígito fraccionario de la nueva representación. Se multiplica ahora la parte fraccionaria del resultado por la base y así sucesivamente.

Ejemplo 1: Representar en Hexadecimal el número 7.65625_{10}



Luego, $7_{10} = 7_{16}$ (que es la parte entera del número que estamos convirtiendo), entonces **$7.65625_{10} = 7\,.\,A\,8_{16}$**

Ejemplo 2: Representar en Binario el número 13.25_{10}



Luego, $13_{10} = 1101_2$ (que es la parte entera del número que estamos convirtiendo), entonces **$13.25_{10} = 1101.01_{16}$**

Observación: Existen racionales cuya representación decimal tiene una cantidad finita de cifras, pero que en otro sistema pueden resultar periódicos. Veamos el caso de convertir a binario el número 0.1_{10}

0.1			
0.2	→	**0**	⟶ *Más significativo*
0.4	→	**0**	
0.8	→	**0**	
1.6	→	**1**	
1.2	→	**1**	
0.4	→	**0**	
0.8	→	**0**	
1.6	→	**1**	
1.2	→	**1**	
0.4	→	**0**	⟶ *Menos significativo*
⋮	⋮	⋮	

Entonces:

$$0.1_{10} = 0.\underbrace{0001}_{1}\underbrace{1001}_{9}\underbrace{1001}_{9}\underbrace{1001}_{9}\ldots{}_2 = 0.1999\ldots{}_{16}$$

Si se debe almacenar dígitos de estas nuevas representaciones en una cantidad finita de casilleros, necesariamente ocurrirán errores de truncamiento.

Operaciones Aritméticas

Las operaciones aritméticas entre números representados en una base cualquiera siguen las reglas similares a las de la aritmética decimal.

La suma binaria toma el valor uno cuando uno solo de los sumandos tiene dicho valor. Cuando ambos sumandos tienen el valor uno, la suma es cero y se produce un acarreo (me llevo 1).

$$\begin{array}{r} 0 \\ +\,0 \\ \hline \mathbf{0} \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline \mathbf{1} \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline \mathbf{1} \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline \boxed{\mathbf{1}}\,\mathbf{0} \end{array}$$

acarreo (el 1 recuadrado)

<u>Ejemplo:</u> Suma Aritmética

Decimal		Binario
22	→	10110
+ 19	→	+ 10011
41	→	**101001**

Representación Interna de la Información en los Sistemas de Computación

Hemos visto que la información que maneja una computadora digital, ya sean datos numéricos, alfanuméricos o instrucciones de programas, sólo puede ser almacenada y procesada internamente en *“casilleros”* elementales llamados ***bits***. Un bit puede tomar uno y sólo uno de dos estados posibles, denominados por convención estado **0** y estado **1**. De aquí la necesidad de codificar la información empleando el sistema binario de numeración (cadenas de ceros y unos). El sistema hexadecimal permite visualizar en forma más compacta esta información, dada la rapidez de conversión al binario, pero teniendo presente que la máquina no lo maneja directamente.

Representación de Caracteres

La convención habitual es asignar un número a cada carácter (letra, número, signo de puntuación u operación, etc.) y almacenar la forma binaria de este número en un ***byte*** u octeto, es decir un conjunto de 8 bits consecutivos. Según vimos, el byte es la unidad mínima accesible en forma directa.

Ejemplo de byte:

0	**1**	**0**	**0**	**1**	**1**	**0**	**0**

De acuerdo a esto, en un byte es posible almacenar números binarios que comprenden los códigos que van desde el 0 hasta el 255 (256 códigos en total).

Byte									**Decimal**		**Hexa**
0	0	0	0	0	0	0	0	→	0	→	0
0	0	0	0	0	0	0	1	→	1	→	1
0	0	0	0	0	0	1	0	→	2	→	2
0	0	0	0	0	0	1	1	→	3	→	3
0	0	0	0	0	1	0	0	→	4	→	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮
1	1	1	1	1	1	0	1	→	253	→	FD
1	1	1	1	1	1	1	0	→	254	→	FE
1	1	1	1	1	1	1	1	→	255	→	FF

Quiere decir que hay 256 caracteres posibles. Los distintos códigos establecen la correspondencia: Nº → carácter.

Uno de los más usados es el ASCII (código americano estándar para el intercambio de información). Otro es el EBCDIC. En general los códigos normalizan menos de 256 caracteres; los restantes quedan disponibles y pueden a veces ser definidos por el usuario.

Tabla de Códigos ASCII (primeros 128 códigos)

0	·	**32**	espacio	**64**	@	**96**	`	**16**	·	**48**	0	**80**	P	**112**	p
1	·	**33**	!	**65**	A	**97**	a	**17**	·	**49**	1	**81**	Q	**113**	q
2	·	**34**	"	**66**	B	**98**	b	**18**	·	**50**	2	**82**	R	**114**	r
3	·	**35**	#	**67**	C	**99**	c	**19**	·	**51**	3	**83**	S	**115**	s
4	·	**36**	$	**68**	D	**100**	d	**20**	·	**52**	4	**84**	T	**116**	t
5	·	**37**	%	**69**	E	**101**	e	**21**	·	**53**	5	**85**	U	**117**	u
6	·	**38**	&	**70**	F	**102**	f	**22**	·	**54**	6	**86**	V	**118**	v
7	·	**39**	'	**71**	G	**103**	g	**23**	·	**55**	7	**87**	W	**119**	w
8	* *	**40**	(	**72**	H	**104**	h	**24**	·	**56**	8	**88**	X	**120**	x
9	* *	**41**	)	**73**	I	**105**	i	**25**	·	**57**	9	**89**	Y	**121**	y
10	* *	**42**	*	**74**	J	**106**	j	**26**	·	**58**	:	**90**	Z	**122**	z
11	·	**43**	+	**75**	K	**107**	k	**27**	·	**59**	;	**91**	[	**123**	{
12	·	**44**	,	**76**	L	**108**	l	**28**	·	**60**	<	**92**	\	**124**	|
13	* *	**45**	-	**77**	M	**109**	m	**29**	·	**61**	=	**93**	]	**125**	}
14	·	**46**	.	**78**	N	**110**	n	**30**	·	**62**	>	**94**	^	**126**	~
15	·	**47**	/	**79**	O	**111**	o	**31**	·	**63**	?	**95**	_	**127**	·

· Microsoft Windows no admite estos caracteres.

*** *** Los valores 8, 9, 10 y 13 se convierten a retroceso, tabulador, avance de línea y retorno de carro, respectivamente. No tienen ninguna representación gráfica, pero dependiendo de la aplicación, pueden influir en la presentación visual del texto.

Estándar Unicode

La especificación Unicode define un esquema de codificación único para prácticamente todos los caracteres usados con más frecuencia en todo el mundo. Todos los equipos convierten de forma coherente los patrones de bits de los datos Unicode en caracteres con la especificación única de Unicode. Esto asegura que el mismo patrón de bits se convierte siempre al mismo carácter en todos los equipos. Los datos se pueden transferir libremente desde un equipo a otro, sin preocuparse de que el sistema que los reciba pueda convertir los patrones de bits de forma correcta.

Un problema con los tipos de datos que usan un byte (código ASCII) para codificar cada carácter es que el tipo de datos sólo puede representar 256 caracteres distintos. Esto supone que tenga que haber varias especificaciones de codificación (o páginas de códigos) para distintos alfabetos, como, por ejemplo, los europeos, que son relativamente pequeños. Tampoco se puede tratar sistemas tales como el alfabeto Kanji japonés o el Hangul coreano, que tienen miles de caracteres.

La especificación Unicode resuelve este problema al utilizar 2 bytes para codificar cada carácter. Hay suficientes patrones distintos (65.536) en 2 bytes para establecer una única especificación que abarque la mayor parte de los idiomas comerciales comunes. Dado que todos los sistemas Unicode usan de forma coherente los mismos patrones de bits para representar todos los caracteres, se resuelve el problema que plantea el hecho de que los caracteres se pudieran convertir de forma incorrecta al pasarlos de un sistema a otro.

Representación de Números Enteros

Un entero cualquiera podría representarse dando una cadena de bytes, cada uno con el código del carácter correspondiente a una cifra decimal. De hecho esto implica un gran desperdicio de memoria, la longitud de las cadenas es variable según la cantidad de dígitos, y todo esto dificulta la realización de las operaciones.

Resulta entonces más adecuado fijar a priori una cantidad determinada de bytes o *palabra* (lo más corriente son 4 bytes) para contener cualquier número entero, y almacenar allí su representación binaria.

Debemos además, adoptar como criterio si los números a representar son sólo positivos (sin signo) o si pueden ser positivos y negativos (con signo).

Sin Signo: en este caso todos los bits de la palabra representan la cifra *(no existe un bit destinado a indicar el signo ya que se asume que el número es positivo).*

Cantidad de Bits de la Palabra	Valor Mínimo	Valor Máximo	Cantidad de cifras representadas
8	0	255	256
16	0	65.535	65.536
32	0	4.294.967.295	4.294.967.296

n $\qquad$ 2^n-1 $\qquad$ 2^n

Entonces, el menor valor representado siempre será cero (independientemente del tamaño de la palabra). La cantidad de cifras (valores) representados estará dado por b^n, donde b es la base y n la cantidad de bits de la palabra utilizada.
El valor máximo almacenado estará dado por b^n-1.

Con Signo: para poder distinguir entre un número positivo y otro negativo, habrá que reservar un bit (el primero de la izquierda) para determinar el signo.

Ej.: qué número entero está representado en la siguiente palabra de 32 bits?

0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1	1 1 0 0 0 0 0 1

Bit de signo

Si el contenido de este bit es 0,se considera que el número es positivo y sus cifras binarias corresponden a los restantes bits.

Cantidad de Bits de la Palabra	Valor Mínimo	Valor Máximo	Cantidad de cifras representadas
8	-128	127	256
16	-32.768	32.767	65.536
32	-2.147.483.648	2.147.483.647	4.294.967.296

n $\qquad$ $2^{(n-1)}$ $\qquad$ $2^{(n-1)}-1$ $\qquad$ 2^n

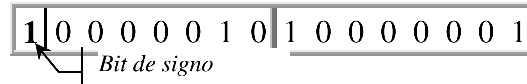
Para encontrar el número representado habrá que resolver el siguiente polinomio:

$$N = a_n.b^n + a_{n-1}.b^{n-1} + ... + a_i.b^i + ... + a_0.b^0$$

Según el ejemplo, la respuesta es 65985.

Cuando el bit de signo es 1, se conviene que el número es negativo, y en este caso, para facilitar la operatoria, se lo almacena con la notación de *complemento a 2*. Esta notación resulta de colocar un 1 donde la representación binaria del valor absoluto hay un 0 y viceversa, sumando luego 1 al resultado.

<u>Ejemplo 1:</u> qué número entero está representado en la siguiente palabra de 16 bits?



Para encontrar el número representado habrá que resolver el siguiente polinomio:

$$N = -a_n.b^n + a_{n-1}.b^{n-1} + ... + a_i.b^i + ... + a_0.b^0$$

Apreciar en signo negativo del primer término

Según el ejemplo, la respuesta es –32127.

<u>Ejemplo 2:</u> Almacenar -100_{10} en una palabra de 16 bits.

Es ahora, en donde utilizamos la representación en complemento a 2.

Entonces, hacemos:

1.- Representamos 100 en binario (con 15 dígitos, uno menos que el total porque reservamos el primero para el signo):

0 0 0 0 0 0 0 0 1 1 0 0 1 0 0

2.- Invertimos:

1 1 1 1 1 1 1 1 0 0 1 1 0 1 1

3.- Sumamos 1 al resultado (en binario):

1 1 1 1 1 1 1 1 0 0 1 1 1 0 0

4.- Agregamos el bit de signo (un 1):

Representación de Números Reales

Las operaciones con números reales (números fraccionarios) introducen un nuevo concepto, que es el del punto que separa la parte entera de la parte fraccionaria.

Representación de los números reales en punto fijo

En esta forma de representación se asigna, tal como su nombre lo indica, una posición fija al punto decimal. Por ejemplo, si se opera con números binarios de ocho bits, el punto puede situarse de forma arbitraria en cualquiera de las posiciones, pero una vez elegida no se modifica.

La principal ventaja de este método de representación es que los algoritmos de realización de las diferentes operaciones son los mismos que para los números enteros.

Por ejemplo, sean los siguientes números de ocho bits con dos decimales: **011011.11** y **100001.01**

La suma de ambos números es:

$$\begin{array}{r} 0\,1\,1\,0\,1\,1.\,1\,1 \\ +\,1\,0\,0\,0\,0\,1.\,0\,1 \\ \hline 1\,1\,1\,1\,0\,1.\,0\,0 \end{array}$$

y se observa que la operación se realiza como si los números fuesen enteros.

El principal inconveniente de la representación en punto fijo es que no se aprovecha la capacidad de los operadores aritméticos. En efecto, si se suponen, al igual que antes, números de ocho bits con dos fraccionarios, el máximo número que puede representarse es **111111.11** y el mínimo distinto de cero es **0.01**. Pero la capacidad de los operadores podría permitir operar el número máximo **11111111** y el mínimo **0.00000001**.

Todo lo desarrollado hasta aquí para los números enteros, corresponde a la notación de punto fijo.

Representación de los números reales en punto flotante

Este modo de representación evita el inconveniente mencionado en la representación en punto fijo. Dado el enorme rango de magnitud en que pueden variar los números reales, y la imposibilidad de almacenar todas sus cifras, los sistemas de computación adoptan formas de representación basadas en la notación exponencial, y que constituyen la notación de punto flotante.

Esto se basa en que un número real cualquiera ***n*** puede descomponerse en el producto de dos factores:

$n = m\, b^{e}$, donde:

1. ***m*** es la mantisa, menor que la unidad y con una cantidad dada de cifras significativas (es decir, la primera de ellas es distinta de cero).
2. b^{e} en un factor exponencial igual a la base de numeración ***b***, elevada a un exponente o característica ***e***; esta característica indica cuantos lugares debe correrse el punto decimal.

Por ejemplo, en el sistema de numeración decimal, un número ejemplo de formato de punto flotante es 2.25×10^{4}. Pero este número puede representarse de muy diversas maneras:

$$\mathbf{2.25 \times 10^{4} = 0.0225 \times 10^{6} = 225000 \times 10^{-1} = \ldots\ldots}$$

que se diferencian por la posición del punto en la mantisa y el valor del exponente.

De todas las posibles formas de representar un número en punto flotante la que se utiliza en la práctica y se define como normalizada es aquella en que el punto decimal se coloca antes de la cifra más significativa distinta de cero. En el ejemplo dado anteriormente, la representación normalizada es: $\mathbf{0.225 \times 10^5}$ donde *mantisa=0.225* y *exponente=5*

Entonces, en una palabra de la memoria se debe almacenar juntos la característica y la mantisa.

Palabra de 4 bytes (puede tener una longitud de 2, 4 o más bytes)

En los sistemas de computación es corriente que la base sea 16.

Debido a que no existe uniformidad entre los distintos equipos y lenguajes en cuanto al lugar y extensión que ocupan característica y mantisa dentro de una palabra, vamos a comentar una de las variantes unas frecuentes:

a) Los dígitos binarios de la mantisa se encuentran a partir del segundo byte sin emplear el complemento a 2.

b) El primer bit del primer byte es 0 si el número es positivo y 1 si es negativo.

c) Los siete bits restantes del primer byte contienen un número binario. Restando 64 a dicho número se obtiene la característica. En siete bits se pueden almacenar enteros entre 0 y 127. Al restar 64, el rango de variación de la característica está entre –64 y +63. Como la base es 16, resulta que:

$\mathbf{16^{-64} \cong 10^{-78}}$ y $\mathbf{16^{63} \cong 10^{75}}$

Estos son los ordenes en que puede variar la magnitud de los reales almacenados.

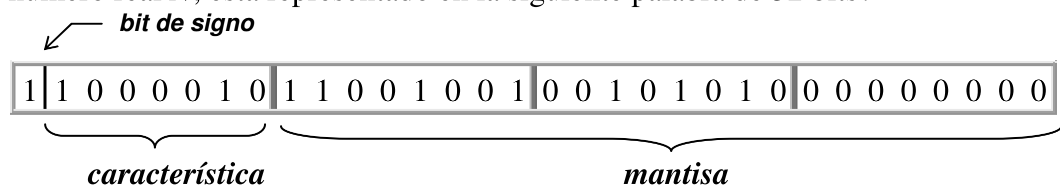
Al mismo tiempo. La existencia de 24 cifras binarias significativas en la mantisa equivale a 6 hexadecimales, lo que permite almacenar números positivos del orden de $\mathbf{16^6 \cong 10^7}$, o sea 7 cifras decimales significativas.

Para aumentar la cantidad de cifras significativas (precisión) se puede recurrir a la representación en doble precisión, o sea un real se almacena en dos palabras consecutivas de 4 bytes cada una (64 bits).

El exponente sigue ocupando en primer byte y su rango no cambia, pero la mantisa tiene ahora 56 bits, o sea 14 dígitos hexadecimales. Como $16^{14} \cong 10^{16}$, hay 16 dígitos decimales significativos.

Ejemplos de resumen

a) Qué número real ***N***, está representado en la siguiente palabra de 32 bits?



Signo: bit=1 → negativo

Característica: $66_{10} - 64_{10} = 2_{10}$ ($64_{10}=1000010_2$) → *correr dos lugares a la derecha el punto decimal.*

Mantisa: $-0.C92600_{16}$

Luego: $\boldsymbol{N = -C9.26_{16} = -201.1484375_{10}}$

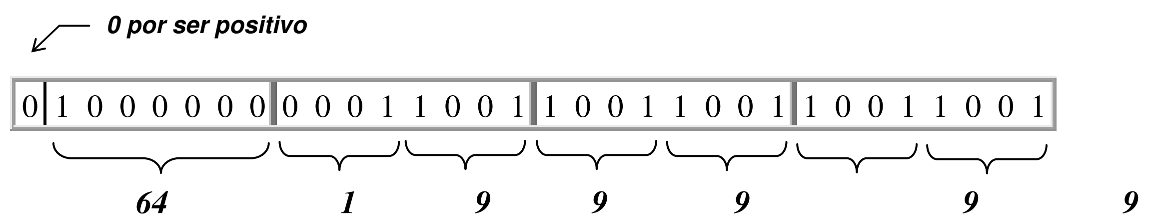
b) Representar 0.1_{10} en una palabra de 32 bits.

Habíamos visto que: $0.1_{10} = 0.1999999\ldots_{16}$, luego

Signo: +

Característica: 0 → almacenar 64 en exponente (1000000_2)

Mantisa: $0.1999999\ldots_{16}$



Aquí se aprecia claramente el error de truncamiento.

c) Representar el número ¶ (3.14159_{10}) en una palabra de 32 bits.

1.- Convierto el número a base 16

Parte entera	**Parte decimal**
$3_{10}= 3_{16}$	$0.14159_{10} = 0.243F3_{16}$
$\mathbf{3.14159_{10} = 3.243F3E0_{16}}$	

2.- Normalizo

$\mathbf{3.243F3E0_{16} = .3243F3_{16} \times 16_{10}^{1}}$

3.- exponente: $1_{10} + 64_{10} = 65_{10} = 1000001_2$

4.- mantisa: $243F3_{16}$ = 0010 0100 0011 1111 0011_2

2 4 3 F 3

bit de signo: 0 (por ser positivo)

$\mathbf{3.14159_{10}}$ = | 0 | 1000001 | 0010 0100 0011 11110011 $_2$ |

Observación:

El autor Enrique Mandado hace referencia a que la mantisa se representa en el convenio de complemento a 2.

Tabla de Código ASCII Multilingual

0		32		64	@	96	`	128	Ç	160	á	192	└	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
9	○	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	☼	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	►	48	0	80	P	112	p	144	É	176	░	208	ð	240	-
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	‼	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	▬	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	∟	60	<	92	\	124	¦	156	£	188	╝	220	▄	252	³
29	↔	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

Apéndice B

A título ilustrativo, se muestra una tabla con tipos de datos, tamaño que ocupan al ser almacenados y el intervalo de representación. Los mismos corresponden al lenguaje de programación Visual Basic 5.0.

Tipo de datos	Tamaño de almacenamiento	Intervalo
Byte	1 byte	0 a 255
Boolean	2 bytes	True o False
Integer	2 bytes	-32.768 a 32.767
Long (entero largo)	4 bytes	-2.147.483.648 a 2.147.483.647
Single (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Double (coma flotante/ precisión doble)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
Currency (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
Decimal	14 bytes	+/-79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/-0,0000000000000000000000000001

Nota Las matrices de cualquier tipo de datos requieren 20 bytes de memoria más cuatro bytes para cada dimensión de matriz, más el número de bytes que ocupan los propios datos. Puede calcular la memoria que ocupan los datos multiplicando el número de elementos de datos por el tamaño de cada elemento. Por ejemplo, los datos de una matriz unidimensional que consten de cuatro elementos de datos tipo Integer de dos bytes cada uno, ocupan ocho bytes. Los ocho bytes que requieren los datos más los 24 bytes necesarios para la matriz suman un requisito total de memoria de 32 bytes para dicha matriz.
Un tipo Variant que contiene una matiz requiere 12 bytes más que la matriz por sí sola.

Bibliografía

Apunte Sistemas de Numeración – Representación Interna, Ing. Fernando Guspí. 1987.
Sistemas Electrónicos Digitales, Enrique Mandado
La PC por dentro. Arquitectura y funcionamiento de computadores. M. C. Ginzburg
DIGITAL Fotran. Languaje Reference Manual. DIGITAL
Micorsoft Visual Basic 5.0
Microsoft SQL Server 7.0