

Sistema de numeración binario

Introducción

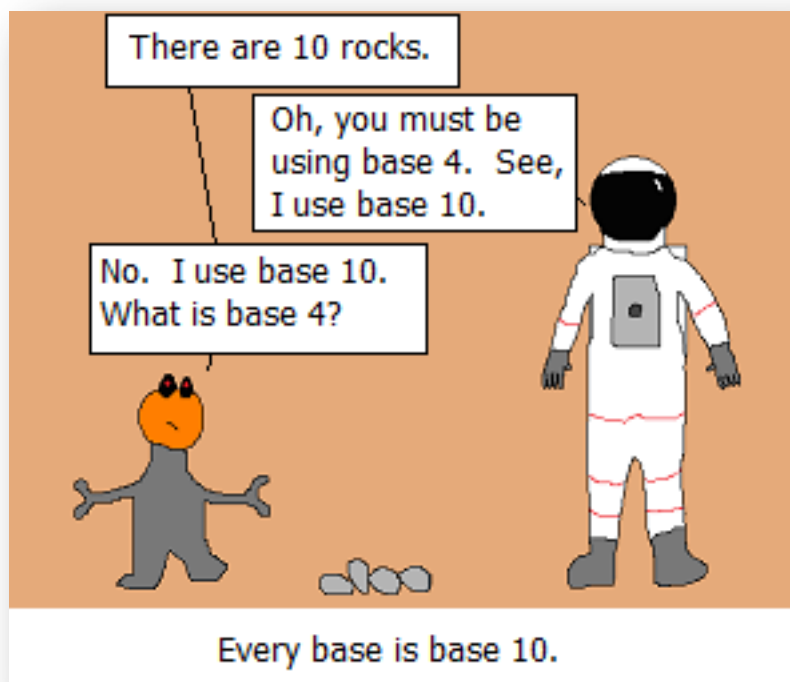
El sistema de numeración que usamos de manera más natural es el sistema decimal. Con este sistema usamos 10 símbolos, del 0 al 9, y los colocamos posicionalmente en función de unos pesos en base 10.

La posición menos significativa, la que menos vale, está a la derecha y se denomina **unidades**. Su peso es 10^0 , es decir, 1. La siguiente posición se denomina **decenas** y su peso es 10^1 , es decir, 10. La tercera posición menos significativa se denomina **centenas** y su peso es 10^2 , es decir 100.

Otros sistemas

Posiblemente por tener 10 dedos, nuestro sistema natural de contar cosas es base 10 ya que tenemos un carácter (un dedo) para cada uno de esos 10 elementos de nuestro alfabeto decimal.

Si los pulpos contaran probablemente lo harían en base 8 ya que tienen 8 patas y para ellos sería la forma natural.



Que el decimal sea nuestro sistema de numeración natural no quiere decir que solo podamos usar este. En realidad estamos acostumbrados a usar sistemas con otras bases. Por ejemplo, los minutos y segundos son base 60, no existe ninguna hora con 61 segundos. Las horas son base 24, ninguna hora comienza por 24. Y si seguimos así podríamos pensar en qué medimos con base 7 o con base 12.

Sistema de numeración binario.

El sistema de numeración binario solamente tiene dos símbolos en su alfabeto, el 0 y el 1. Los pesos de sus posiciones tienen como base el número 2. De esta manera, la posición más a la derecha, la menos significativa, tiene un peso de 1, es decir 2^0 . La siguiente posición tiene un peso de 2, la siguiente de 4, la siguiente de 8 y la siguiente de 16. Así hasta el infinito.

El sistema binario es sencillo ya que al solo tener dos símbolos en su alfabeto podemos leer un número viendo si se aplica el peso de cada una de sus cifras o no.

Por ejemplo, el número 101 tiene 3 cifras binarias. La posición más a la derecha tiene un 1 e indica que el peso de esa posición se aplica. La segunda posición desde la derecha tiene un 0 e indica que el peso de esa posición, un 2, no se aplica. Por último, la tercera posición desde la derecha tiene un 1 e indica que el peso de esa posición, un 4 sí se aplica. Para obtener el valor en decimal de ese número binario tendremos que sumar los pesos efectivos. En este caso es un 1 y un 4 por lo que el número binario 101 equivale al número decimal 5.

En muchas ocasiones veremos que junto al número se puede indicar la base en forma de subíndice. Por lo tanto es mucho más correcto decir que el número 101_2 equivale al número 5_{10} .

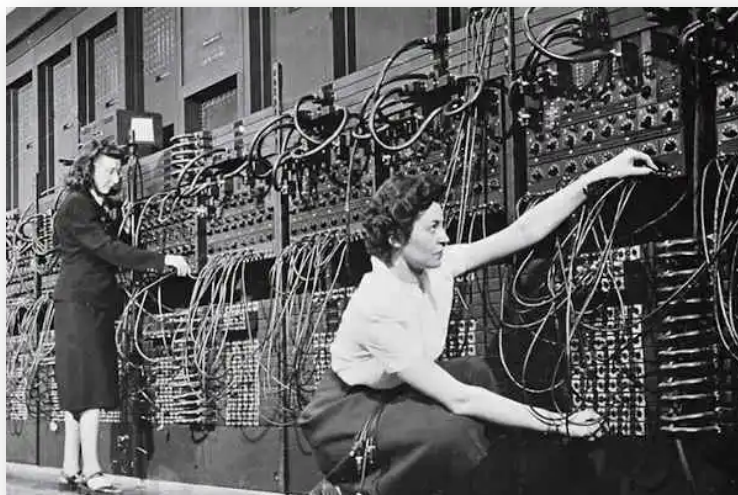
En informática el binario es fundamental y representa la base de prácticamente todo. Los tipos de datos de los lenguajes de programación están íntimamente asociados a la cantidad de cifras binarias que puede almacenar ese tipo concreto.

TIPO	ANCHO EN BIT	RANGO EN PC
<i>char</i>	8	-128 a 127
<i>unsigned char</i>	8	0 a 255
<i>signed char</i>	8	-128 a 127
<i>int</i>	16	-32768 a 32767
<i>unsigned int</i>	16	0 a 65535
<i>signed int</i>	16	-32768 a 32767
<i>short int</i>	16	-32768 a 32767
<i>unsigned short int</i>	16	0 a 65535
<i>signed short int</i>	16	-32768 a 32767
<i>long int</i>	32	-2147483648 a 2147483647
<i>signed long int</i>	32	-2147483648 a 2147483647
<i>unsigned long int</i>	32	0 a 4294967295
<i>float</i>	32	3.4E-38 a 3.4E+38
<i>double</i>	64	1.7E-308 a 1.7E+308
<i>long double</i>	64	1.7E-308 a 1.7E+308

COMBINACIONES DE TIPOS DE DATOS

Códigos y sistema binario

En las primeras etapas de la informática los equipos eran grandes calculadoras cableadas para hacer unas determinadas operaciones. Los técnicos se encargaban de cargar los operandos mediante interruptores de válvulas de vacío. La ejecución del programa cableado permitía hacer cálculos matemáticos complejos de manera automática como por ejemplo trayectorias balísticas.



De una manera u otra, el concepto de código ya lo tenemos interiorizado. El código Morse es un sistema que asocia puntos y rayas a caracteres. La relación entre una determinada combinación de puntos y rayas a una determinada letra es algo definido en el código por lo que todos los que intervengan en la comunicación deben conocer y aplicar este código.

A nivel informático decimos que un código **mapea** números binarios en otro conjunto de elementos que pueden ser números, letras, comandos o cualquier concepto que queramos tratar. Usamos el verbo **mapear** para indicar que hay una asociación entre binario y otro concepto.

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

En la captura anterior podemos ver el código ASCII de 7 bits, el primero y más antiguo código que nos representa un mapeo de los caracteres alfanuméricos usados en las computadoras que tenían algún periférico de salida como un monitor o una impresora. Además de caracteres alfanuméricos también se incluían signos de puntuación, espacio en blanco y algunos comandos especiales como escape, beep, etc.

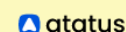
Este código requería de mapear 128 elementos y por lo tanto requería de 7 bits. Recordemos que con 7 bits podemos hacer 128 combinaciones desde el 0 (000 0000) hasta el 127 (111 1111).

A medida que los periféricos de salida mejoraban en tecnología se hizo necesario incluir en el código ASCII otros nuevos caracteres para poder dibujar ventanas o escribir caracteres propios de otros idiomas como la ñe o las

vocales acentuadas. Con 7 bits no era posible mapear tantos elementos por lo que hubo que modificar el código ASCII y convertirlo a 8 bits.

De este código de 8 bits nace la unidad mínima de información de informática, el **byte**.

Decimal System (SI)			Binary System		
Name	Symbol	Decimal Unit	Name	Symbol	Decimal Unit
Kilobyte	KB	10^3	Kibibyte	KiB	2^{10}
Megabyte	MB	10^6	Mebibyte	MiB	2^{20}
Gigabyte	GB	10^9	Gigibyte	GiB	2^{30}
Terabyte	TB	10^{12}	Tebibyte	TiB	2^{40}
Petabyte	PB	10^{15}	Pebibyte	PiB	2^{50}
Exabyte	EB	10^{18}	Exbibyte	EiB	2^{60}
Zettabyte	ZB	10^{24}	Zebibyte	ZiB	2^{70}
Yottabyte	YB	10^{21}	Yobibyte	YiB	2^{80}



A medida que el espacio de almacenamiento y el tamaño de los archivos crece usamos múltiplos de byte para informar de manera un poco más legible.

Estrictamente hablando usaremos los prefijos kilo, mega, giga, ... para hacer referencia al sistema decimal en el que el factor de conversión de unas unidades a otras es 1.000. El término informático preciso no usa el sistema decimal sino el binario y por ello el factor de conversión es 1.024. Para no confundirlos tienen nombres similares pero distintos. Así pues, en informática usamos los kibis, mebis, gibis,.. haciendo referencia a “kilo binario”, “mega binario”, etc.

La realidad es que lo habitual es usar los prefijos decimales (kilo, mega, giga,...) y el factor de conversión binario (1.024).

Pasar de unas unidades a otras consiste simplemente en aplicar el factor de conversión que toque o bien multiplicando cuando pasamos a una unidad menor, o bien dividiendo cuando pasamos a una unidad mayor.

Operaciones en binario

En binario tenemos el mismo juego de operaciones aritméticas que en decimal por lo que el concepto de acarreo o llevada es el mismo.

En la operación de suma solo tenemos un caso especial que es $1 + 1$. En este caso el resultado es 10_2 , o dicho de otra manera, un 0 y un 1 de acarreo.

En la operación de resta tenemos otro caso especial que es $0 - 1$. En este caso el resultado es 11_2 , o dicho de otra manera, un 1 y un 1 de acarreo.

La multiplicación es exactamente igual a la decimal. Una multiplicación por 0 se resuelve como 0 y una multiplicación por 1 es neutra quedando el mismo resultado.

La división también es igual que en el sistema decimal. Consiste en bajar una cifra y buscar el número que, multiplicado por el divisor, de un número menor o igual al que estamos tratando. Ese número se resta del que estamos tratando y se baja otra cifra. El proceso se repite hasta que el resto es menor que el divisor.

Números flotantes en binario. IEEE-754

Hasta ahora hemos visto los números enteros en binario. En el sistema de numeración decimal los pesos enteros son 1, 10, 100, 1000,... y los pesos de números flotantes son 0.1, 0.01, 0.001. Para la separación para saber qué parte es entera y qué parte es flotante usamos la coma. En el sistema binario los pesos para la parte decimal empiezan por 0,5 seguida de 0,25, seguida de 0,125.

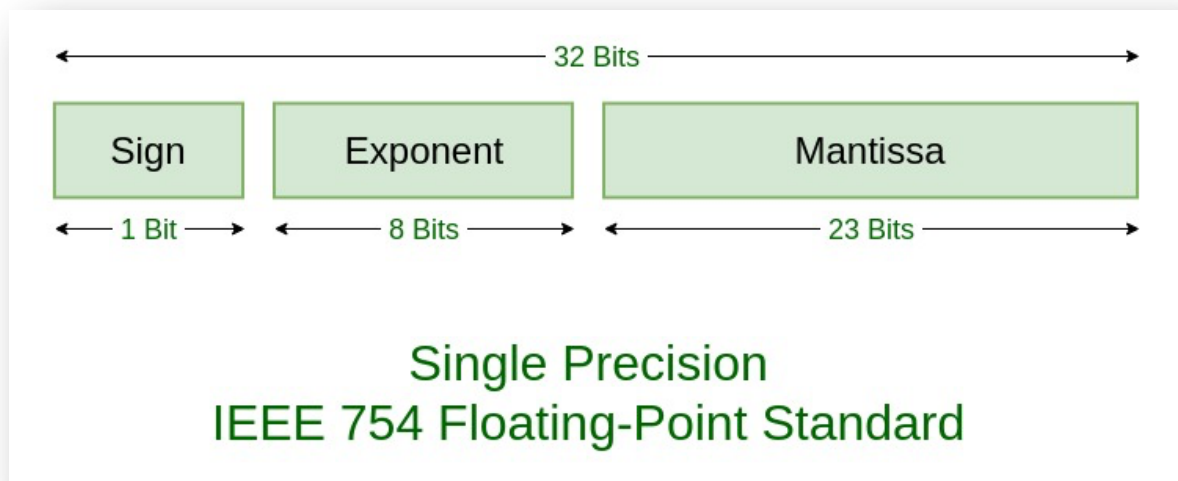
La idea es la misma que la aplicada a la parte entera. Siempre es el doble o la mitad en función del sentido en que se recorra.

Con los decimales nos encontraremos con problemas que no existían con los números enteros y es que habrá veces en las que no podremos escribir el número exacto y tendremos que aproximar. Por ejemplo, el número decimal 0,5 o el número decimal 0,75 se pueden pasar a binario exactamente como 0,1 y 0,11 en binario respectivamente. Pero el número decimal 0,6 es mucho más difícil de escribir porque necesitaremos muchas cifras decimales para aproximarlos.

Otro de los problemas que nos encontramos con los números decimales en binario es que debemos decidir cuantos bits ocupa ese tipo de datos y cuantos de esos bits reservamos para la parte entera y cuantos para la decimal.

La solución a estos problemas viene el estándar IEEE-754 que nos especifica como deben ser los números decimales en binario.

Como se ve en la siguiente captura, el tipo de datos float de 4 bytes (32 bits) divide sus bits de la siguiente forma:



En el primer paso, el bit más significativo (el de mayor peso o más a la izquierda) indica el signo del número decimal. Pondremos un cero si el número es positivo o un 1 si el número es negativo.

En el segundo paso convertiremos el número decimal a binario usando los pesos de la parte entera y de la parte decimal que ya conocemos

El tercer paso consiste en normalizar ese número binario del paso 2. Normalizar en este contexto significa mover la coma tantos lugares como sea necesario para quedarnos con un número binario que tenga un único 1 a la izquierda de la coma. Por ejemplo, normalizar el número binario $10010,001_2$ resultaría en $1,0010001_2$ habiendo movido la coma 4 posiciones a la izquierda.

En el cuarto paso calculamos el exponente de 8 bits sumando en binario el 127_{10} al número de saltos de coma que calculamos en el paso anterior. En el ejemplo anterior hemos movido la coma 4 posiciones a la izquierda por lo que sumamos $127 + 4 = 131$ que, en binario es $1\ 0\ 0\ 0\ 0\ 1\ 1$. Estos son los 8 bits que van en la parte reservada al exponente.

El quinto paso consiste en poner la parte decimal del número binario normalizado que calculamos en el paso tres y rellenar con ceros a la derecha hasta llegar al tope de los 23 bits reservados para la mantisa.

En internet podremos encontrar calculadoras que nos convertirán números decimales en su equivalente binario con el estándar IEEE-754.

Pasar de IEEE-754 a número decimal

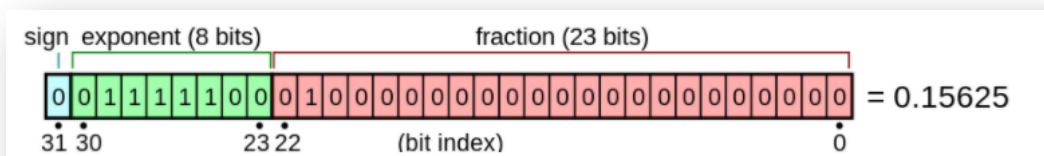
El algoritmo para realizar el paso de un número codificado en IEEE-754 a número decimal consiste en realizar los pasos que hemos visto anteriormente pero de manera inversa.

Recibiremos una secuencia de 32 bits y debemos separar en signo, exponente y mantisa contando 1, 8 y 23 bits respectivamente desde la izquierda, empezando por el más significativo.

En primer paso consistiría en, con los 23 bits de la derecha o los 23 bits menos significativos tenemos la mantisa. Esta es la parte que contiene la parte significativa del número. Es necesario recordar que los números en IEEE-754 se representan de una manera similar a la científica así que la mantisa siempre será un número binario que empieza por 1,xxx a la que, en el momento de codificarlo, le quitamos ese uno entero para ahorrar un bit. Por lo tanto ahora es necesario añadirlo. Si la mantisa fuera todo ceros, después de este paso deberíamos tener un 1 (coma) y todo cero.

El segundo paso consiste en averiguar el exponente que nos indicará cuantas posiciones hay que mover la coma para obtener el número original. El proceso es inverso al que vimos anteriormente. Ahora es necesario restarle 127 a lo que indique el exponente del número IEEE-754. Por ejemplo, si en el exponente tuviéramos el número 131, la operación a realizar sería $131 - 127 = 4$. Esto nos estaría indicando que hay que mover la coma 4 saltos hacia la derecha. Por ejemplo, si después del primer paso tuviéramos un 1 (coma) y todo cero, tendríamos que mover la coma 4 lugares a la derecha y quedaría el número 10000 (coma) y todo cero.

Por último, el tercer paso consistiría en asignar el valor positivo o negativo en función del valor del bit más significativo. Este bit a cero indica que es un número positivo y ese bit a 1 indica que es un número negativo.



The real value assumed by a given 32-bit *binary32* data with a given *sign*, biased exponent *e* (the 8

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29} \dots b_{23})_2 - 127} \times (1.b_{22}b_{21} \dots b_0)_2,$$

which yields

$$\text{value} = (-1)^{\text{sign}} \times 2^{(e-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} \right).$$

In this example:

- $\text{sign} = b_{31} = 0,$

Para aquellos que tengan un perfil muy matemático podrían usar la fórmula de la captura anterior.

Tablas de ayuda

Las siguientes tablas pueden servir de ayuda rápida para solucionar problemas con un enfoque más práctico que teórico.

La siguiente tabla representa los pesos de las primeras cifras binarias de un número binario. La parte a la izquierda de la coma representa la parte entera y la parte a la derecha de la coma representa la parte decimal.

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
								,					

Esta tabla se puede usar para pasar rápidamente un número decimal binario a un número decimal. Por ejemplo, el número 10,11 binario representa el número 2,75 en decimal.

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
						1	0	,	1	1			2,75

También se puede utilizar esta tabla para pasar de decimal a binario. En este caso hay que realizar un proceso iterativo en el que vamos restando pesos hasta llegar a cero. Por ejemplo, si tenemos que pasar el número decimal 19 a binario tendríamos que realizar el siguiente algoritmo.

Necesito un bit con peso 128 para representar un 19? No, me paso. Es un cero en esa posición.

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
0								,					19

Y así con los pesos 64 y 32 hasta llegar a un nuevo caso que sería, necesito un bit con peso 16 para representar a un 19? Sí. Lo uso con un 1 y a partir de ahora busco un nuevo valor que se calcula como lo que buscaba menos el peso que acabo de asignar ($19-16=3$)

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
0	0	0	1					,					19

Necesito un bit con peso 8 para representar a un número 3? No, me paso. Es un cero en esa posición

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
0	0	0	1	0				,					19

Necesito un bit con peso 4 para representar a un número 3? No, me paso. Es un cero en esa posición

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
								,					

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
0	0	0	1	0	0	1		,					19

128	64	32	16	8	4	2	1	,	0,5	0,25	0,125	0,0625	Decimal
0	0	0	1	0	0	1	1	,					19

$$28 = 11100_2$$

Este sistema solo nos vale para los números enteros. En caso de querer aplicarlo a la parte decimal el sistema varía un poco. Consiste en tomar la parte decimal, multiplicarla por 2 y tomar la parte entera de la operación en ese orden. Si queremos seguir aproximando tomamos la parte decimal y la volveríamos a multiplicar por 2.

$$8,7 \longrightarrow 8 \mid \begin{array}{r} 2 \\ 0 \ 4 \mid 2 \\ 0 \ 2 \mid 2 \\ 0 \ 1 \end{array}$$

$$1000, ???$$

$$\begin{array}{l} 2 \times 0,7 = 1,4 \longrightarrow 1 \\ 2 \times 0,4 = 0,8 \longrightarrow 0 \\ 2 \times 0,8 = 1,6 \longrightarrow 1 \\ 2 \times 0,6 = 1,2 \longrightarrow 1 \end{array}$$

$$1000,1011$$
