

Bases numéricas

Representación de Datos en la Computadora

Básicamente se necesita representar en la computadora dos tipos de datos: texto y números.

Pero estos datos han de representarse internamente por medio de secuencias de impulsos eléctricos ya que, en definitiva, eso es lo que puede manipular la computadora. Por tanto, los símbolos tipográficos y las cantidades numéricas han de tener una representación en base a impulsos eléctricos. Un impulso eléctrico lo representamos por medio de un 1 (representación lógica): hay presencia de impulso. Y a través de 0, hay ausencia de impulso. Estos son los únicos dos símbolos utilizados, por ello se dice que es un sistema de representación binaria.

Fuera del ámbito de las computadoras actuales también ha habido muchos intentos para representar símbolos y lograr la comunicación entre las personas. Por esta última razón se complementa mucho con el criterio humano y a computadora se necesita un tipo de representación que no dé lugar a interpretaciones ambiguas o erróneas, sino únicas y acertadas. La solución es utilizar una secuencia de dígitos binarios de igual tamaño para cada símbolo y que el espacio entre un símbolo y otro sea otro símbolo de igual tamaño.

Existieron muchos intentos a lo largo de la historia de la computación para fabricar tablas de equivalencias de símbolos tipográficos y secuencias de dígitos binarios, el más difundido es el código ASCII (American Standard Code for Information Interchange), que utilizó 7 dígitos en su primera versión y utiliza 8 actualmente.

La cantidad de dígitos de esa secuencia dependerá de la cantidad de símbolos que se quiera representar en casillas, en general, con N casillas y dos dígitos para realizar combinaciones, estas se pueden representar con 2^N combinaciones posibles.

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledg.	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
00011101	29	Group separator	00111101	61	=	01011101	93]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
10000000	128	Ç	10100000	160	á	11000000	192	+	11100000	224	Ó
10000001	129	ü	10100001	161	í	11000001	193	-	11100001	225	ß
10000010	130	é	10100010	162	ó	11000010	194	-	11100010	226	Ô
10000011	131	â	10100011	163	ú	11000011	195	+	11100011	227	Ò
10000100	132	ä	10100100	164	ñ	11000100	196	-	11100100	228	ö
10000101	133	à	10100101	165	Ñ	11000101	197	+	11100101	229	Õ
10000110	134	â	10100110	166	ª	11000110	198	ä	11100110	230	µ
10000111	135	ç	10100111	167	º	11000111	199	Ã	11100111	231	þ
10001000	136	ê	10101000	168	¿	11001000	200	+	11101000	232	Ð
10001001	137	ë	10101001	169	®	11001001	201	+	11101001	233	Ú
10001010	138	è	10101010	170	¬	11001010	202	-	11101010	234	Û
10001011	139	ï	10101011	171	½	11001011	203	-	11101011	235	Ü
10001100	140	î	10101100	172	¼	11001100	204	¡	11101100	236	Ý
10001101	141	ì	10101101	173	í	11001101	205	-	11101101	237	Ÿ
10001110	142	Ä	10101110	174	«	11001110	206	+	11101110	238	-
10001111	143	Å	10101111	175	»	11001111	207	©	11101111	239	·
10010000	144	É	10110000	176	-	11010000	208	ð	11110000	240	-
10010001	145	æ	10110001	177	-	11010001	209	Ð	11110001	241	±
10010010	146	Æ	10110010	178	-	11010010	210	Ê	11110010	242	-
10010011	147	ô	10110011	179	-	11010011	211	Ê	11110011	243	¾
10010100	148	ö	10110100	180	-	11010100	212	È	11110100	244	¶
10010101	149	ò	10110101	181	Á	11010101	213	í	11110101	245	§
10010110	150	û	10110110	182	Â	11010110	214	Î	11110110	246	÷
10010111	151	ù	10110111	183	Ã	11010111	215	Ï	11110111	247	-
10011000	152	ÿ	10111000	184	©	11011000	216	Ï	11111000	248	ó
10011001	153	Ö	10111001	185	-	11011001	217	+	11111001	249	“
10011010	154	Ü	10111010	186	-	11011010	218	+	11111010	250	•
10011011	155	ß	10111011	187	+	11011011	219	-	11111011	251	¹
10011100	156	£	10111100	188	+	11011100	220	-	11111100	252	º
10011101	157	Ø	10111101	189	¢	11011101	221	-	11111101	253	²
10011110	158	×	10111110	190	¥	11011110	222	Î	11111110	254	-
10011111	159	f	10111111	191	+	11011111	223	-	11111111	255	-

Bit de paridad

Un bit de paridad es un poco, con un valor de 0 o 1, que se agrega a un bloque de datos para fines de detección de errores. Les da a los datos un par o impar paridad, que se utiliza para validar la integridad de los datos.

Los bits de paridad a menudo se usan en la transmisión de datos para garantizar que los datos no se corrompan durante el proceso de transferencia. El bit de paridad para cada paquete de datos se calcula antes de que se transmitan los datos.

Paridad Par: A cada grupo de ocho bits, de la hilera, se le agrega un bit adicional a la izquierda. Este será un uno si la cantidad de unos es impar, y cero si la cantidad de unos es par, de tal manera que la cantidad de unos del nuevo grupo de nueve bits sea par.

Hacer par: Letra “a” en binario

0	1	1	0	0	0	0	1	>	1	0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ya es par: Letra “A” en binario

0	1	0	0	0	0	0	1	>	0	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Paridad Impar: A cada grupo de ocho bits, de la hilera, se le agrega un bit adicional a la izquierda. Este será un cero si la cantidad de unos es impar, y uno si la cantidad de unos es par, de tal manera que la cantidad de unos del nuevo grupo de nueve bits sea impar.

Hacer impar: Letra “A” en binario

0	1	0	0	0	0	0	1	>	1	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ya es impar: Letra “a” en binario

0	1	1	0	0	0	0	1	>	0	0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Si bien la verificación de paridad es una forma útil de validar los datos, no es un método infalible. Por ejemplo, los valores 1010 y 1001 tienen la misma paridad. Por lo tanto, si se transmite el valor 1010 y se recibe 1001, no se detectará ningún error. Esto significa que las comprobaciones de paridad no son 100% confiables al validar datos. Aun así, es poco probable que más de un bit sea incorrecto en un pequeño paquete de datos. Mientras solo se cambie un bit, se producirá un error. Por lo tanto, las comprobaciones de paridad son más confiables cuando se utilizan paquetes pequeños.

Representación de Cantidades Numéricas

Sistemas de Numeración Posicional

Las personas utilizamos los números en muchas actividades de la vida diaria, Es imposible escapar de los números. Existen dos formas de representar los valores numéricos de las cantidades:

- Analógica.

Una cantidad se denota por otra que es directamente proporcional a ella.

Características de estas cantidades:

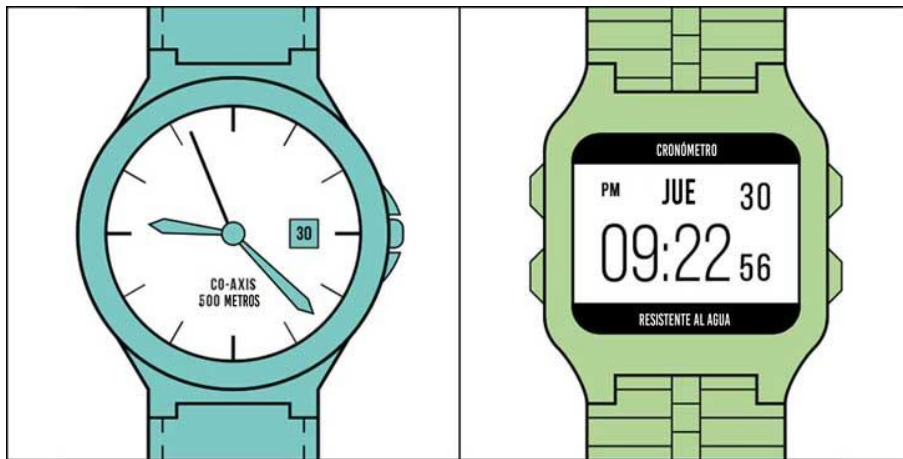
- Intervalo continuo de valores.
- El valor se presta a interpretación y es aproximado.

- Digital.

Las cantidades se representan por símbolos denominados dígitos.

Características:

- La representación de cantidades varía en etapas discretas.
- No hay ambigüedad al leer las cantidades.



Sistemas de Números Digitales

Los más comunes: Decimal, Octal, Binario y Hexadecimal.

Se componen de los siguientes dígitos: (base N números representables y será base N-1 el número mayor)

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Conversión de números de una unidad a otra

Cuando se trabaja con números en diferentes bases, para indicar que un número está en determinada base se escribe entre paréntesis y se le coloca, como sub índice, la base en la que está escrito, ejemplo: $(N)_B$

Conversión de Números en Base 10 a otras bases

- Conversión de la parte entera:

Se realiza por medio de divisiones enteras sucesivas entre la base a la que se desea convertir, comenzando con el número entero original y continuando con los cocientes resultantes, hasta que ya no se puedan seguir realizando divisiones.

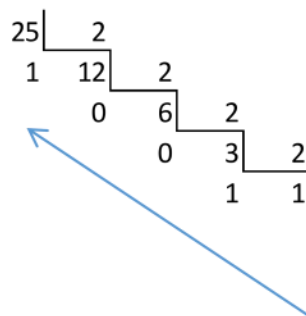
- Conversión de la parte fraccionaria:

Se realiza por medio de productos sucesivos por la base a la que se desea convertir, comenzando con la fracción original y continuando con cada fracción resultante, hasta obtener 0 como fracción o una aproximación bastante buena al valor equivalente.

Ejemplos:

Conversión de $(25)_{10}$ a binario:

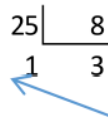
Realizando las divisiones sucesivas entre 2 y tomamos los dígitos en el orden inverso en que aparecen:



La respuesta es: **$(11001)_2$**

Conversión de $(25)_{10}$ a octal:

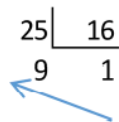
Realizando las divisiones sucesivas entre 8 y tomamos los dígitos en el orden inverso en que aparecen:

$$\begin{array}{r|l} 25 & 8 \\ \hline 1 & 3 \end{array}$$


La respuesta es: **$(31)_8$**

Conversión de $(25)_{10}$ a hexadecimal:

Realizando las divisiones sucesivas entre 16 y tomamos los dígitos en el orden inverso en que aparecen:

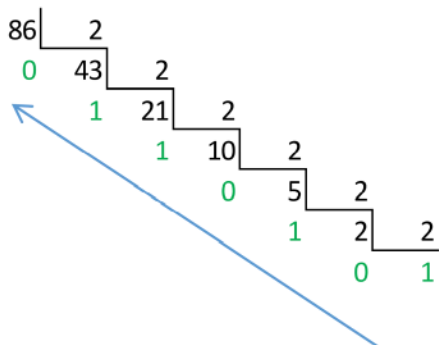
$$\begin{array}{r|l} 25 & 16 \\ \hline 9 & 1 \end{array}$$


La respuesta es: **$(19)_{16}$**

Conversión de $(86.57)_{10}$ a binario:

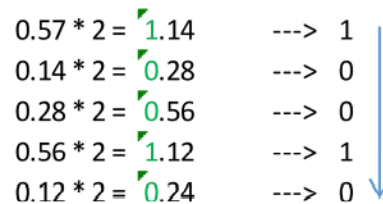
i. Conversión de la parte entera:

Resolvemos por medio de divisiones sucesivas entre 2 y tomamos los dígitos en el orden inverso en que aparecen:

$$\begin{array}{r|l} 86 & 2 \\ \hline 0 & 43 \\ 1 & 21 \\ 1 & 10 \\ 0 & 5 \\ 1 & 2 \\ 0 & 1 \end{array}$$


ii. Conversión de la parte fraccionaria:

Resolvemos por medio de productos sucesivos por 2 y tomamos los dígitos en el orden de aparición:

$$\begin{array}{lll} 0.57 * 2 = 1.14 & \text{--->} & 1 \\ 0.14 * 2 = 0.28 & \text{--->} & 0 \\ 0.28 * 2 = 0.56 & \text{--->} & 0 \\ 0.56 * 2 = 1.12 & \text{--->} & 1 \\ 0.12 * 2 = 0.24 & \text{--->} & 0 \end{array}$$


La respuesta es: **$(1010110.10010)_2$**

Conversión de $(86.57)_{10}$ a octal:

i. Conversión de la parte entera:

Resolvemos por medio de divisiones sucesivas entre 8 y tomamos los dígitos en el orden inverso en que aparecen:

ii. Conversión de la parte fraccionaria:

Resolvemos por medio de productos sucesivos por 8 y tomamos los dígitos en el orden de aparición:

$0.57 * 8 =$	4.56	--->	4
$0.56 * 8 =$	4.48	--->	4
$0.48 * 8 =$	3.84	--->	3
$0.84 * 8 =$	6.72	--->	6
$0.72 * 8 =$	5.76	--->	5

La respuesta es: **(126.44365)₈**

Conversión de $(86.57)_{10}$ a hexadecimal:

i. Conversión de la parte entera:

Resolvemos por medio de divisiones sucesivas entre 16 y tomamos los dígitos en el orden inverso en que aparecen:

$$\begin{array}{r|l} 86 & 16 \\ \hline 6 & 5 \end{array}$$

ii. Conversión de la parte fraccionaria:

Resolvemos por medio de productos sucesivos por 16 y tomamos los dígitos en el orden de aparición:

$0.57 * 16 = 9.12$	--->	9
$0.12 * 16 = 1.92$	--->	1
$0.92 * 16 = 14.72$	--->	E
$0.72 * 16 = 11.52$	--->	B
$0.52 * 16 = 8.32$	--->	8

La respuesta es: **(56.91EB8)₁₆**

Conversión de Números expresados en otras bases a Base 10

Esta conversión se hace realizando la suma de los productos de cada dígito por la base elevada al exponente según la posición que ocupa el dígito.

Debajo de cada dígito del número se coloca la posición que ocupa, así:

1	0	0	1	1	0	1	
6	5	4	3	2	1	0	<--- Estos son los exponentes de la base

Luego se realiza la suma de los productos de cada dígito multiplicado por una potencia de la base, así:

$$\begin{aligned} &1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 \\ &= 64 + 0 + 0 + 8 + 4 + 0 + 1 \\ &= \mathbf{(77)_{10}} \end{aligned}$$

Debajo de cada dígito del número se coloca la posición que ocupa, así:

1	1	0	1	1	1	1	0	.	1	0	1	0	1
7	6	5	4	3	2	1	0		-1	-2	-3	-4	-5

Luego se realiza la suma de los productos de cada dígito multiplicado por una potencia de la base, así:

$$\begin{aligned} &1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} + 0*2^{-4} + 1*2^{-5} \\ &= 128 + 64 + 0 + 16 + 8 + 4 + 2 + 0 + 0.5 + 0 + 0.125 + 0 + 0.03125 \\ &= \mathbf{(222.65625)_{10}} \end{aligned}$$

Debajo de cada dígito del número se coloca la posición que ocupa, así:

F	4	C	.	4	B	6
2	1	0		-1	-2	-3

Luego se realiza la suma de los productos de cada dígito multiplicado por una potencia de la base, así:

$$F \cdot 16^2 + 4 \cdot 16^1 + C \cdot 16^0 + 4 \cdot 16^{-1} + B \cdot 16^{-2} + 6 \cdot 16^{-3}$$

OJO con las letras: ya conocemos la equivalencia. De acuerdo a la tabla que vimos en un pdf anterior: F es 15, C es 12 y B es 11.

$$15 \cdot 16^2 + 4 \cdot 16^1 + 12 \cdot 16^0 + 4 \cdot 16^{-1} + 11 \cdot 16^{-2} + 6 \cdot 16^{-3}$$

$$= 3840 + 64 + 12 + 0.25 + 0.04297 + 0.001465$$

$$= (3916.294435)_{10}$$

La respuesta es: **(15.EA)₁₆**

- Si la base origen es mayor que la base destino

$$(N)_{Ba} \ggg (\cdot?)_{Bb}$$

Y se sabe: $Ba = (Bb)^N$

Ejemplos:

Como se va a convertir de base 8 a base 2 y se sabe que $8 = 2^3$, entonces se puede hacer la conversión de manera directa, sin realizar ninguna operación.

Además como la base origen es mayor que la base destino, cada dígito octal se va a reescribir como su equivalente binario de 3 bits.

3	6	1	•	4	6
0 1 1	1 1 0	0 0 1		1 0 0	1 1 0

OJO con los extremos: al hacer los grupos de 4 partiendo de punto decimal, observar que los extremos no quedan de tres dígitos, así que se completan rellenando con ceros de tal manera de no alterar el número.

Así que tenemos: $(011110001.100110)_2$

Y como los ceros de los extremos no aportan, los podemos quitar

La respuesta es: **$(11110001.10011)_2$**

Como se va a convertir de base 16 a base 2 y se sabe que $16 = 2^4$, entonces se puede hacer la conversión de manera directa, sin realizar ninguna operación.

Además como la base origen es mayor que la base destino, cada dígito hexadecimal se va a reescribir como su equivalente binario de 4 bits.

D	1	C	•	A	F	E
1 1 0 1	0 0 0 1	1 1 0 0		1 0 1 0	1 1 1 1	1 1 1 0

Así que tenemos: $(110100011100.101011111110)_2$

OJO con el extremo derecho: ese cero no aporta valor al número, así que se puede omitir al escribir la respuesta.

La respuesta es: **$(110100011100.10101111111)_2$**