Repaso de Fundamentos de Representación de la Información

Sistemas Basados en Microprocesadores

Víctor Manuel Sánchez Corbacho

Dpto. de Automática, Electrónica, Arquitectura y Redes de Computadores

2017

Contenido

Representación de la información en los sistemas digitales

- Los computadores, microprocesadores y microcontroladores son sistemas digitales binarios.
- Toda la información que procesan consiste en números binarios.

Sistemas de numeración posicional

- Un sistema de numeración en **base n** usa un conjunto de **n símbolos**.
- Un número se representa mediante una secuencia de símbolos contribuyendo cada uno con un valor que depende de
  - El valor asignado a cada símbolo.
  - La posición que cada símbolo ocupa en la secuencia.

Ejemplo

En base 10:

- Conjunto de símbolos: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $345 = 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$

Teorema fundamental de la numeración

- Relaciona una cantidad expresada en cualquier base de numeración posicional con la misma cantidad expresada en el sistema decimal.

$$D = \sum_{i=-d}^{n-1} (dígito)_i \cdot (base)^i$$

donde

D	valor expresado en decimal.
i	posición respecto a la coma.
d	número de dígitos a la derecha de la coma.
n	número de dígitos a la izquierda de la coma.
$(dígito)_i$	dígito en la posición i respecto de la coma.
$base$	base del sistema de numeración.

Sistema binario

- El sistema binario, o base 2, usa sólo dos símbolos $\{0, 1\}$.
- Cada dígito de un número binario se llama **bit** (**bi**nary digi**t**).
- Los pesos son potencias de 2:

$\cdots$	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	$\cdots$
$\cdots$	1	1	0	0	1	,	0	1	1	$\cdots$

- **Orden de un bit**: Exponente de la potencia de 2 correspondiente.
- **Bit más significativo (MSB)**: Bit de mayor orden.
  Ejemplo: **1**10010101
- **Bit menos significativo (LSB)**: Bit de menor orden.
  Ejemplo: 11001010**1**

Bytes, nibbles y palabras

- Agrupaciones de bits

Byte Número binario de **8** bits.

- También se llama **octeto**.

Nibble Número binario de **4** bits.

- Un byte está compuesto por dos nibbles.

Palabra Tamaño de los números binarios usados por un determinado sistema o aplicación.
Su tamaño varía según el contexto.

Rangos de números sin signo

- Rango de valores representables con N bits

$$[0, 2^N - 1]$$

Ejemplo

¿Qué rangos de valores pueden representarse con 8, 16 y 32 bits?

Bits	Rango
8	$[0, 255]$
16	$[0, 65535]$
32	$[0, 4294967295]$

Conversión binario a decimal

- Usamos el teorema fundamental de la numeración.
- Se suman los pesos (potencias de 2) de cada bit a 1.

Ejemplo

El binario 101101 equivale al decimal

$$2^5 + 2^3 + 2^2 + 2^0 = 45$$

Conversión decimal a binario

- Dividir sucesivamente entre 2 hasta que el cociente sea 0.
- Bits del número: los restos, empezando por el último.

Ejemplo

Convertir a binario el número decimal 54:

54	2					
0	27	2				
	1	13	2			
		1	6	2		
			0	3	2	
				1	1	2
					1	0

Resultado: 110110

Representación del signo en binario

- Habitualmente usamos el símbolo – para expresar negativo.
- Pero eso es un símbolo adicional a los de la base.
- En un sistema digital binario **sólo** hay **dos símbolos**.
- El signo sólo puede codificarse usando estos dos símbolos.
- Diferentes sistemas:
  - Signo y magnitud.
  - Complemento a dos.
  - Exceso M.

Signo y magnitud

- Si el sistema usa palabras de N bits:
  - Los bits de orden 0 hasta N – 2 representan el valor absoluto.
  - El bit N – 1 (MSB) representa el signo: $0 \Rightarrow +$, $1 \Rightarrow -$
  - Rango representable: $[-2^{N-1}+1, 2^{N-1}-1]$

Ejemplo

Expresar el número –85 en signo y magnitud usando 8 bits:

signo	magnitud = 85						
1	1	0	1	0	1	0	1

- **Ventaja**: codificación sencilla.
- **Inconvenientes**:
  - Existen dos representaciones del 0.
  - El signo se trata de forma distinta en las sumas y en las restas ⇒ Hardware complejo.

Complemento a dos

- Para calcular el complemento a dos:
  - Se obtiene el complemento a 1 $\Rightarrow$ invertir todos los bits.
  - Se suma 1 al resultado.

Ejemplo

Calcular el complemento a dos de 01101110 usando una palabra de 8 bits:

$$\boxed{01101110} \xrightarrow{\text{complemento a 1}} \boxed{10010001} \xrightarrow{+1} \boxed{10010010}$$

Rango de valores en complemento a dos

- Rango de valores representables en complemento a dos con N bits

$$[-2^{N-1}, 2^{N-1} - 1]$$

Ejemplo

¿Qué rangos de valores pueden representarse en complemento a dos con 8, 16 y 32 bits?

Bits	Rango
8	$[-128, 127]$
16	$[-32768, 32767]$
32	$[-2147483648, 2147483647]$

Conversión de decimal a binario en complemento a dos. Método 1

- Conversión de decimal a complemento a dos con N bits. Método 1.
  - Sea k el valor que queremos convertir.
  - k debe estar en el intervalo $[-2^{N-1}, 2^{N-1} - 1]$.
  - Si $k \geq 0$ su representación es igual al binario natural.
  - Si $k < 0$:
    - Se obtiene el complemento a 1 de $|k|$.
    - Se suma 1.
    - Se retienen sólo los N bits menos significativos.

Ejemplo

Expresar el número –3 en complemento a dos con 8 bits:

$$\boxed{3} \xrightarrow{\text{binario}} \boxed{00000011} \xrightarrow{\text{complemento a 1}} \boxed{11111100} \xrightarrow{+1} \boxed{11111101}$$

Conversión de decimal a binario en complemento a dos. Método 2

- Conversión de decimal a complemento a dos con N bits. Método 2.
  - Sea k el valor que queremos convertir.
  - k debe estar en el intervalo $[-2^{N-1}, 2^{N-1} - 1]$.
  - Si $k \geq 0$ su representación es igual al binario natural.
  - Si $k < 0$ su representación en complemento a dos es igual a la representación binaria de $2^N - |k|$.

Ejemplo

Expresar el número –3 en complemento a dos con 8 bits:

$$2^8 - 3 = 256 - 3 = 253 \xrightarrow{\text{binario}} 11111101$$

Ventajas del complemento a dos

- Una única representación del cero.
- El MSB indica el signo: 0 $\Rightarrow$ positivo, 1 $\Rightarrow$ negativo.
- El mismo hardware puede usarse para restar y sumar números positivos y negativos $\Rightarrow$ Hardware más sencillo.
- Las restas se hacen sumando el complemento a dos.

Sumas y restas en complemento a dos

Ejemplo

Sumar $3 + (-7)$ con 8 bits.
La representación de -7 en complemento a dos es 11111001.

$$\begin{array}{rr} & 00000011 \\ + & 11111001 \\ \hline & 11111100 \end{array}$$

11111100 es -4 en complemento a dos.

Ejemplo

Restar $-7 - 10$ con 8 bits.
-7: 11111001
-10: 11110110

$$\begin{array}{rr} & 11111001 \\ + & 11110110 \\ \hline & \not{1}11101111 \end{array}$$

11101111 es -17 en complemento a dos.

Exceso M

- El valor A se representa como A + M.
- M suele valer 2^{N-1}, siendo N el número de bits disponibles.

Ejemplo

$N = 8,\ M = 2^{8-1} = 128$

A	A + M	Binario
–3	125	01111101
0	128	10000000
–128	0	00000000
127	255	11111111

Sistemas de numeración intermedios

- Para los humanos el binario es tedioso.
  - Muchos dígitos hasta para números relativamente pequeños.
- El sistema decimal es más conveniente para nosotros.
  - Pero la conversión binario $\leftrightarrow$ decimal no es directa.
- Uso de sistemas intermedios: octal y hexadecimal.
  - Más compactos que el binario.
  - Conversión hacia/desde binario muy sencilla.

Sistema hexadecimal

- Usa dieciséis símbolos: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

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

- Cada cifra hexadecimal representa un bloque de cuatro bits $\Rightarrow$ La conversión hexadecimal $\leftrightarrow$ binario es muy sencilla.

Conversión hexadecimal $\leftrightarrow$ binario

Ejemplo

Convertir el binario 1101010111110011 a hexadecimal

binario	$\underbrace{1101}$	$\underbrace{0101}$	$\underbrace{1111}$	$\underbrace{0011}$
hexadecimal	D	5	F	3

Resultado: D5F3

Ejemplo

Convertir el haxadecinal 7AE6 a binario

hexadecimal	7	A	E	6
binario	$\overbrace{0111}$	$\overbrace{1010}$	$\overbrace{1110}$	$\overbrace{0110}$

Resultado: 0111101011100110

Sistema octal

- Usa ocho símbolos: $\{0, 1, 2, 3, 4, 5, 6, 7\}$

decimal	binario	octal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

- Cada cifra octal representa un bloque de tres bits $\Rightarrow$ La conversión octal $\leftrightarrow$ binario es muy sencilla.

Conversión octal $\leftrightarrow$ binario

Ejemplo

Convertir el binario 110101011111 a octal

binario	$\underbrace{110}$	$\underbrace{101}$	$\underbrace{011}$	$\underbrace{111}$
octal	6	5	3	7

Resultado: 6537

Ejemplo

Convertir el octal 241 a binario.

octal	2	4	1
binario	$\overbrace{010}$	$\overbrace{100}$	$\overbrace{001}$

Resultado: 010100001

Códigos BCD

- BCD significa Binary Coded Decimal = Decimal Codificado en Binario.
- Permite codificar números decimales en binario de una forma sencilla.
- Se codifica el valor absoluto del número. El signo, si se usa, se codifica aparte.
- Dos variantes: desempaquetado y empaquetado.
- La conversión desde/hacia decimal es muy sencilla.
- Desventajas respecto al binario natural:
  - Normalmente, el hardware no permite hacer operaciones aritméticas directamente en BCD.
  - Se desaprovecha capacidad de almacenamiento de información.

BCD desempaquetado

- Cada byte codifica una sola cifra decimal.
- Solo se aprovecha el nibble menos significativo de cada byte.
- El nibble más significativo de cada byte siempre está a 0000.
- Sólo se aprovechan 10 de las 256 combinaciones que permite cada byte.

Ejemplo

Codificar el valor decimal 9347 en BCD desempaquetado

decimal	9	3	4	7
BCD desempa.	00001001	00000011	00000100	00000111

BCD empaquetado

- Cada byte codifica dos cifras decimales
  - Una en el nibble menos significativo
  - Otra en el nibble más significativo.
- Sólo se aprovechan 100 de las 256 combinaciones que permite cada byte. (Aunque en este aspecto es mejor que el desempaquetado)
- La codificación es algo más compleja que en el BCD desempaquetado.

Ejemplo

Codificar el valor decimal 9347 en BCD empaquetado

decimal	93	47
BCD empaquetado	$\overbrace{10010011}$	$\overbrace{01000111}$

Códigos de E/S - Representación de textos

- Los textos están compuestos por símbolos gráficos llamados caracteres o glifos.
- Los caracteres tienen que representarse por números binarios.
- Código de caracteres: establece una correspondencia entre caracteres y números.
- Códigos de caracteres más importantes actualmente:
  - ASCII
  - Unicode

Código ASCII

- ASCII: American Standard Code for Information Interchange.
- Usa códigos de 7 bits (valores entre 0 y 127).
- En la práctica, cada carácter ocupa un byte completo (8 bits).
  - El código ASCII se sitúa en los 7 bits menos significativos del byte.
  - El bit 7 siempre es cero. Originalmente se usaba como bit de paridad.

Tabla del código ASCII

non-printing					printing			printing			printing		
Name	Control char	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
null	ctrl-@	NUL	00	00	SP	20	32	@	40	64	‘	60	96
start of heading	ctrl-A	SOH	01	01	!	21	33	A	41	65	a	61	97
start of text	ctrl-B	STX	02	02	"	22	34	B	42	66	b	62	98
end of text	ctrl-C	ETX	03	03	#	23	35	C	43	67	c	63	99
end of xmit	ctrl-D	EOT	04	04	$	24	36	D	44	68	d	64	100
enquiry	ctrl-E	ENQ	05	05	%	25	37	E	45	69	e	65	101
acknowledge	ctrl-F	ACK	06	06	&	26	38	F	46	70	f	66	102
bell	ctrl-G	BEL	07	07	'	27	39	G	47	71	g	67	103
backspace	ctrl-H	BS	08	08	(	28	40	H	48	72	h	68	104
horizontal tab	ctrl-I	HT	09	09	)	29	41	I	49	73	i	69	105
line feed	ctrl-J	LF	0A	10	*	2A	42	J	4A	74	j	6A	106
vertical tab	ctrl-K	VT	0B	11	+	2B	43	K	4B	75	k	6B	107
form feed	ctrl-L	FF	0C	12	,	2C	44	L	4C	76	l	6C	108
carriage return	ctrl-M	CR	0D	13	-	2D	45	M	4D	77	m	6D	109
shift out	ctrl-N	SO	0E	14	.	2E	46	N	4E	78	n	6E	110
shift in	ctrl-O	SI	0F	15	/	2F	47	O	4F	79	o	6F	111
data line escape	ctrl-P	DLE	10	16	0	30	48	P	50	80	p	70	112
device control 1	ctrl-Q	DC1	11	17	1	31	49	Q	51	81	q	71	113
device control 2	ctrl-R	DC2	12	18	2	32	50	R	52	82	r	72	114
device control 3	ctrl-S	DC3	13	19	3	33	51	S	53	83	s	73	115
device control 4	ctrl-T	DC4	14	20	4	34	52	T	54	84	t	74	116
neg acknowledge	ctrl-U	NAK	15	21	5	35	53	U	55	85	u	75	117
synchronous idle	ctrl-V	SYN	16	22	6	36	54	V	56	86	v	76	118
end of xmit block	ctrl-W	ETB	17	23	7	37	55	W	57	87	w	77	119
cancel	ctrl-X	CAN	18	24	8	38	56	X	58	88	x	78	120
end of medium	ctrl-Y	EM	19	25	9	39	57	Y	59	89	y	79	121
substitute	ctrl-Z	SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
escape	ctrl-[	ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
file separator	ctrl-\	FS	1C	28	<	3C	60	\	5C	92	|	7C	124
group separator	ctrl-]	GS	1D	29	=	3D	61	]	5D	93	}	7D	125
record separator	ctrl-^	RS	1E	30	>	3E	62	^	5E	94	~	7E	126
unit separator	ctrl-_	US	1F	31	?	3F	63	_	5F	95	DEL	7F	127

Extensiones del código ASCII a 8 bits

- Un byte puede representar valores entre 0 y 255.
- El código ASCII solo usa los códigos de 0 a 127.
- Diversas extensiones del código ASCII usan los códigos del 128 al 255 para representar caracteres adicionales.
  - ISO/IEC 8859-1 Latin-1. Caracteres de Europa Occidental.
  - ISO/IEC 8859-2 Latin-2. Caracteres de Europa Central.
  - ISO/IEC 8859-5 Latin/Cyrillic: Caracteres cirílicos.
  - CP-1552. Basado en el ISO/IEC 8859-1. Usado por Windows.

Inconvenientes del ASCII y sus extensiones

- El número de símbolos es insuficiente incluso usando extensiones.
- Algunas veces no resulta fácil saber qué extensión se ha usado.
- Están basados en caracteres occidentales: los ideogramas chinos, japoneses y coreanos no están contemplados.

Código UNICODE

- Propuesto por un consorcio de empresas y organismos.
- Pretende cubrir todos las sistemas de escritura actuales.
- Tres codificaciones **UTF-8**, **UTF-16, UTF-32**.
- **UTF-8**
  - Códigos de longitud variable de entre 1 y 4 bytes.
  - Maximiza compatibilidad con ASCII.
  - Se usa mucho en páginas web y en sistemas Unix-Linux.
- **UTF-16**
  - Códigos de longitud variable de uno o dos bloques de 16 bits.
  - Codificación usada internamente en Windows.
- **UTF-32**
  - Códigos de longitud fija de 32 bits.

Representación de números reales

- Muchas aplicaciones necesitan usar números con parte fraccionaria.
- Existen dos métodos:
  - Punto (o coma) fijo.
  - Punto (o coma) flotante.

Punto fijo

- Si las palabras usadas tiene N bits:
  - Se usan F bits menos significativos para la parte fraccionaria.
  - Se usan E = N – F bits para la parte entera.
- El signo se codifica usando complemento a dos.

Punto flotante

- Los números se almacenan en notación exponencial:

$$N = (-1)^S \cdot M \cdot 2^E$$

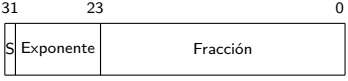
donde

S Signo: 0 $\Rightarrow$ positivo, 1 $\Rightarrow$ negativo.
M Mantisa.
E Exponente.

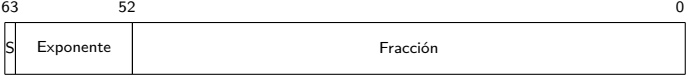
- S, M y E se almacenen en binario.

Norma IEEE 754

- Usada por la mayoría de los sistemas que emplean punto flotante.
- Formato de precisión simple de 32 bits.



- Formato de precisión doble de 64 bits.



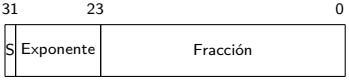
S Bit de signo.

Exponente Exponente desplazado o polarizado.

Fracción Parte fraccionaria de la mantisa normalizada.

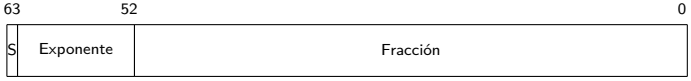
Real de precisión simple

- Se codifica con 32 bits (4 bytes).
- Se corresponde con el **tipo `float` del lenguaje C**.
- Exponente de 8 bits.
- Fracción de 23 bits.
- Precisión de 6 a 9 cifras significativas en base 10.
- Valor mínimo (> 0): $1{,}175494351 \times 10^{-38}$
- Valor máximo: $3{,}402823466 \times 10^{38}$

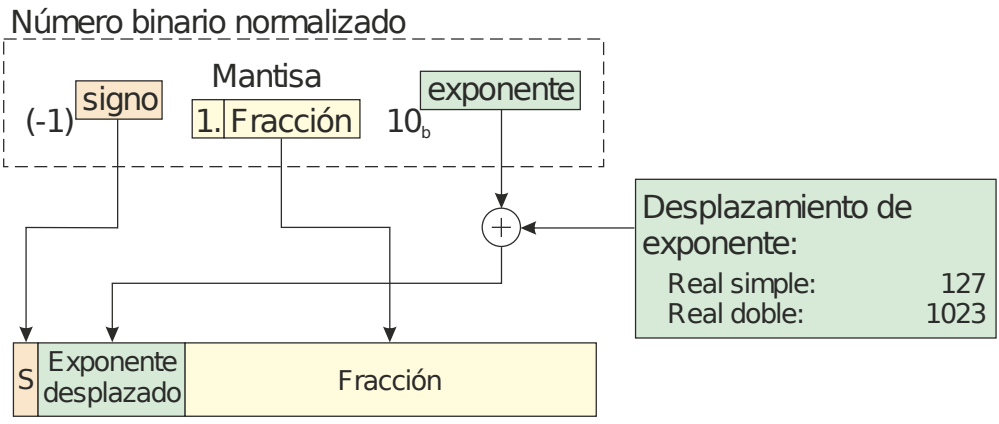


Real de precisión doble

- Se codifica con 64 bits (8 bytes).
- Se corresponde con el **tipo `double` del lenguaje C**.
- Exponente de 11 bits.
- Fracción de 52 bits.
- Precisión equivalente de 15 a 17 cifras significativas en base 10.
- Valor mínimo (> 0): $2{,}2250738585072014 \times 10^{-308}$
- Valor máximo: $1{,}7976931348623158 \times 10^{308}$



Codificación IEEE754

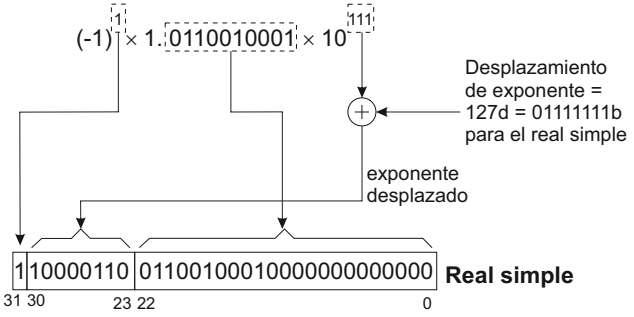


Ejemplo de codificación de un real simple (float)

- Codificación del número decimal -178.125 en un real simple (float).

$$-178.125_d = -10110010.001_b =$$

$$= (-1)^1 \times 10110010.001_b \times 10_b^0 \xrightarrow{\text{Normalización}} (-1)^1 \times 1.0110010001_b \times 10_b^{111_b}$$



En hexadecimal: C3322000h