

Tema-2-Representacion-de-la-info...



atmuntenas



Fundamentos de Informática



1º Grado en Ingeniería de las Tecnologías de Telecomunicación



**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada**



[Accede al documento original](#)

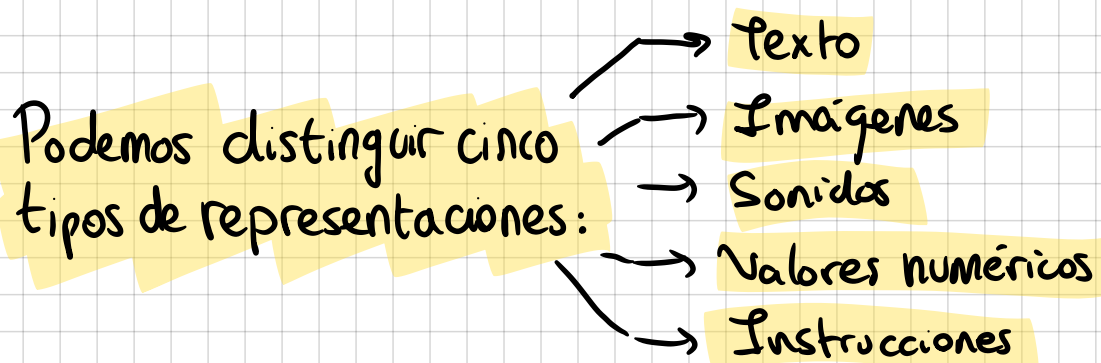
TEMA 2: Representación de la información en computadores

Dos de los aspectos más importantes que se presentan en informática relacionados con la información es cómo REPRESENTARLA y cómo REGISTRARLA físicamente. En este tema abordaremos el primer problema.

En general, el diseño de un sistema informático resulta sencillo. Se hace uso del sistema menos complejo y más fiable, el sistema binario.

El sistema binario es simple, tenemos dos valores, 0 y 1, que físicamente se corresponden a dos niveles de tensión (ej: 0 y 3'5 voltios), de corriente (ej: 0 y 16 mA), de magnetización (N y S)...

Esto implica que la información es retenida por medio de dos valores de magnitud física (BIT) que las podemos representar por cero y uno.



Otros aspectos adicionales relacionados con la REPRESENTACIÓN de la información en el interior del computador: DETECCIÓN DE ERRORES Y COMPRESIÓN DE DATOS.

REPRESENTACIÓN DE TEXTOS

Podemos representar cualquier información escrita por medio de caracteres, solemos agruparlos en cinco categorías:

→ Caracteres alfabéticos: Son las letras mayúsculas y minúsculas del

abecedario inglés (A, B, C, ..., a, b, c, ...)

→ **Caracteres numéricos:** constituido por las diez cifras decimales (0,1,2,3,4,5...)

→ **Caracteres especiales**: son símbolos ortográficos y matemáticos no incluidos en los grupos anteriores $() \{ * / ; : + \tilde{N} \pi = ! ? . " > \# \underline{SP} \dots$
espacio blanco

→ **Caracteres geométricos y gráficos:** símbolos o módulos con los que se pueden representar marcos de cuadros o de tablas, formas geométricas o iconos elementales como:

→ **Caracteres de control**: representan órdenes de control, como el carácter para pasar a la línea siguiente (NL), o para ir al comienzo de una línea (CR) o para la sincronización de una transmisión (SYN), etc.

Al introducir un texto en un computador los caracteres se codifican con un **CÓDIGO DE ENTRADA/SALIDA**, de forma que a cada carácter se le asocia una determinada combinación de n bits. Un código de E/S es una correspondencia entre dos conjuntos:

* $\alpha \rightarrow$ conjunto de caracteres (símbolos) que deseamos codificar.

* $\beta \rightarrow$ conjunto de combinación binarias de n bits ($\{0,1\}^n$).

Con n bits podemos representar m símbolos distintos, conforme a la siguiente relación matemática: $m = 2^n / 2^{n-1} < m < 2^n$. Para calcular cuántos bits son necesarios para representar m símbolos, $n \geq \log_2(m) = 3.32 \cdot \log(m)$, $n \in \mathbb{N}$

Importante, n es un número natural (entero positivo).

Los códigos E/S más utilizados en la actualidad son:

⊗ Nota editor: ASCII y EBCDIC codifican códigos de control.

→ Código Extended Binary Coded
Decimal Interchange Code (EBCDIC)

utiliza 8 bits para representar cada

carácter, pudiendo representar 2^8 símbolos distintos.

→ Código American Standard Code for Information Interchange (ASCII)
básico utiliza 7 bits, aunque se hace uso de un octavo bit para detectar errores de transmisión o grabación (luego lo veremos).

Bits de zona		Bits de posición				
0	1	2	3	4	5	6
00 Caracteres sin usar		00 caracteres A-I				
01 Caracteres especiales		01 caracteres J-R				
10 Letras mayúsculas		10 caracteres S-Z				
11 Letras minúsculas		11 caracteres numéricos				

Figura 4.1. Significado de los campos en la codificación EBCDIC.

Es de los códigos más utilizados, la mayor parte de los dispositivos lo usan para transmitir datos y corresponde a la norma **ISO 646**. Existen varias versiones ampliadas de este código que utilizan 8 bits y respetan la norma del ASCII básico. Destacan los estándares **ISO 8859-n**, donde n€/N representa el juego de caracteres usados.

Tabla 4.5. Familia de la normalización ISO 8859, ampliaciones de ASCII.

Denominación	Estándar	Área geográfica
Latín-1	ISO 8859-1	Oeste y Europa del este
Latín-2	ISO 8859-2	Europa central y del este
Latín-3	ISO 8859-3	Europa sur, maltés y esperanto
Latín-4	ISO 8859-4	Europa norte
Alfabeto latín/cirílico	ISO 8859-5	Lenguajes eslavos
Alfabeto latín/árabe	ISO 8859-6	Lenguajes arábigos
Alfabeto latín/griego	ISO 8859-7	Griego moderno
Alfabeto latín/hebraico	ISO 8859-8	Hebreo y Yiddish
Latín-5	ISO 8859-9	Turco
Latín-6	ISO 8859-10	Nórdico (Sámi, Inuit e islandés)
Denominación	Estándar	Área geográfica
Alfabeto Latín/Thai	ISO 8859-11	Lenguaje Thai
Latín-7	ISO 8859-13	Báltico <i>Rim</i>
Latín-8	ISO 8859-14	Céltico
Latín-9 (<i>alias Latín-0</i>)	ISO 8859-15	Latín 1 con ligeras modificaciones (símbolo €)

→ Código **UNICODE** es un código E/S que surgió ante la necesidad de poder representar más caracteres y dar cabida a un sistema compatible con la vasta variedad cultural mundial. Está reconocido como estándar **ISO 10646**. Se basa en los siguientes valores:

* Universalidad [trata de cubrir la mayoría de lenguajes escritos en la actualidad]

* Unicidad [a cada caracter le corresponde un código único]

* Uniformidad [todos los símbolos se representan con 16 bits]

No contempla la codificación de caracteres de control, permite la combinación de caracteres, con un código de combinación único, no determina cómo es la forma del

carácter y evita la duplicación de caracteres idénticos para varios idiomas.

Tabla 4.7. Esquema de asignación de códigos en Unicode.

Zona	Códigos		Símbolos codificados	Nº de caracteres
A	0000	0000 a 007F	Latín Básico (00 a 7F), definido en la norma ASCII ANSI-x3.4	128
		0080 a 00FF	Suplemento Latín-1 (ISO 8859-1)	128
		0100 a 017F	Ampliación A de Latín	128
		0180 a 024F	Ampliación B del Latín	80
		0250 a 02AF	Ampliación del Alfabeto Fonético Internacional (IPA)	96
		02BF a 02FF	Espaciado de letras modificadoras	65
		0300 a 036F	Combinación de marcas diacríticas (tilde, acento grave, etc.)	112
		0370 a 03FF	Griego	144
		0400 a 04FF	Cirílico	256
		0530 a 058F	Armenio	96
		0590 a 05FF	Hebreo	112
		0600 a 06FF	Árabe	256
		0700 a 074F	Sirio	80
			etc.	
	3FFF	2000 a 3FFF	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos	8 192
I	4000			
	9FFF	Ideogramas		24 576
O	A000			
	DFFF	Pendiente de asignación		16 384
R	E000			
	FFFF	Caracteres locales y propios de los usuarios. Compatibilidad con otros códigos		8 192

REPRESENTACIÓN DE SONIDOS

Las aplicaciones multimedia han adquirido una gran importancia, y son capaces de procesar textos, imágenes y sonido.

Una señal de sonido es de carácter ANALÓGICO, es una señal que puede tomar cualquier valor dentro de un determinado intervalo continuo. Para trabajar la señal solemos amplificarla, y posteriormente, aplicar un proceso de digitalización **MUESTREO**.

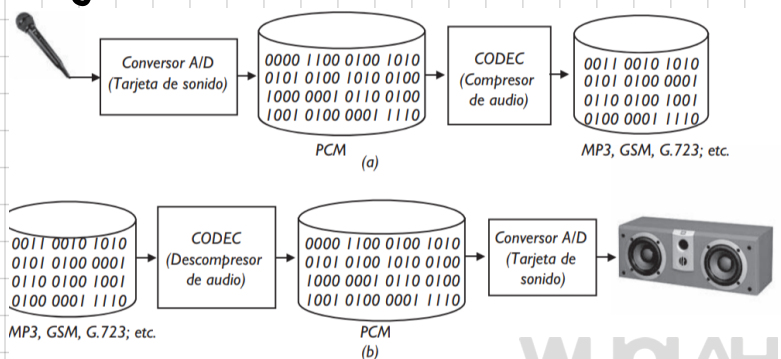
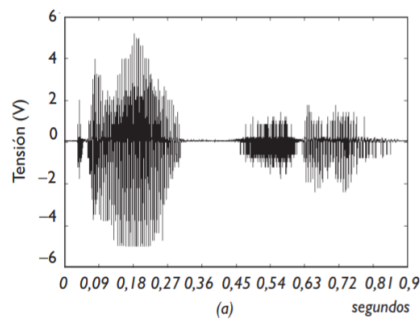
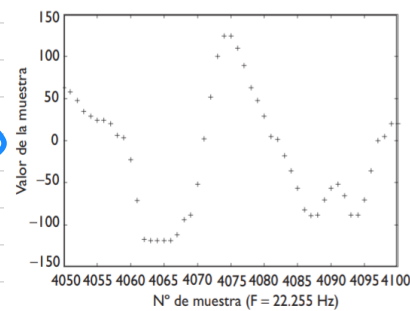


Figura 4.3. Esquema simplificado de un sistema de (a) grabación digital de sonido, (b) reproducción digital de sonido.

El **muestreo** es un proceso por el se digitaliza una señal analógica. En un intervalo de tiempo se disponen de ∞ valores de señal analógica, y para poder almacenarla digitalmente, seleccionamos muestras de la señal a una determinada frecuencia, así cada $T_s = \frac{1}{F_s}$ segundos se dispone de un valor de señal.



Posición	contenido
1	2
2	2
3	2
4	2
.	.
4050	63
4051	58
4052	48
4053	35
4054	29
4055	24
4056	24



Posición	contenido
4057	20
4058	6
4059	3
4060	-23
4061	-71
4062	-118
.	.
19996	1
19997	1
19998	1
19999	1
20000	1

La definición matemática de la muestra dice que es un conjunto de determinado tamaño menor que la población. En este caso, la población es la señal analógica, y la muestra es el intervalo de señal de t segundos. En el muestreo lo que hacemos es quedarnos con el valor $\oplus$ significativo del intervalo.

Paralelamente transformamos esos valores muestreados en bits con un CONVERSOR ANALÓGICO/DIGITAL.

Por tanto, el sonido en la informática es una secuencia de valores que corresponden a una muestra analógica.

En el muestreo intervienen:

- **FRECUENCIA DE MUESTREO** → Cuanto más alta, más muestras tengo de la señal, y por tanto, el análisis es más fiable a la forma de señal original.
- **NÚMERO DE BITS** con la que se representa cada muestra, debe ser lo suficiente para recuperar la señal con la calidad requerida.

Para procesar la señal analógica producida por un conversor A/D o para generar la señal a aplicar a un conversor D/A, se utiliza un circuito denominado CODEC (**C**odificador / **D**ECodificador o **C**ompresor / **D**ECompresor).

Hay varios tipos de CODEC, destacando:

- **Pulse Code Modification**: se transmiten o graba un tren de pulsos correspondientes a cada muestra.

→ **Differential Pulse Code Modulation** es un códec que disminuye el tamaño de los archivos, almacena la diferencia entre los valores de la muestra y no el valor absoluto (ej: Es como las graficas de los stocks, en diagramas de )

→ **Adaptive DPCM** es un algoritmo que predice el valor de muestra siguiente y almacena el error entre el valor predicho y real.

→ **μ -law**, que es parecido al ADPCM.

→ **MPEG Audio Layer-III**, para formatos mp3 y AAC, y funciona en la premisa de que hay determinados rangos de sonidos que normalmente no se llegan a escuchar, y por tanto, prescinde de las muestras que sobrepasa los límites de escucha promedio.

→ **CÓDECS propietarios**, como WMA (Microsoft) o Real Networks (APPLE).

Definimos la tasa de bits (**bit rate**) como el número de bits que se transfieren por segundo (b/s o bps), y viene dado por la siguiente relación:

$$R = F_s \cdot n \rightarrow R \text{ tasa de datos}$$

F_s frecuencia de muestreo
 n número de bits por muestreo.

REPRESENTACIÓN DE IMÁGENES

Las imágenes las podemos representar de dos formas, mediante:

→ **MAPAS DE BITS**. Una imagen que vemos con nuestros propios ojos tiene infinitos detalles. Al igual que el sonido, afrontamos limitaciones técnicas de la tecnología, y por ello, haremos uso de un "**proceso de muestreo**", que consiste en dividir una imagen en una fina retícula de celdas (**PIXELES**) y atribuir a cada uno de ellos el nivel de gris o el color correspondiente. Las fotos de color, esta se suele descomponer en tres colores básicos: Rojo, Verde y Azul (RGB); y la intensidad media de cada uno de ellos en cada celda se codifica por separado.

La **resolución de imagen** [nº píxeles por línea x nº de píxeles columnas] determina la calidad de imagen.

Otro parámetro destacable es el **atributo de píxel**, describe las características visuales del píxel, como su color o tonalidad. Por ejemplo, para conseguir una gran calidad de colores, la intensidad de cada color se codifica con 8 bits, como tenemos 3 colores, 3 bytes por píxel para codificar una imagen.

Hay formatos que permiten que el píxel sea transparente. Estos puntos se codifican con un cuarto valor llamado **canal alfa (α)**. Entonces, este parámetro recoge como de opaco o transparente es un píxel.

Tabla 4.11. Resoluciones usuales para codificar imágenes.

		Resolución (horizontal × vertical)	Movimiento
Convencionales	Fax (A4)	(100, 200, 400) × (200, 300, 400) píxeles/pulgada	Estática
	Foto (8" × 11")	128, 400, 1200 píxeles/pulgada	Estática
Televisión	Videoconferencia	176 × 144 píxeles/imagen	10 a 36 imágenes/s
	TV	720 × 480 píxeles/imagen	30 imágenes/s
	HDTV (TV alta definición)	1 920 × 1 080 píxeles/imagen	30 imágenes/s
Pantalla computador	VGA	640 × 480 píxeles	
	SVGA	800 × 600 píxeles	
	XGA	1 024 × 768 píxeles	

→ **MAPA DE VECTORES**. Otra forma de representar una imagen consiste en descomponerla en una colección de objetos tales como líneas, polígonos y textos con sus respectivos atributos, moldeables por medio de vectores y ecuaciones matemáticas que determinan su forma y posición dentro de la imagen. Cuando vemos la imagen, un programa evalúa las ecuaciones y escala los vectores a creando una imagen concreta a ver.

Este tipo de representación es adecuado para gráficos de tipo geométrico, y usualmente se generan archivos más ligeros y las imágenes escaladas **NO PIERDEN CALIDAD**.

REPRESENTACIÓN DE DATOS NUMÉRICOS

Los datos numéricos se introducen usando un lenguaje escrito, como secuencias de caracteres, y, por tanto, se codifican de acuerdo con un código E/S.

Sin embargo, si se va a realizar algún cálculo matemático, la representación de los datos numéricos como textos es inapropiada, pues no podemos aplicar las tablas y reglas de la aritmética para operar, puesto que la codificación no se basa en los sistemas de numeración.

EJEMPLO 4.9

Representar en ASCII (8 bits) los números decimales 255 y 42, y obtener el resultado de sumarlos como si estuviesen en el sistema de numeración base 2.

En la segunda columna de la siguiente tabla se tiene la representación ASCII de las cifras de los números correspondientes, que se obtiene sin más que consultar la tabla de códigos ASCII (8 bits) que aparece en la Tabla 4.6. La última fila contiene el resultado de sumar las representaciones binarias como si fuesen números del sistema de numeración base 2. Consultando nuevamente la tabla de códigos ASCII comprobamos que el resultado corresponde a la cadena de caracteres 2ig; muy lejos del resultado buscado que es 297.

Números (caracteres)	Representación en ASCII	Resultado en ASCII
255	0011 0010 0011 0101 0011 0101	
42	0011 0100 0011 0010	
Suma →	0011 0010 0110 1001 0110 0111	→ 2ig

no bits → no bits

EJEMPLO 4.10

Representar en los números decimales 255 y 42 en binario natural, y obtener el resultado de sumarlos en base 2.

Los resultados se muestran en la siguiente tabla:

Números	Representación en base 2	Resultado en decimal
255	1111 1111	
42	0010 1010	
Suma →	1 0010 0111	→ 297

Cuando queremos realizar un cálculo matemático, lo mejor es representar los datos numéricos en alguna forma que esté basada en el sistema de numeración matemático.

Usando el sistema de numeración base 2 se obtiene un resultado correcto.

La solución adoptada para representar datos numéricos consiste en almacenarlos como texto. Dentro de un programa, cada dato debe estar asociado a un tipo específico, que determina las operaciones posibles, teniendo en cuenta el rango y precisión numérica. Ver ejemplo de los principales datos aritméticos utilizables en C++.

Tabla 4.12. Principales tipos de datos aritméticos utilizables en el lenguaje de programación C++ (compilador Borland para PC).

	Tipo	Nº de bits	Rango de valores	Precisión (dígitos decimales)
Tipos enteros	Carácter	8	-128,127	3
	Carácter sin signo	8	0 a 255	3
	Entero corto	16	-32.768 a 32.767	3
	Entero corto sin signo	16	0 a 65.535	5
	Enumerado	16	-32.768 a 32.767	5
	Entero	*	*	*
	Entero sin signo	*	*	*
	Entero largo	32	-2.147,484.648 a 2.147,484.648	10
	Entero largo sin signo	32	0 a 4.294,967.295	10
Tipos reales	Coma flotante	32	$\pm [3,4E - 38 \text{ a } 3,4E38]$, 0	7
	Coma flotante doble	64	$\pm [1,7E - 308 \text{ a } 1,7E308]$, 0	15
	Coma flotante doble largo	80	$\pm [3,4E - 4932 \text{ a } 1,1E4932]$, 0	19

* En máquinas de 16 bits igual a entero corto, y en máquinas de 32 bits a entero largo.

En los programas existen reglas que determinan la tipología del dato introducido. Por ejemplo, Microsoft Excel sigue las siguientes reglas:

- El texto tecleado: {
- Deberá comenzar con un dígito, o uno de los siguientes caracteres: +, -, =, ,
 - Puede terminar con el símbolo %
 - Puede tener como máximo una coma decimal
 - Puede introducirse en *notación científica*; por ejemplo: 2,7E-15

El dato completo debe encajar en datos de memoria, si el tamaño es mayor, se troceará convenientemente, y dichos trozos se almacenarán en posiciones consecutivas, siguiendo, o bien el **CRITERIO DEL EXTREMO MENOR** [almacenamos primero la parte menos significativa del dato] o **MAYOR** [lo contrario, primero lo más importante].

Una vez definidos los datos, un proceso se encarga de convertir la cadena de caracteres que simboliza el número en la representación numérica.

Podemos representar los datos numéricos como:

→ **NÚMERO NATURAL**, distinguimos dos formas básicas de representación:

Tabla 4.14. Diferentes representaciones de datos enteros.

Representación binaria	– Enteros sin signo	• Signo y magnitud • Complemento a 1 • Complemento a 2 • Sesgada (o en exceso)
	– Enteros con signo	
Representación de dígitos decimales codificados en binario (BCD)	– BCD natural	
	– BCD Aiken	
	– BCD exceso en 3	

ENTEROS SIN SIGNOS

En este caso todos los bits del dato representan el valor del número expresado en binario natural (sistemas de numeración base 2).

Entonces, como hay n bits para representar el número, los valores mayor y menor representables son: $N(\min) = 0$; $N(\max) = 2^n$.

ENTEROS EN SIGNO Y MAGNITUD

El signo se representa con el bit más significativo del dato (bit $n-1$). Este bit es 0 si el número es positivo y 1 si el número es negativo. El resto de los bits ($n-2$ a 0) representan el valor absoluto del número en binario natural.

Sin embargo, esta representación puede dar problemas, pues para el número cero existen dos representaciones: $+0$ y -0 , provocando la necesidad de verificar siempre que el resultado de una operación se verifique para ambos casos.

En general, como hay $n-1$ bits para representar la magnitud del número,

$$N(\min) = -(2^{n-1} - 1); N(\max) = 2^{n-1} - 1$$

ENTEROS EN COMPLEMENTO A 1

La diferencia con el anterior es que, los bits $n-2$ a 0 representan:

* Si $N > 0$, el valor absoluto del número en binario natural.

* Si $N < 0$, su complemento a 1.

En general, como hay $n-1$ bits para representar la magnitud del número,

$$N(\min) = -(2^{n-1} - 1); N(\max) = 2^{n-1} - 1$$

ENTEROS EN COMPLEMENTO A 2

Lo mismo que el anterior, solo que complemento de dos. Pero con una ligera diferencia, el número cero solo puede ser representado de una sola forma.

Además, el valor máximo y mínimo representables son: $N(\text{mín}) = -2^{n-1}$
 $N(\text{máx}) = 2^{n-1} - 1$

REPRESENTACIÓN SESGADA (REPRESENTACIÓN CON EXCESO)

Le sumamos al número N un sesgo S , de forma que el número resultante siempre es positivo, no siendo necesario reservar explícitamente un bit de signo. El dato almacenado es el valor $N+S$ en binario natural. El sesgo suele ser $S = 2^{n-1}$.

Los números positivos empiezan por 1 y los negativos por 0. Los números mayor o menor representables son: $N(\text{mín}) = -2^{n-1}$ $N(\text{máx}) = 2^{n-1} - 1$.

EJEMPLO

Obtener la representación del nº entero $M=87$ y $N=-87$, usando 16 bits, en las cuatro formas vistas. (HEX)

Paso a binario natural.

$$\begin{array}{r} 87 \div 2 = 43 \text{ r } 1 \\ 43 \div 2 = 21 \text{ r } 1 \\ 21 \div 2 = 10 \text{ r } 1 \\ 10 \div 2 = 5 \text{ r } 0 \\ 5 \div 2 = 2 \text{ r } 1 \\ 2 \div 2 = 1 \text{ r } 0 \\ 1 \div 2 = 0 \text{ r } 1 \end{array} \quad \begin{array}{r} -87 \div 2 = -43 \text{ r } -1 \\ -43 \div 2 = -21 \text{ r } -1 \\ -21 \div 2 = -10 \text{ r } -1 \\ -10 \div 2 = -5 \text{ r } 0 \\ -5 \div 2 = -2 \text{ r } 1 \\ -2 \div 2 = -1 \text{ r } 0 \\ -1 \div 2 = 0 \text{ r } 1 \end{array}$$

$$87_{10} = 1010111_2$$

Represento (uso calculadora para HEX)

Signo y magnitud:

M ⇒	0	00000000	1010111	H'0057
N ⇒	1	00000000	1010111	H'8067

Complemento a 1:

M ⇒	0	00000000	01010111	H'0057
N ⇒	1	11111111	10101000	H'FFA8

Aquí tan fácil como cambiar los ceros por unos y los unos por ceros.

Complemento a 2:

M ⇒	0	00000000	01010111	H'0057
N ⇒	1	11111111	10101001	H'FFA9

Complemento de 1 + 1

→ Ssgado:

S →	1000 0000 0000 0000		
M →	+ 1000 0000 0101 0111	M + S	
S + M →	1000 0000 0101 0111	1000 0000 0101 0111	→ H'8057
S →	1000 0000 0000 0000		
N →	- 1000 0000 0101 0111	N + S	
S + N →	0111 1111 1010 1001	0111 1111 1010 1001	→ H'7FA9

En caso de calcular un número tan grande, fuera del intervalo posible, la ALU genera un patrón binario no válido, y se dice que se ha producido un desbordamiento, y se debe desecharse.

La forma más usada para representar los números enteros con signo es la de complemento a dos. La segunda, en la representación de exponentes de los datos de tipo real.

DATOS ENTEROS REPRESENTADOS CON DÍGITOS DECIMALES CODIFICADOS EN BINARIO (BCD).

Los datos se representan codificando aisladamente cada dígito. En un byte se pueden representar dos dígitos decimales [BCD Empaquetado], o bien un único dígito decimal [BCD Desempaquetado]. Los datos con signo se suelen utilizar 4

9 8 3 2 5 = BCD'1001 1000 0011 0010 0101
1001 1000 0011 0010 0101

bits para representar el signo:

ejemplo: 0000 para el signo positivo
1001 para el signo negativo

→ **DATOS DE TIPO REAL.** Cuando se opera con números muy grandes se suele emplear la **NOTACIÓN EXPONENCIAL/CIENTÍFICA/COMA FLOTANTE**. A día de hoy los procesadores contienen un **PROCESADOR DE COMA FLOTANTE (FPU/Float Point Unit)** dedicado a procesar este tipo de datos.

IEEE 754 - Representación de datos de tipo real.

Todo número N lo podemos expresar de la forma $N = M \cdot B^E$, donde M es la mantisa y E el exponente. El IEEE 754 establece las siguientes normas de representación:

ELEMENTOS ALMACENADOS Y ORDEN DE ALMACENAMIENTO.

La base $B=2$, solo es necesario almacenar M y E con sus respectivos

signos. Los elementos sufren una transformación y se almacena los denominados **campo del signo (s), campo del exponente (e) y campo de la mantisa (m)** que contiene las cifras más significativas del número. Se utiliza un bit para el signo,

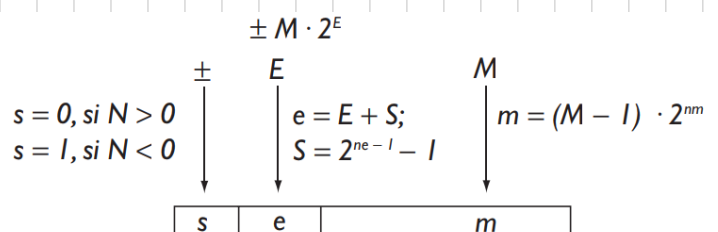


Figura 4.10. Esquema de la representación IEEE 754.

ne bits para el exponente y nm bits para el campo de la mantisa, con n el número total de bits utilizados para representar el número real, verifica: $n = 1 + ne + nm$.

Se utiliza este orden de almacenamiento para que lo más significativo quede de Izq. a Der., y así los algoritmos de comparación para números enteros se puedan emplear para la representación de números reales.

CAMPO DEL SIGNO

Cero para los positivos y uno para los negativos.

CAMPO DEL EXPONENTE

El exponente se almacena de forma sesgada, permitiendo en los ne bits reservados la inclusión de exponentes positivos y negativos sin bits adicionales.

CAMPO DE LA MANTISA

Por lo general el exponente se ajusta de forma que el 1 más significativo se encuentre en la posición cero (pos. de las uds.) Si el número está ajustado se dice que está **NORMALIZADO**, al contrario **DESNORMALIZADO**.

Resumidas cuentas, reajustar la mantisa y exponente de forma que:

$$\text{con } M = [1, m], \quad 1 \leq M < 2.$$

Las normalizaciones son necesarias para no perder precisión (cifras significativas) en operaciones sucesivas.

CASOS ESPECIALES

a) Número denormalizado	→	0 0000 0000 m ≠ 0
b) Cero	→	0 0000 0000 000 0000 0000 0000 0000 0000
c1) Más Infinito (+ ∞)	→	0 1111 1111 000 0000 0000 0000 0000 0000
c2) Menos Infinito (- ∞)	→	1 1111 1111 000 0000 0000 0000 0000 0000
c3) Número indeterminado (NaN)	→	0 1111 1111 m ≠ 0

Figura 4.11. Patrones de bits asociados a situaciones especiales en la notación IEEE 754 (suponemos ne = 8 bits y nm = 23 bits).

REDONDEOS

@atmuntenas de distritounico.es

Un problema al representar números reales es que no se pueden expresar con precisión exacta usando un número fijo de cifras binarias, lo que obliga a usar técnicas de redondeo. Durante las operaciones, la ALU o FPU trabaja con más bits para mayor precisión, pero al almacenar el resultado final también realiza un redondeo, introduciendo errores.

Existen dos técnicas de redondeo:

→ POR DEFECTO (TRUNCAR)

→ POR EXCESO

la más común es el redondeo al más próximo, minimizando el error de aproximación, según este criterio:

→ Si la primera cifra despreciada (cifra de guarda) es mayor que 5, se redondea por exceso.

→ Si es menor que 5, se redondea por defecto.

El redondeo en sistemas binarios introduce errores acumulativos, especialmente en cálculos iterativos, y es relevante cuando la cifra de guarda representa la mitad del peso del bit menos significativo retenido. Para evitar sesgos, el estándar recomienda el REDONDEO AL PAR, consiste en redondear por exceso o defecto, siempre de forma que el bit menos significativo del número resultante sea 0, que distribuye equitativamente los errores. Este método considera bits adicionales, guarda (posición $-(nm+1)$) y redondeo (posición $-(nm+2)$), para garantizar precisión y minimizar tendencias acumulativas.

PRECISIÓN

El estándar IEEE 754 define cuatro niveles de precisión para representar números en coma flotante: simple, simple ampliada, doble, doble ampliada, aunque formalmente solo se especifica la simple y la doble. Estos se distinguen por el número de bits usados para la representación:

a) Simple precisión [32 bits]:

WUOLAH

* Bits de precisión ($nm+1$): 24

* Exponentes $\rightarrow$ Máximo [$E_{\max}$]: 127
 $\rightarrow$ Mínimo [$E_{\min}$]: -126

* Sesgo del exponente (S): 127.

* Formatos $\rightarrow$ $e = 255$ y $m \neq 0 \rightarrow \text{NaN}$
 $\rightarrow$ $e = 255$ y $m = 0 \rightarrow \infty$
 $\rightarrow$ $0 < e < 255 \rightarrow$ número normalizado.
 $\rightarrow$ $e = 0$ y $m \neq 0 \rightarrow$ número desnormalizado.
 $\rightarrow$ $e = 0$ y $m = 0 \rightarrow$ el 0.

b) Doble precisión (64 bits):

* Bits de precisión ($nm+1$): 53

* Exponentes $\rightarrow$ Máximo [$E_{\max}$]: 1023
 $\rightarrow$ Mínimo [$E_{\min}$]: -1022

* Sesgo del exponente (S): 1023.

* Formatos $\rightarrow$ $e = 2047$ y $m \neq 0 \rightarrow \text{NaN}$
 $\rightarrow$ $e = 2047$ y $m = 0 \rightarrow \infty$
 $\rightarrow$ $0 < e < 2047 \rightarrow$ número normalizado.
 $\rightarrow$ $e = 0$ y $m \neq 0 \rightarrow$ número desnormalizado.
 $\rightarrow$ $e = 0$ y $m = 0 \rightarrow$ el 0.

VALORES LÍMITES

Con toda representación se obtienen unos valores máximos y mínimos representables. Considerando las situaciones especiales y el número de bits de cada campo, podemos obtener los números máximos y mínimos representables.

El valor del número mayor $N(\text{máx}) = m(\text{máx}) \cdot 2^{E(\text{máx})}$

$\rightarrow$ la mantisa mayor $m(\text{máx}) = 2 - 2^{-nm}$

$\rightarrow$ exponente $E(\text{máx}) = e(\text{máx}) - S$

WUOLAH

El valor del número más pequeño próximo a cero

→ Normalizado: $M(\text{mín}) = 1$

$$E(\text{mín}) = 1 - S$$

$$N(\text{mín}, \text{nor}) = 1 \cdot 2^{E(\text{mín})}$$

→ Desnormalizado: $M(\text{mín}) = 2^{-nm}$

$$E(\text{den}) = 1 - S$$

$$N(\text{mín}, \text{den}) = M(\text{mín}) \cdot 2^{E(\text{den})}$$

Los números reales que cumplen las siguientes condiciones no serán representados:

⊗ $-N(\text{mín}, \text{den}) < N < N(\text{mín}, \text{den})$; $N \neq 0$ Si el resultado se produce en ese intervalo, se dice que se produce un **AGOTAMIENTO** (underflow).

Se llaman zonas de **AGOTAMIENTO GRADUAL A CERO** los intervalos:

$$N(\text{mín}, \text{den}) < N < N(\text{mín}, \text{nor}) \text{ y } -N(\text{mín}, \text{nor}) < N < -N(\text{mín}, \text{den})$$

⊗ $N < -N(\text{máx})$ y $N > N(\text{máx})$, $N \neq \pm\infty$.

Si el resultado de una operación se encuentra en esa zona, se produce un **DESBORDAMIENTO** (overflow).

CONSIDERACIONES SOBRE LA PRECISIÓN

Aunque no limitado a:

→ La precisión puede perderse al restar números muy similares o al dividir un número mucho menor que el divisor.

→ El desbordamiento ocurre con resultados excesivamente altos, como al dividir un número grande entre otro mucho menor o al hacer sumas/productos de números grandes.

→ La representación inexacta de las mantisas puede generar errores al comparar números, ya que el computador solo considera dos números iguales si todos sus bits coinciden.

→ Las operaciones con números reales pueden no cumplir las propiedades asociativa y distributiva, resultando en diferentes resultados según el orden de las operaciones.

DETECCIÓN DE ERRORES

Como hemos visto antes, para representar m símbolos distintos al menos n bits, siendo n el menor natural que verifica: $n \geq \log_2 m = 3.32 \log(m)$, $n \in \mathbb{N}$.

Pero a veces no se usan todas las combinaciones posibles de los n bits. Cuanto menos combinaciones se desperdician, se dice que el código más eficiente.

La EFICIENCIA DE UN CÓDIGO (τ) se define con

$$\tau = \frac{\text{nº de símbolos representados}}{\text{número de símbolos que se pueden representar}} = \frac{m}{2^n}, \quad 0 \leq \tau \leq 1$$

Cuanto más eficiente, τ se acercará más a 1.

Un código poco eficiente se dice que es redundante, definiéndose la redundancia:

$$R_d = (1 - \tau) \cdot 100\%$$

A veces la redundancia no es mala, se introduce a veces para poder detectar errores de transmisión. Estas redundancias se efectúan conforme a un algoritmo predeterminado, permitiendo la comprobación automática de los códigos mediante circuitos adecuados.

Uno de los algoritmos más conocidos añade al código inicial de cada carácter un nuevo bit llamado **BIT DE PARIDAD**. Existen dos criterios para introducir este bit:

→ **PARIDAD PAR**: se añade un bit (0 o 1) de forma tal que el número total de unos de código que resulte par.

→ **PARIDAD IMPAR**: se añade un bit (0 o 1) de tal forma que el número total de unos del código resulte impar.

Veamos el siguiente ejemplo:

WUOLAH

Mensaje o código inicial	Mensaje o código con bit de paridad (criterio par)
100 0001	0100 0001
010 1111	1010 1111
110 1000	1110 1000
111 0111	0111 0111

en rojo subrayo el nº de unos

dos unos, cero el bit de paridad por ya hay un nº par de unos

tres unos, uno el bit de paridad, ahora hay un número par de unos

Bit de paridad

Mensaje o código inicial	Mensaje o código con bit de paridad (criterio impar)
100 0001	1100 0001
010 1111	0010 1111
110 1000	0110 1000
111 0111	1111 0111

lo mismo que el anterior, solo que IMPAR

Bit de paridad

El bit de paridad se introduce antes de transmitir o grabar la información. Al leer la información se detectaría el error, en el caso de la transmisión se volvería a solicitar la información.

Existen otros sistemas para introducir redundancias, como el de **VERIFICACIÓN DE CUENTA FIJA**. Este tipo de sistema de codificación utiliza un número fijo y predeterminado de unos y ceros para representar cada carácter, donde la posición relativa determina el carácter específico. Los circuitos comprueban si el número de unos y ceros coincide con el número prefijado, asegurando la integridad de la codificación en la recepción o lectura.

Otro ejemplo de detección de errores es

Detección de errores CRC-CITT (Comité Consultatif International Téléphonique et Télégraphique)

- Se demuestra que utilizando el polinomio generador en una codificación polinómica (CRC)
 - CRC-CCITT $\rightarrow G(x) \equiv x^{16} + x^{12} + x^5 + 1$
- se detectan:
 - todos los errores de 1 bit individual,
 - todos los errores que alteren dos bits,
 - todos los errores que alteren un número impar de bits,
 - todos los errores de ráfagas de 16 o menos bits,
 - el 99.997% de errores en ráfagas de 17 bits, y
 - el 99.998% de errores en ráfagas de 18 o más bits.

También existen **CÓDIGOS CORRECTORES DE ERRORES** con la que se introducen las suficientes redundancias para una vez detectado el error, corregirlo y recuperar la información. Destaca el **código Hamming**.

También existen códigos que permiten cifrar la información, de forma que no pueda ser descifrada por intrusos.

COMPRESIÓN DE DATOS

@atmuntenas de distritounico.es

Existe un conjunto de técnicas para reducir el tamaño de un archivo. La transformación se denomina **COMPRESIÓN DE DATOS**. Cuando se almacena un archivo se comprime mediante un algoritmo de compresión, y al recuperarlo se aplica la técnica inversa para descomprimirlo (descompresión).

A veces se utilizan varias técnicas superpuestas para obtener altos grados de compresión. La compresión se reduce al recodificar la información representada internamente.

Hay algoritmos que son capaces de producir una **COMPRESIÓN SIN PÉRDIDA**, es decir, la información original se comprime sin perder nada de la información inicial, pudiéndose recuperar toda la información original.

Pero a veces interesa un mayor grado de compresión, permitiendo un margen de pérdida de calidad. En ese caso se hace uso de la **COMPRESIÓN CON PÉRDIDAS**, pudiéndose recuperar una información aproximada al original.

Definimos el **FACTOR DE COMPRESIÓN** como: $f_c = \frac{\text{capacidad previa}}{\text{capacidad posterior}}$; a la compresión, diciéndose que el archivo tiene una **compresión** $f_c: 1$.

El **PORCENTAJE DE CONVERSIÓN** indica el % que queda de la capacidad original después de la conversión: $P_c = \frac{1}{f_c} \cdot 100\%$.

Algunos algoritmos:

- Sin pérdidas:

- En la descompresión se puede recuperar exactamente el documento original
 - Codificación por secuencias (RLE)
 - Codificación relativa o incremental
 - Codificación dependiente de la frecuencia
 - Codificación con diccionario adaptativo
 - Codificación Lempel-Ziv LZ77

- Con pérdidas

- En la descompresión no se puede recuperar exactamente el documento original
 - Compresión MP3 (sonidos)
 - Compresión GIF (imágenes)
 - Compresión JPEG (imágenes)
 - Compresión MPEG (imágenes)

WUOLAH