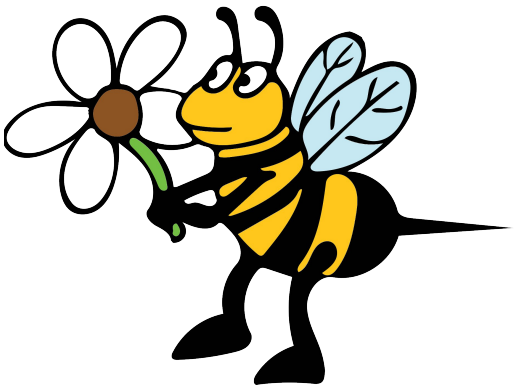
Tema 2. Representación de la información en computadores.



En informática, cuando trabajamos con la información tenemos que resolver dos grandes problemas: cómo la representamos y cómo la almacenamos.
En este tema descubriremos cómo la representamos.

En general, el diseño de un sistema informático resulta sencillo. Se hace uso del sistema menos complejo y más fiable: el sistema binario. Es simple porque solo tenemos dos valores para usar, el cero y el uno, que físicamente se pueden traducir a dos niveles de tensión, de corriente, de magnetización, etcétera.

La información la procesamos y almacenamos usando dos valores de magnitud física, el cero y el uno (bit). Podemos distinguir cinco tipos de representaciones: texto, imágenes, sonidos, valores numéricos e instrucciones.

Cuando trabajamos para representar la información también tenemos que contemplar dos problemas: los errores esporádicos cuando trabajamos con la información (errores) y el tamaño de la información. Tendremos que implementar algoritmos o mecanismos para poder detectar errores en el proceso de transmisión, recuperación o uso de la información y también encontrar formas para almacenar la misma información usando el menor espacio posible.

Representación de textos

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

- **Caracteres alfabéticos** (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, ...).
- **Caracteres especiales** (símbolos ortográficos y matemáticos no incluidos en los grupos anteriores:) (* / + : ; Ñ ñ = ! ? . “ > # SP).

  **SP es el carácter que representa el espacio en blanco*

- **Caracteres geométricos y gráficos** (símbolos con los que se pueden representar marcos de cuadros o de tablas, formas geométricas o iconos elementales: ■ ◈ ┐ ♪ ▌...).
- **Caracteres de control** (representan órdenes de control: carácter para pasar a la línea siguiente (NL), ir al comienzo de una línea (CR), sincronización de una transmisión (SYN)...).

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:

- α (conjunto de caracteres o símbolos que deseamos codificar)
- β (conjunto de combinación binarias de n bits ($\{0,1\}^n$)

Con n bits podemos representar m símbolos distintos, siguiendo la siguiente relación matemática:

$$m = 2^n \text{ tal que } 2^{n-1} < m < 2^n$$

Para calcular cuántos bits son necesarios para representar m símbolos:

$$n \geq log_2(m) = 3{,}32\, log(m) \text{ tal que } n \in \mathbb{N}$$

El número de bits siempre es un natural o entero positivo.

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

- Código ***E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode* o **EBCDIC**, que utiliza **8 bits para representar cada carácter**, pudiendo representar 2^8 símbolos distintos.
- Código ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange* o **ASCII**, usa 8 bits: los **primeros 7 para diferenciar el carácter y el octavo para detectar errores de transmisión o grabación**. Más adelante se ampliará este código, pero cabe destacar que es de los códigos más utilizados por los dispositivos para transmitir datos. Está estandarizado bajo la norma ISO 646. Hay varias versiones derivadas y ampliadas que siguen manteniéndose fiel a la norma del ASCII básico, como el estándar ISO 8895-n, donde la letra n es un número entero y representa el juego de caracteres usados:

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 del sur, maltés y esperanto
Latín-4	ISO 8859-4	Europa del norte

Denominación	Estándar	Área geográfica
Alfabeto latín/cirílico	ISO 8859-5	Lenguaje eslavo
Alfabeto latín/árabe	ISO 8859-6	Lenguajes arábigo
Alfabeto latín/griego	ISO 8859-7	Griego moderno
Alfabeto latín/hebreo	ISO 8859-8	Hebreo y Yiddish
Latín-5	ISO 8859-9	Turco
Latín-6	ISO 8859-10	Nórdico (Sámi, Inuit e islandés)
Alfabeto latín/Thai	ISO 8859-11	Lenguaje Thai
Latín-7	ISO 8859-13	Báltico *Rim*
Latín-8	ISO 8859-14	Céltico
Latín-9 o Latín-0	ISO 8859-15	Latín 1 con ligeras modificaciones (símbolo €)

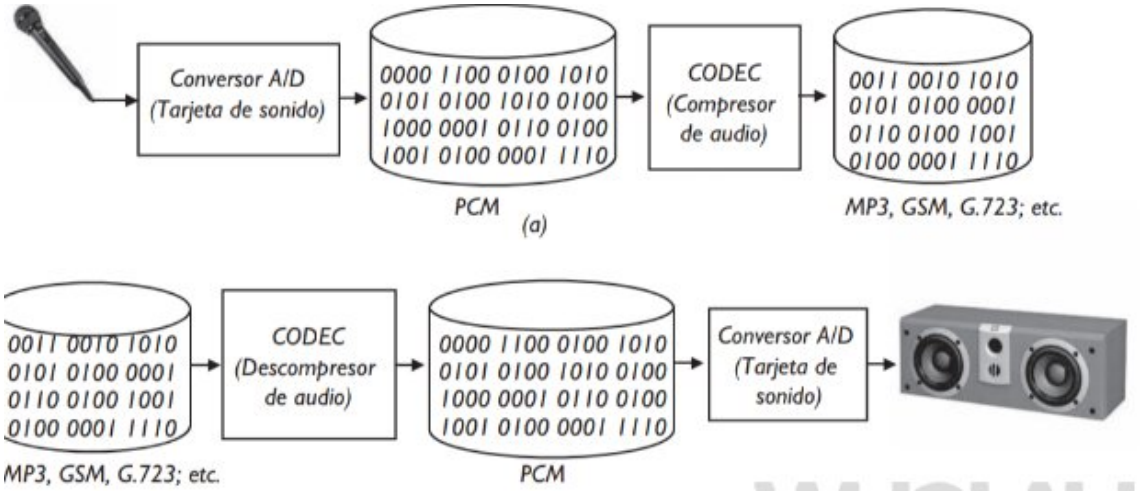
– Código **UNICODE**, que surge ante la necesidad de poder representar más caracteres y dar cabida a un sistema compatible con la vasta variedad mundial. Está reconocido como estándar por la ISO 10646 y promueve los siguientes valores:
  - **Universalidad** (trata de cubrir la mayoría de lenguajes escritos en la actualidad).
  - **Unicidad** (a cada carácter 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.

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	8192
		0080 a 00FF	Suplemento Latín-1 (ISO 8895-1)	128	
		0100 a 017F	Ampliación A del 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	

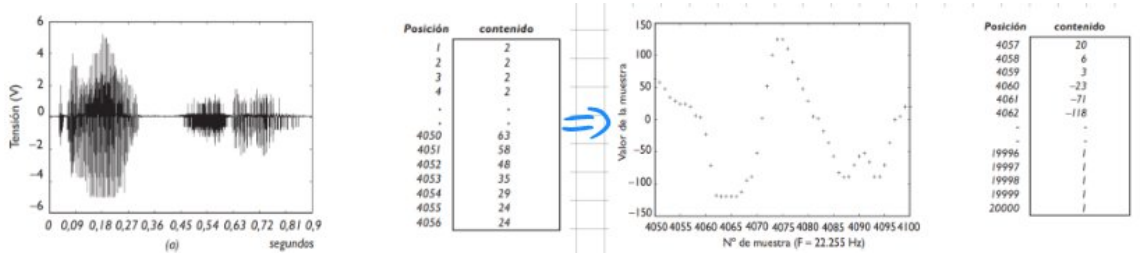
Zona		Códigos	Símbolos codificados	Nº de caracteres
			etc.	
	3FFF	2000 a 3FFF	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos	8192
I	4000 9FFF		Ideogramas	24576
O	A000 DFFF		Pendiente de asignación	16384
R	E000 FFFF		Caracteres locales y propios de los usuarios. Compatibilidad con otros códigos.	8192

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, puede tomar cualquier valor dentro de un determinado intervalo continuo. Para trabajar la señal solemos amplificarla y posteriormente, aplicar un **proceso de digitalización mediante muestreo**.



El muestreo es un proceso por el que 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 $Ts = \frac{1}{Fs}$ segundos se dispone de un valor de la señal.

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 este proceso 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**: que se usa para representar cada muestra, debe ser lo suficiente para recuperar la señal con la calidad requerida.

Para procesar la señal analógica producido por un conversor A/D o para generar la señal a partir de una secuencia de valores con un conversor D/A, se utiliza un circuito denominado códec (**co**dificador/**dec**odificador o **co**mpresor/**desc**ompresor). Hay varios tipos de códec, destacando:

- **P**ulse **C**ode **M**odification (PCM): se transmite o graba un tren de pulsos correspondiente a cada muestra.
- **D**ifferential **P**ulse **C**ode **M**odulation (DPCM): es un códec que disminuye el tamaño de los archivos, almacena la diferencia entre los valores de la muestra y no el valor absoluto.
- **A**daptative **DPCM** (ADPCM): es un algoritmo que predice el valor de muestra siguiente y almacena el error entre el valor predicho y real.
- μ**law**: parecido al ADPCM.
- **MPEG Audio Capa III**, para formatos MP2/3 y AAC, 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 sobrepasan los límites de escucha promedio.
- CÓDECS propietarios: como WMA (Microsoft) o Real Networks (APPLE).

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

$$R = Fs * n$$

donde R es la tasa de datos, Fs la frecuencia de muestreo y n el número de bits por muestreo.

Representación de imágenes

Las imágenes las podemos representar de dos formas:

– **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 (PÍXELES) y atribuir a cada uno de ellos el nivel de gris o el color correspondiente. En 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 en píxeles por línea por nº de columnas de píxeles determina la calidad de la imagen.

  Otro parámetro destacable es el atributo de píxel, que 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, son 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 cómo de opaco o transparente es un píxel.

		Resolución (horizontal x vertical)	Movimiento
Convencionales	Fax (A4)	(100,200,400) x (200,300,400) píxeles/pulgada	Estática
	Foto (8'' x 11'')	128, 400, 1200 píxeles/pulgada	Estática
Televisión	Videoconferencia	176x144 píxeles/imagen	10 a 36 imágenes/s
	TV	720x480 píxeles/imagen	30 imágenes/s
	HDTV	1920x1080 píxeles/imagen	30 imágenes/s
Pantalla computador	VGA	640 x 480 píxeles	
	SVGA	800 x 600 píxeles	
	XGA	1024 x 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 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 texto 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.

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 no consiste en almacenarlos como texto, sino en utilizar tipos de datos aritméticos binarios. 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.

Categoría	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	± [3,4E - 38 a 3,4E38], 0	7
	Coma flotante doble	64	± [1,7E - 308 a 1,7E308], 0	15
	Coma flotante doble largo	80	± [3,4E - 4932 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 digito, 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 posiciones de memoria; si el tamaño es mayor, se troceará convenientemente, y dichos trozos se almacenan en posiciones consecutivas, siguiendo, o bien el **criterio del extremo menor** (almacenamos primero la parte menos significativa del dato) o el **criterio del extremo 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:

Números reales

Enteros sin signos

En este caso todos los bits del dato representan el valor del número expresado en binario natural (sistema de numeración base 2). Entonces, como hay n bits para representar el número, el menor y mayor valor representables son:

$$N(min) = 0 \quad N(max) = 2^n$$

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:

$$N(min) = -\left(2^{n-1}-1\right) \qquad N(max) = 2^{n-1}-1$$

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:

$$N(min) = -\left(2^{n-1}-1\right) \qquad N(max) = 2^{n-1}-1$$

Enteros en complemento a 2

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

Además, el valor mínimo y máximo representables son:

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

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 menor y mayor representables son:

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

Ejemplo

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

Empezamos pasando el número 87 a binario. Para ello hacemos una división sucesiva entre dos:

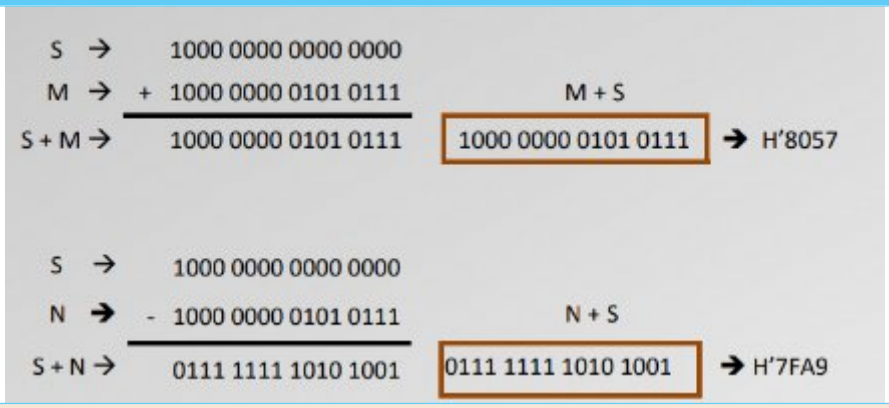
- 87 / 2 = 43 (resto **1**)
- 43 / 2 = 21 (resto **1**)
- 21 / 2 = 10 (resto **1**)
- 10 / 2 = 5 (resto **0**)

- 5 / 2 = 2 (resto **1**)
- 2 / 2 = 1 (resto **0**)
- Cociente final: **1**

Obtenemos que $87)_{10} = 1010111)_2$

Ahora, empezamos a representar el número usando los distintos métodos:

Signo y magnitud			
M	Signo: 0	Magnitud: 000000001010111	H'0057
N	Signo: 1	Magnitud: 000000001010111	H'8067
Complemento a 1 para obtener la N cambiamos los ceros de M por unos y los unos por ceros			
M	Signo: 0	Magnitud: 000000001010111	H'0057
N	Signo: 1	Magnitud: 111111110101000	H'FFA8
Complemento a 2 hacemos el complemento a 1 y le sumamos 1 al complementario			
M	Signo: 0	Magnitud: 000000001010111	H'0057
N	Signo: 1	Magnitud: 111111110101001	H'FFA9
Sesgado			



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 desechar.

La forma más usada para representar los números enteros con signo es la de complemento a dos. La sesgada, 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 Empaquetada*) o bien un único dígito decimal (*BCD Desempaquetado*).

Los datos con signo suelen usar 4 bits para representar el signo. Por ejemplo:: 0000 para el signo positivo y 1001 para el negativo.

Por ejemplo:

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

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.

IEEE754 – Representación de datos de tipo real

Todo número N lo podemos expresar de la forma $N = MB^E$, donde M es la mantisa y E el exponente. El IEEE754 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 almacenan 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 1 bit para el signo, n_e bits para el exponente y n_m bits para el campo de la mantisa, donde el número total de bits utilizados para representar el número real verifica: $n = 1 + n_e + n_m$.

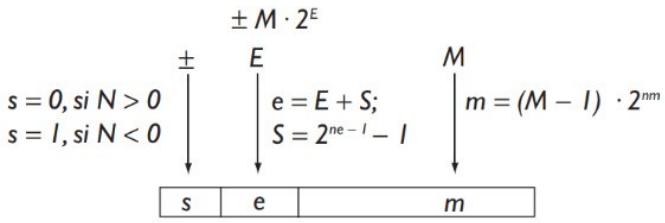


Figura 4.10. *Esquema de la representación IEEE 754.*

  Se utiliza este orden de almacenamiento para que lo más significativo quede de izquierda a derecha, 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 n_e 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 (posición de las unidades). Si el número está ajustado se dice que está *normalizado*, al contrario *desnormalizado*.

  En resumen, reajustar la mantisa y exponente de forma que con $M = [1,m]$ cumpla que $1 \leq M < 2$.

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

- **Casos especiales**:

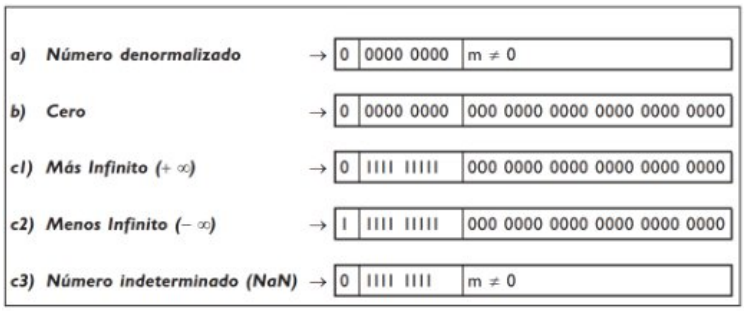


Figura 4.11. *Patrones de bits asociados a situaciones especiales en la notación IEEE 754 (suponemos ne = 8 bits y nm = 23 bits).*

- **Redondeos**: 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) o por exceso. Lo 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 representar la mitad del peso del bit menos significativo retenido. **Para evitar sesgos**, el estándar recomienda el **redondeo al par**, que consiste en redondear por

exceso o defecto siempre de forma que el bit menos significativo del número resultante sea 0, lo que distribuye equitativamente los errores.

Para garantizar la precisión y minimizar tendencias acumulativas, este método considera bits adicionales: el **bit de guarda** (en la posición $-nm+1$) y el de **redondeo** (en la posición $-nm+2$).

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

Simple precisión (32 bits)		
Parámetro	*Valor / Condición*	*Resultado / Significado*
Bits de precisión ($nm+1$)	24	
Exponente máximo (E_{max})	127	
Exponente mínimo (E_{min})	-126	
Sesgo del exponente (S)	127	
Formatos	$e = 255,\ m \neq 0$	NaN
	$e = 255,\ m = 0$	∞
	$0 < e < 255$	número normalizado
	$e = 0,\ m \neq 0$	número desnormalizado
	$e = 0,\ m = 0$	el 0

Doble precisión (64 bits)		
Parámetro	*Valor / Condición*	*Resultado / Significado*
Bits de precisión ($nm+1$)	53	
Exponente máximo (E_{max})	1023	
Exponente mínimo (E_{min})	-1022	
Sesgo del exponente (S)	1023	
Formatos	$e = 2047,\ m \neq 0$	NaN
	$e = 2047,\ m = 0$	∞
	$0 < e < 2047$	número normalizado
	$e = 0,\ m \neq 0$	número desnormalizado
	$e = 0,\ m = 0$	el 0

– **Valores límite**: 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.

Caso / Tipo	**Mantisa (M)**	**Exponente (E)**	**Valor del número (N)**
Número mayor	$M(max) = 2 - 2^{-nm}$	$E(max) = e(max) - S$	$N(max) = M(max)\ 2^{E(max)}$

Número más pequeño (Normalizado)	$M(min) = 1$	$E(nor) = 1 - S$	$N(min, nor) = 2^{E(min)}$
Número más pequeño (Desnormalizado)	$M(min) = 2^{-nm}$	$E(des) = 1 - S$	$N(min, des) = M(min)\ 2^{E(max)}$

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

- Si el número real resultante de una operación cumple $-N(min, des) < N < N(min, des)$ y $N \neq 0$, se dice que el resultado produce un agotamiento (*underflow*).

  Se llama zonas de agotamiento gradual a cero los intervalos: $N(min, des) < N < N(min, nor)$ y $-N(min, nor) < N < -N(min, des)$.

- Si $N < -N(max)$ o $N > N(max)$ con $N \neq 0$, decimos que el resultado de la operación está en una zona de desbordamiento (*overflow*).

Consideramos los siguientes aspectos sobre la precisión (aunque no limitado a esta):

- 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 se necesitan al menos n bits, siendo n el menor natural que verifica:

$$n \geq log_2(m) = 3{,}32\, log(m) \text{ con } n \in \mathbb{N}$$

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

Definimos la **eficiencia del código (τ)** como:

$$\tau = \frac{n^o\ símbolos\ representados}{n^o\ símbolos\ que\ se\ pueden\ representar} = \frac{n}{2^n}$$

El valor resultante debe de cumplir que $0 \leq \tau \leq 1$. Cuanto más eficiente el código usado, τ se aproxima más a 1.

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

$$R_d = (1 - \tau) * 100$$

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 ó 1) de forma tal que el número total de unos del código resulte par.

Mensaje o código inicial	Mensaje o código con bit de paridad (criterio par)
100 0001	**0**100 0001
010 1111	**1**010 1111
110 1000	**1**110 1000
111 0111	**0**111 0111

Subrayado en amarillo el bit de paridad.

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

Mensaje o código inicial	Mensaje o código con bit de paridad (criterio impar)
100 0001	**1**100 0001
010 1111	**0**010 1111
110 1000	**0**110 1000
111 0111	**1**111 0111

Subrayado en amarillo el bit de paridad.

El bit de paridad se introduce antes de transmitir o grabar la información. Al leer la información se detectaría el error y, 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 el **CRC-CITT** o *Comité Consultatif International Téléphonique et Télegraphique*. Se demuestra que utilizando el polinomio generador en una codificación polinómica (CRC):

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

Se detectan:

- Todos los errores de 1 bit individual.
- Todos los errores que me 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.
- El 99.998% de errores de ráfagas de 18 o más bits.

También existen **CÓDIGOS CORRECTORES DE ERRORES** con los 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

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:

$$f_c = \frac{capacidad\ previa}{capacidad\ posterior}$$

Decimos que un archivo comprimido tiene una compresión f_c:1.

El **porcentaje de compresión** indica el porcentaje que queda de la capacidad original después de la compresión:

$$P_c = \frac{1}{f_c} * 100\%$$

Algunos algoritmos son:

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