Tema 2. Representación de Datos

OBJETIVO

Como se estudió en el tema 1, un ordenador es una máquina que procesa datos. Pero antes de que podamos hablar sobre el procesamiento de datos, necesita comprender la naturaleza de los mismos. En este tema se analizan los diferentes tipos de datos y cómo se representan dentro de un ordenador.

Bibliografía

Forouzan, A. Introducción a la Ciencia de la Computación. Cap. 2 y 3. Ed. Thompson, 2003

CONTENIDOS

---

1. Tipos de datos

En la actualidad los datos se presentan de diferentes maneras, por ejemplo números, texto, imágenes, audio y video (figura 2.1). La gente necesita procesar todos estos tipos de datos.

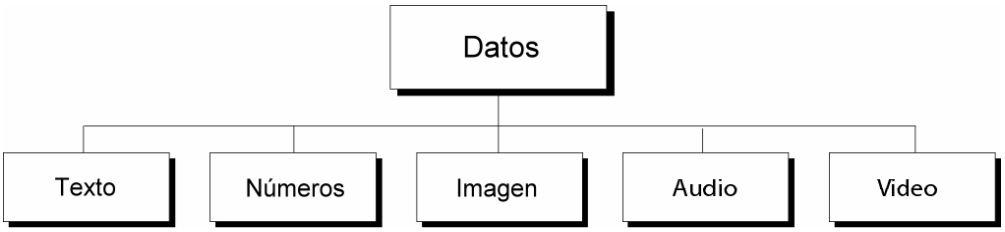


Figura 2.1. Diferentes tipos de datos.

- Un programa de ingeniería utiliza un ordenador principalmente para procesar números: hacer aritmética, resolver ecuaciones algebraicas o trigonométricas, encontrar las raíces de una ecuación diferencial, y así por el estilo.

- Un programa de procesamiento de palabras, por otra parte, utiliza un ordenador más que nada para procesar texto: justificarlo, moverlo, eliminarlo, etcétera.
- Un programa de procesamiento de imágenes usa un ordenador para manipular imágenes: crearlas, reducirlas, ampliarlas, rotarlas, etcétera.
- Una ordenador también puede manejar datos de audio. Se puede reproducir música en una ordenador e introducir voz como datos.
- Finalmente, una ordenador puede usarse no sólo para mostrar películas, sino también para crear los efectos especiales que se ven en ellas.

La industria de la computación usa el término *multimedia* para definir información que contiene números, texto, imágenes, audio y video.

2. Datos dentro del Ordenador

La pregunta es: ¿Cómo se manejan todos estos tipos de datos? ¿Se necesitan otras ordenadores para procesar los distintos tipos de datos? Es decir, ¿se tiene una categoría de ordenadores que procesan sólo números? ¿Hay una categoría de ordenadores que procesan sólo texto?

Esta solución de diferentes ordenadores para procesar distintos tipos de datos no es económica ni práctica porque los datos por lo general son una mezcla de tipos. Por ejemplo, aunque un banco procesa principalmente números, también necesita almacenar, como texto, los nombres de sus clientes. Como otro ejemplo, una imagen con frecuencia es una mezcla de gráficos y texto.

La solución más eficaz es usar una representación uniforme de los datos. Todo tipo de datos que entran del exterior a una ordenador se transforman en esta representación uniforme cuando se almacenan en una ordenador y se vuelven a transformar en su representación original cuando salen de el ordenador. Este formato universal se llama *patrón de bits.*

2.1. Bit

Antes de continuar con el análisis de los patrones de bits, se debe definir un bit. Un bit (bi*nary digit:* dígito binario) es la unidad más pequeña de datos que puede almacenarse en una ordenador; puede ser ya sea 0 ó 1. Un bit representa el estado de un dispositivo que puede tomar uno de dos estados. Por ejemplo, un interruptor puede estar ya sea apagado o encendido. La convención es representar el estado de encendido como 1 y el estado de apagado como 0. Un interruptor electrónico puede representar un bit. En otras palabras, un interruptor puede almacenar un bit de información. Actualmente los ordenadores utilizan varios dispositivos binarios de dos estados para almacenar datos.

2.2. Patrón de bits

Un solo bit no puede resolver el problema de la representación de datos; si cada pieza de datos pudiera representarse por un 1 o un 0, entonces sólo se necesitaría un bit. Sin embargo, es necesario almacenar números más grandes, almacenar texto, gráficos y otros tipos de datos.

Para representar diferentes tipos de datos se utiliza un **patrón de bits**, una secuencia o, como a veces se le llama, una cadena de bits. La figura 2.2 muestra un patrón de bits formado por 16 bits: es una combinación de ceros (0) y unos (1). Esto significa que si usted quiere almacenar un patrón de bits formado por 16 bits, necesita 16 interruptores electrónicos. Si quiere almacenar 1000 patrones de bits, cada uno de 16 bits, necesita 16 000 bits y así sucesivamente.

1 0 0 0 1 0 1 0 1 0 1 1 1 1 1

Figura 2.2. Patrón de bits.

Ahora la pregunta es: ¿Cómo sabe la memoria del ordenador qué tipo de datos representa el patrón de bits? No lo sabe. La memoria del ordenador sólo almacena los datos como patrones de bits. Es responsabilidad de los dispositivos de entrada/salida o de los programas interpretar un patrón de bits como un número, texto o algún otro tipo de datos. En otras palabras, los datos se codifican cuando entran al ordenador y se decodifican cuando se presentan al usuario (figura 2.3).

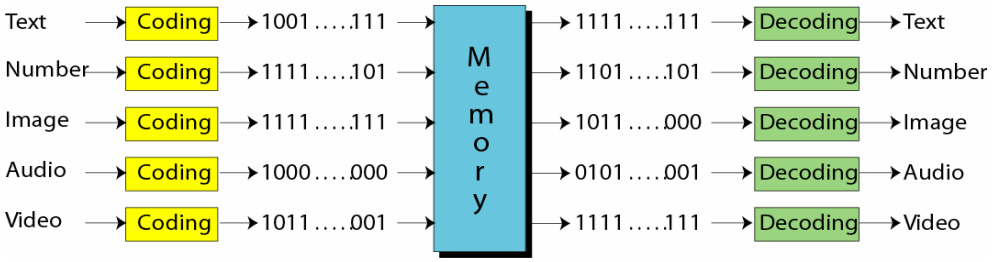


Figura 2.3. Ejemplos de patrones de bits.

2.3. Byte

Por tradición, un patrón de bits con una longitud de 8 bits se llama **byte**. Este término también se ha utilizado para medir el tamaño de la memoria o de otros dispositivos de almacenamiento. Por ejemplo, se dice que la memoria de un ordenador que puede almacenar 8 millones de bits de información es una memoria de 1 millón de bytes.

3. Representación de Datos

Ahora podemos explicar cómo pueden representarse diferentes tipos de datos usando patrones de bits.

3.1. Texto

Una pieza de **texto** en cualquier idioma es una secuencia de símbolos usados para representar una idea en ese idioma. Por ejemplo, el idioma inglés utiliza 26 símbolos (A, B, C, ..., Z) para representar las letras mayúsculas, 26 símbolos (a, b, c,..., z) para representar las letras minúsculas, 9 símbolos (0, 1, 2,..., 9) para los caracteres numéricos (no números; la diferencia se verá más adelante) y símbolos (., ?, :, :..., !) para representar la puntuación. Otros símbolos como el espacio en blanco, la línea nueva y el tabulador se usan para alineación de texto y legibilidad.

Se puede representar cada símbolo con un patrón de bits. Dicho de otra forma, texto como la palabra "BYTE", formada por cuatro símbolos, puede representarse como 4 patrones de bits, en los que cada patrón define un solo símbolo (figura 2.4).

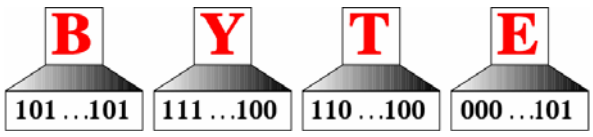


Figura 2.4. Representación de símbolos usando patrones de bits.

La pregunta es: ¿Cuántos bits se necesitan en un patrón de bits para representar un símbolo en un idioma? Depende de cuántos símbolos haya en la secuencia. Por ejemplo, si se crea un idioma imaginario que utilice sólo las letras mayúsculas del idioma inglés, sólo necesita 26 símbolos. Un patrón de bits en

este idioma requiere representar al menos 26 símbolos. Para otro idioma, como el chino, pueden necesitarse muchos símbolos más. La longitud del patrón de bits que representa un símbolo en un idioma depende del número de símbolos usados en ese idioma. Más símbolos significan un patrón de bits más grande.

Aunque la longitud del patrón de bits depende del número de símbolos, la relación no es lineal; es logarítmica. Si se requieren dos símbolos, la longitud es 1 bit (el $\log_2 2$ es 1). Si se necesitan cuatro símbolos, la longitud es 2 bits ($\log_2 4$ es 2). La tabla 2.1 muestra esta relación, la cual es fácilmente perceptible. Un patrón de bits de 2 bits puede tomar cuatro formas diferentes: 00, 01, 10 y 11; cada una de las cuales representa un símbolo. Del mismo modo, un patrón de tres bits puede tomar ocho formas diferentes: 000, 001, 010, 011, 100, 101, 110 y 111.

Número de símbolos	Longitud del patrón de bits
2	1
4	2
8	3
16	4
…	…
128	7
256	8
…	…
65 536	16

Tabla 2.1. Número de símbolos y longitud de un patrón de bits.

3.2. Códigos

Se han diseñado diferentes secuencias de patrones de bits para representar símbolos de texto. A cada secuencia se le conoce como **código** y al proceso de representar los símbolos se le llama codificación. En esta sección explicamos los códigos comunes.

ASCII

El Instituto Nacional Norteamericano de Estándares (ANSI: *American Nacional Standards Institute)* desarrolló un código llamado **Código norteamericano de estándares para intercambio de información (ASCII:** *American Standard Code for Information Interchange).* Este código utiliza siete bits para cada símbolo. Esto significa que 128 (27) símbolos distintos pueden definirse mediante este código. Los patrones de bits completos para el código ASCII están en el apéndice A. La figura 2.5 muestra cómo se representa la palabra "BYTE" en código ASCII.

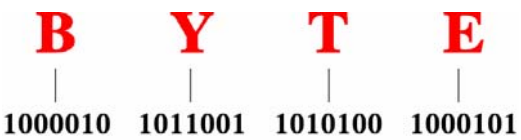


Figura 2.5. Representación de la palabra "BYTE" en código ASCII.

La lista siguiente destaca algunas de las características de este código:

- ASCII utiliza un patrón de siete bits que varía de 0000000 a 1111111.
- El primer patrón (0000000) representa el carácter nulo (la ausencia de carácter).

- El último patrón (1111111) representa el carácter de eliminación.
- Hay 31 caracteres de control (no imprimibles).
- Los caracteres numéricos (0 a 9) se codifican antes que las letras.
- Hay varios caracteres de impresión especiales.
- Las letras mayúsculas (A ... Z) están antes que las letras minúsculas (a ... z).
- Los caracteres en mayúsculas y en minúsculas se distinguen sólo por un bit. Por ejemplo, el patrón para A es 1000001; el patrón para a es 1100001. La única diferencia es el sexto bit a partir de la derecha.
- Hay seis caracteres especiales entre las letras mayúsculas y minúsculas.

ASCII extendido

Para hacer que el tamaño de cada patrón sea de 1 byte (8 bits), a los patrones de bits ASCII se les aumenta un 0 más a la izquierda. Ahora cada patrón puede caber fácilmente en un byte de memoria. En otras palabras, en **ASCII extendido** el primer patrón es 00000000 y el último es 01111111.

Algunos fabricantes han decidido usar el bit de más para crear un sistema de 128 símbolos adicional. Sin embargo, este intento no ha tenido éxito debido a la secuencia no estándar creada por cada fabricante.

EBCDIC

A principios de la era de los ordenadores, IBM desarrolló un código llamado **Código extendido de intercambio decimal codificado en binario (EBCDIC:** *Extended Binary Coded Decimal Interchange Code).* Este código utiliza patrones de ocho bits, de manera que puede representar hasta 256 símbolos. Sin embargo, este código no se utiliza más que en ordenadores mainframe de IBM.

Unicode

Ninguno de los códigos anteriores representa símbolos que pertenecen a idiomas distintos al inglés. Por eso, se requiere un código con mucha más capacidad. Una coalición de fabricantes de hardware y software ha diseñado un código llamado **Unicode** que utiliza 16 bits y puede representar hasta 65536 (2^{16}) símbolos. Diferentes secciones del código se asignan a los símbolos de distintos idiomas en el mundo. Algunas partes del código se usan para símbolos gráficos y especiales. El lenguaje Java™ utiliza este código para representar caracteres. Microsoft Windows usa una variación de los primeros 256 caracteres. En el apéndice B hay un pequeño conjunto de símbolos Unicode.

ISO

La **Organización Internacional para la Estandarización** *(Internacional Standard Organization),* conocida como ISO, ha diseñado un código que utiliza patrones de 32 bits. Este código representa hasta 4,294,967,296 (2^{32}) símbolos, definitivamente lo suficiente para representar cualquier símbolo en el mundo actual.

3.3. Números

En una ordenador, los números se representan usando el **sistema binario.** En este sistema, un patrón de bits (una secuencia de ceros y unos) representa un número. Sin embargo, un código como el ASCII no se usa para representar datos. La razón para ello y un análisis de la representación de números se presentan en la sección "Representación de Números".

3.4. Imágenes

Hoy día las **imágenes** se representan en un ordenador mediante uno de dos métodos: **gráficos de mapa de bits** o **gráficos de vectores** (figura 2.6). Esta no es una división tajante, ya que las imágenes vectoriales suelen admitir la incrustación de imágenes de mapa de bits en su interior y los programas especializados en dibujo vectorial (Illustrator, Freehand y CorelDraw!) cada vez tienen más cualidades de los programas de tratamiento de imágenes de mapa de bits (Photoshop, o Corel Photopaint). Lo contrario también es cierto.

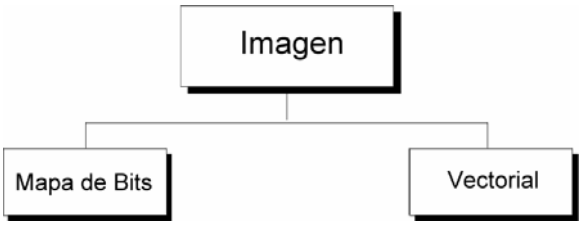


Figura 2.6. Métodos de representación de imágenes.

En este método, una imagen se divide en una matriz de **pixeles** *(picture elements:* **elementos de imagen),** donde cada pixel es un pequeño punto. El tamaño del pixel depende de lo que se conoce como *resolución.* Por ejemplo, una imagen puede dividirse en 1000 pixeles o 10 000 pixeles. En el segundo caso, aunque hay una mejor representación de la imagen (mejor resolución), se necesita más memoria para almacenarla.

Después de dividir una imagen en pixeles, a cada pixel se asigna un patrón de bits. El tamaño y el valor del patrón dependen de la imagen. Para una imagen formada sólo por puntos blancos y negros (por ejemplo, un tablero de ajedrez), un patrón de un bit es suficiente para representar un pixel. Un patrón de 0 representa un pixel negro y uno de 1 representa un pixel blanco. Luego los patrones se registran uno tras otro y se almacenan en el ordenador. La figura 2.7 muestra una imagen de este tipo y su representación.

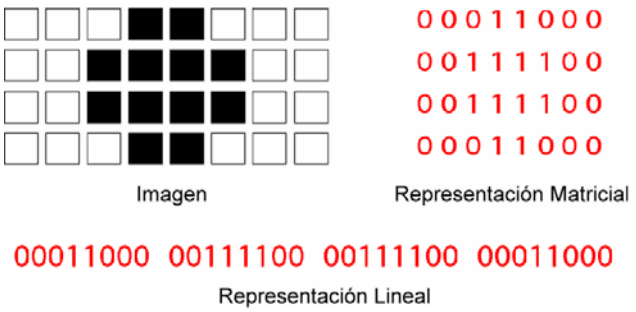


Figura 2.7. Método de gráficos de mapa de bits de una imagen blanca y negra.

Si una imagen no se forma de pixeles puramente blancos y pixeles puramente negros, se puede aumentar el tamaño del patrón de bits para representar escalas de grises. Por ejemplo, para mostrar cuatro niveles de la escala de grises, se puede usar un patrón de dos bits. Un pixel negro puede representarse por 00, un gris oscuro por 01, un pixel gris claro por 10 y un pixel blanco por 11.

De este modo, podemos hablar de una imagen que tenga 200 × 100 píxeles sin saber que tamaño real y físico tiene. Lo único que sabemos es que la hemos dividido en 20.000 celdillas.

Sin embargo, cuando le asignemos a esa imagen una resolución, entonces sí sabremos qué tamaño tiene esa imagen. Por ejemplo, si decimos que tiene 100 píxeles por pulgada, querrá decir que cada 2,54 cm. (pues eso es lo que mide una pulgada), habrá 100 celdillas, con lo que cada píxel equivaldrá a 2,54 mm.

Si dijéramos que esa imagen tiene una resolución de 1 píxel por pulgada, lo que sabríamos es que ahora esa celdilla tomaría el valor de 2,54 cm.

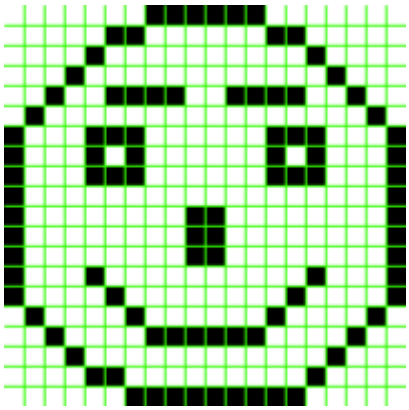
Todo ello significa, insisto, que el píxel es sólo una unidad de división sin un tamaño real concreto. Sólo cuando asignamos una resolución a la imagen de la que hablamos estamos dándole un tamaño concreto al píxel.

Como ya hemos visto en otro sitio, hay imágenes de mayor resolución e imágenes de más baja resolución. A mayor resolución, mayor nitidez del dibujo y mejor se reflejan los detalles. Sin embargo, hay que tener presente que cualquier resolución que supere la que el dispositivo de salida (pantalla, impresora, etc...) es capaz de representar no hace más que sobrecargar el sistema y ralentizar el trabajo.

3.5. Tipos de imágenes de mapa de bits

Una forma muy importante de clasificar las imágenes de mapa de bits es según la cantidad y tipo de información que se asigne a cada píxel (aunque en algunos aspectos es una clasificación un poco "mixta" y puede parecer un poco desordenada, se hace así por claridad explicativa):

1. Imágenes de 1 bit por píxel: en este tipo de imágenes cada celdilla (píxel) sólo puede tener uno de dos valores: Uno o cero. Como basta 1 bit para definir esa alternativa, se les llama "imágenes de 1 bit" (también se les llama "imágenes de mapa de bits, de alto contraste, o imágenes de línea").

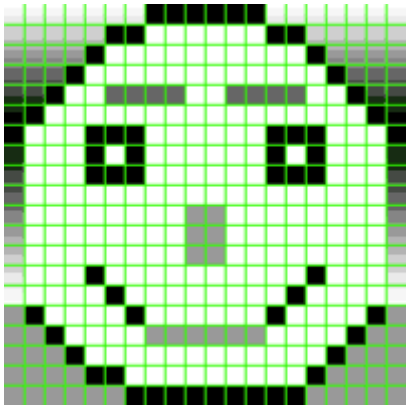


Una imagen de 20×20 píxeles (400). Podría medir 2 cm. o un campo de futbol. Los píxeles son sólo una división de la información que contiene.

Una imagen de 200×200 píxeles en este modo. La información es muy escasa para reproducir bien este tipo de imagen.

2. Imágenes de escala de grises (8 bits por píxel)

Cada píxel puede tener 256 valores diferentes (las 256 posibilidades combinatorias de un byte u octeto). Este es el modo de las imágenes digitales de blanco y negro "normales". Aunque te pueda parecer increíble, en ellas sólo se distinguen hasta 256 tonos diferentes de gris (y no suelen aparecer todos a la vez, por cierto).



Una imagen de 20×20 píxeles (400) con 1 byte (8 bits) por píxel. Pesará 400 × 8 bits, es decir: 3.200 bits.

La imagen de 200×200 píxeles en escala de grises. La información es suficiente para reproducir fotografías en blanco y negro.

3.6. Imágenes RGB o Lab (24 bits por píxel)

Para representar imágenes a color, cada pixel coloreado se descompone en tres colores primarios: rojo, verde y azul (RGB). Luego se mide la intensidad de cada color y se le asigna un patrón de bits (por lo general ocho bits). En otras palabras, cada pixel tiene tres patrones de bits: uno para representar la intensidad del color rojo, uno para la intensidad del color verde y uno para la intensidad del color azul. Por ejemplo, la figura 2.8 muestra cuatro patrones de bits para algunos pixeles en una imagen a color.

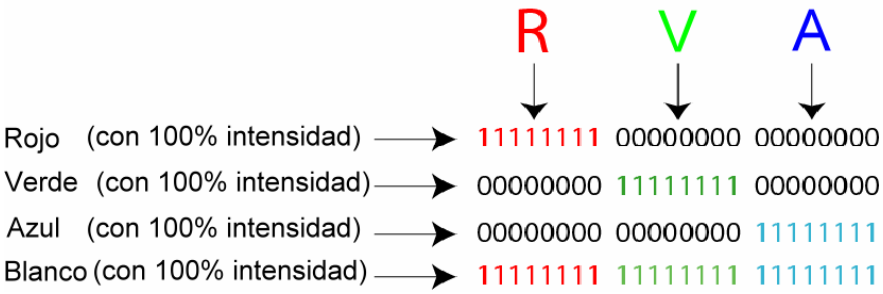
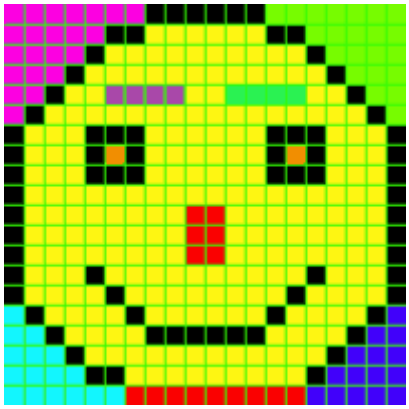


Figura 2.8. Representación de pixeles de color.

Si tomamos un píxel y le asignamos tres bytes, dispondremos de 24 bits en tres grupos de ocho, podemos "colorearlo" siguiendo el sistema de color de los monitores de televisión, que se basan en tres "canales" de luz de color (Rojo, Azul y Verde). De este modo podemos distinguir hasta 16.777.216 millones de tonos de color (256 Rojo× 256 Azul × 256 Verde). En realidad, lo que estamos hciendo es superponer tres canales de luz, uno rojo, otro verde y otro azul, cada uno con 256 posibilidades de tono.

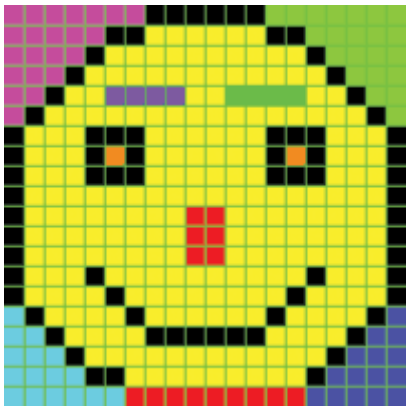


Una imagen de 20×20 píxeles (400) con 3 bytes (8 bits) por cada píxel. Pesará 400 × 8 × 3 bits, es decir: 9.600 bits.

La imagen de 200×200 píxeles en modo RGB, el tipo de color de las televisiones.

Imágenes CMYK (32 bits por píxel)

Si a cada píxel le asignamos 4 bytes, podríamos representar (teóricamente), los valores CMYK propios de la cuatricromía profesional (1 byte para el cian, otro para el magenta, otro para el amarillo y un cuarto para el negro).



Una imagen de 20×20 píxeles (400) con 4 bytes (8 bits) por cada píxel. Pesará 400 × 8 × 4 bits, es decir: 12.800 bits.

La imagen de 200×200 píxeles en modo CMYK. Lo que ves es una representacion RGB, no hay monitores CMYK.

Teóricamente, porque la representación del color que la pantalla de un ordenador puede hacer es mediante imágenes RGB, ya que ese es el modo de reproducir el color de los monitores.

Imágenes en color de 8 bits o menos

Es lo que se llama color indexado. Lo que se hace es que se crea una tabla o índice de 256 colores y a cada una de los posibles valores de un píxel se le asigna uno de ellos. Si la tabla la construimos con menos posibilidades (16, por ejemplo), esa imagen no será un color indexado de 256 tonos sino uno de 16. Para más información sobre este tipo de imágenes, sigue leyendo imágenes de escala de grises .

Otros modos especiales de imagen

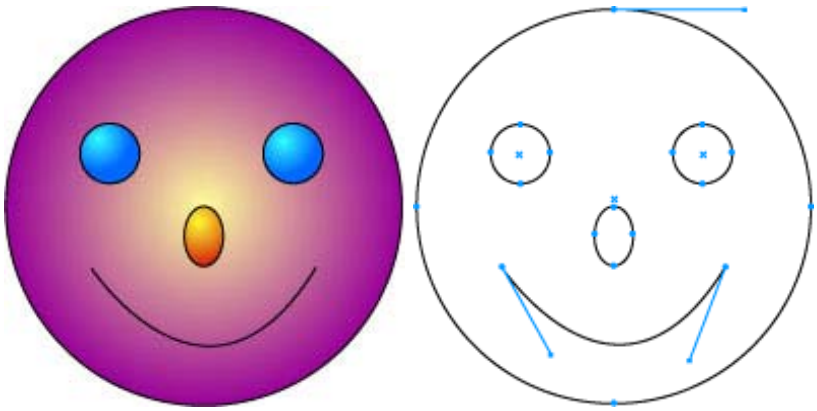
Son ampliaciones de los modos vistos anteriormente. Puede ocurrir que tengan más de cuatro canales (debido a que tengan canales de máscara (para efectuar operaciones especiales) o canales de tinta planas (ficheros multicanal). También puede ser que aunque tengan sólo dos, tres o cuatro canales, éstos

tengan asignado un valor cromático especial (archivos multicanal). Otra posibilidad es que para representar tonos asignen más de un byte por píxel, es decir, que sean imágenes que son capaces de representar más de 256 valores por tonos. Imágenes de este tipo se dan, por ejemplo, como resultado de escanear en modos con "más profundidad de bits".

3.7. Las imágenes vectoriales

El problema con el método de los gráficos de mapa de bits es que los patrones de bits exactos para representar una imagen particular deben guardarse en una ordenador. Posteriormente, si se desea cambiar el tamaño de la imagen debe cambiar el tamaño de los pixeles, lo cual crea una apariencia difusa y granulada. Una forma muy distinta de formar una imagen es la de hacerlo mediante operaciones matemáticas. Es decir, en vez de trazar una retícula con miles o millones de puntos para trazar una línea, le decimos a la máquina unas coordenadas **`x1`** e **`y1`** le decimos que trace una línea hasta otras coordenadas **`x2`** e **`y2`** .

Así podemos dibujar círculos, cuadrados, triángulos y miles de formas. Sin entrar en detalles, diremos que esa es la base de los llamados dibujos vectoriales. Los programas de dibujo vectorial los suelen representar de dos maneras: Representación completa (es decir, tal cual se imprimirán) y como líneas (es decir, sólo el esqueleto de las formas básicas, mucho menos pesado para el ordenador).



Un dibujo vectorial en modo de representación completa y visto como líneas básicas, con sus elementos de dibujo.

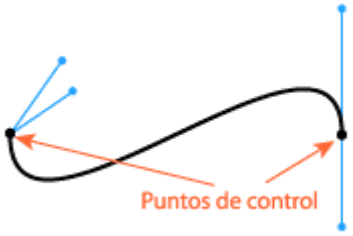
Los trazados (líneas curvas o rectas propias de un dibujo vectorial) se pueden modificar fácilmente, se almacenan en muy poco espacio y además son independientes de la resolución, ya que no dependen de una retícula dada y basándose en que cualquier operación geométrica es multiplicable o divisible en su conjunto sin que eso afecte al aspecto del resultado, sino sólo a su tamaño final.

Las imágenes vectoriales de dos dimensiones suelen tener varias partes. Sólo el contorno y el relleno serán visibles al imprimir. Lo demás son instrumentos de trabajo. La base de estas operaciones son las llamadas "Curvas Bezier":

- Trazado

(Path). Es la línea en sí, puede ser curva o recta. Puede ser simple (una sóla línea o complejo (está compuesto por sucesivas líneas con varios puntos de control).

- Puntos de control o anclaje

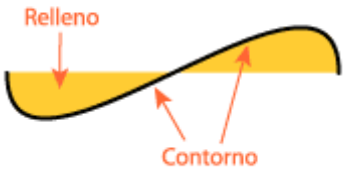


(Control points y *anchor points).* Son los extremos de una línea o los diversos puntos en los que un trazado complejo cambia de forma.

- Manejadores o tiradores

(Handlers). Son unas pequeñas líneas rectas que salen a mayor o menor distancia de los puntos de control y terminan en una especie de bolita. Tirando de esta bolita se puede modificar la forma del trazado en esa zona.

- Contorno y relleno



El contorno *(Outline / path).* Es la línea de borde de un trazado. Puede ser un color continuo o una sucesión de puntos, rayas o un motivo repetitivo.

El relleno*(Fill).* Es lo que llena un trazado por sus partes más cerradas puede ser un color o un motivo repetitivo *(pattern).*

Las imágenes vectoriales tienen el inconveniente de tener dificultades en tratar algunas cosas de forma "natural" (sombras, luces, etc...) y cuando son muy grandes o muy complejas pueden volverse extremadamente difíciles de manejar para la capacidad de un ordenador hasta el extremo de que el RIP PostScript no sea capaces de procesarlas.

En artes gráficas, el formato "natural" de las imágenes vectoriales es el de ficheros EPS . Cualquier otro (Corel o Freehand) es sólo un formato de fichero de trabajo interno de diseño gráfico y no debe tener otro uso.

3.8. Audio

El audio es una representación de sonido o música. Aunque no hay un estándar para almacenar el sonido o la música, la idea es convertir el audio a datos **digitales** y usar patrones de bits. El audio por naturaleza es información **análoga.** Es continuo (análogo), no discreto (digital). La figura 2.9 muestra los pasos a seguir para cambiar los datos de audio a patrones de bits.

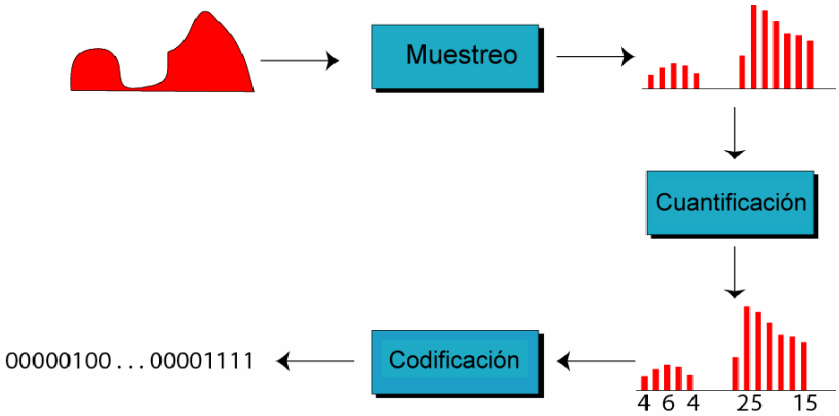


Figura 2.9. Representación de audio.

Estos pasos son los siguientes:

1. La señal análoga se muestrea. El **muestreo** significa medir el valor de la señal a intervalos iguales.
2. Las muestras se cuantífican. La **cuantificación** significa asignar un valor (de un conjunto) a una muestra. Por ejemplo, si el valor de una muestra es 29.2 y el conjunto es el conjunto de enteros entre 0 y 63, se asigna un valor de 29 a la muestra.
3. Los valores cuantificados se cambian a patrones binarios. Por ejemplo, el número 25 se cambia al patrón binario 00011001 (consulte el capítulo 3 para la transformación de números en patrones).

4. Los patrones binarios se almacenan.

3.9. Video

El video es una representación de imágenes (llamadas cuadros o *frames)* en el tiempo. Una película es una serie de cuadros desplegados uno tras otro para crear la ilusión de movimiento. Así que si se sabe cómo almacenar una imagen dentro de una ordenador, también se sabe cómo almacenar un video; cada imagen o cuadro cambia a una serie de patrones de bits y se almacena. La combinación de las imágenes representa el video. Obsérvese que el video actual se comprime normalmente.

4. Notación Hexadecimal

El patrón de bits se diseñó para representar datos cuando éstos se almacenan dentro de un ordenador. Sin embargo, para la gente es difícil manipular los patrones de bits. Escribir una serie de números 0 y 1 es tedioso y propenso al error. La notación hexadecimal ayuda.

La **notación hexadecimal** se basa en 16 *(hexadec* es la palabra griega para 16). Esto significa que hay 16 símbolos (dígitos hexadecimales): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. La importancia de la notación hexadecimal se hace evidente cuando se convierte un patrón de bits a notación hexadecimal.

Cada dígito hexadecimal puede representar cuatro bits y cuatro bits pueden representarse mediante un dígito hexadecimal. La tabla 2.2 muestra la relación entre un patrón de bits y un dígito hexadecimal.

Un patrón de 4 bits puede representarse mediante un dígito hexadecimal, y vice-versa.

Patrón de bits	Dígito hexadecimal	Patrón de bits	Dígito hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Tabla 2.2. Dígitos hexadecimales

CONVERSIÓN

La conversión de un patrón de bits a notación hexadecimal se realiza por medio de la organización del patrón en grupos de cuatro y luego hallar el valor hexadecimal para cada grupo de cuatro bits. Para una conversión de hexadecimal a patrón de bits se convierte cada dígito hexadecimal a su equivalente de cuatro bits (figura 2.10).

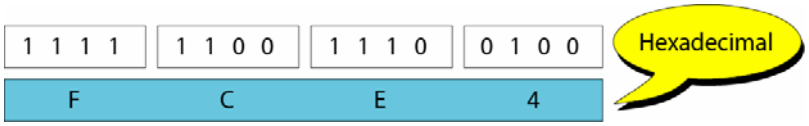


Figura 2.10. Transformación de binario a hexadecimal y de hexadecimal a binario.

Observe que la notación hexadecimal se escribe en dos formatos. En el primer formato, se añade una x minúscula (o mayúscula) antes de los dígitos para mostrar que la representación está en hexadecimal. Por ejemplo, xA34 representa un valor hexadecimal en esta convención. En otro formato, usted indica la base del número (16) como el subíndice después de cada notación. Por ejemplo, $A34_{16}$ muestra el mismo valor en la segunda convención. En este libro se usan ambas convenciones.

Ejemplo 1

Determine el hexadecimal equivalente del patrón de bits 110011100010.

Solución

Cada grupo de cuatro bits se traduce a un dígito hexadecimal. El equivalente es xCE2.

Ejemplo 2

Determine el hexadecimal equivalente del patrón de bits 0011100010.

Solución

El patrón de bits se divide en grupos de cuatro bits (a partir de la derecha). En este caso, se añaden dos 0 más a la izquierda para hacer el número total de bits divisible entre cuatro. Así que usted tiene 000011100010, lo cual se traduce a x0E2.

Ejemplo 3

¿Cuál es el patrón de bits para x24C?

Solución

Cada dígito hexadecimal se escribe como su patrón de bits equivalente y se obtiene 001001001100.

5. Notación Octal

Otra notación usada para agrupar patrones de bits es la notación octal. La **notación octal** se basa en 8 *(oct* es la palabra griega para ocho). Esto significa que existen ocho símbolos (dígitos octales): 0, 1, 2, 3, 4, 5, 6, 7. La importancia de la notación octal se hace evidente a medida que se aprende a convertir un patrón de bits en notación octal.

Cada dígito octal representa tres bits y tres bits pueden representarse mediante un dígito octal. La tabla 2.1 muestra la relación entre un patrón de bits y un dígito octal.

Un patrón de tres bits puede representarse por medio de un dígito octal y viceversa.

Patrón de bits	Dígito octal	Patrón de bits	Dígito octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Tabla 2.1. Dígitos octales

Conversión

La conversión de un patrón de bits a notación octal se realiza mediante la organización del patrón en grupos de tres y la determinación del valor octal de cada grupo de tres bits. Para la conversión de octal a patrón de bits, se convierte cada dígito octal a su equivalente de tres bits (figura 2.11).

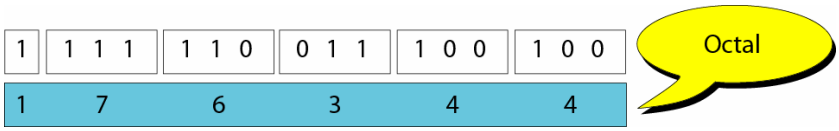


Figura 2.11**.** Transformación de binario a octal y de octal a binario

Obsérvese que la notación octal también se escribe en dos formatos. En el primer formato, se añade 0 (cero) antes de los dígitos para mostrar que la representación está en notación octal (a veces se utiliza una o minúscula). Por ejemplo, 0634 representa un valor octal en esta convención. En el otro formato, usted indica la base del número (8) como el subíndice después de la notación. Por ejemplo, 634_8 muestra el mismo valor en la segunda convención.

Ejemplo 4

Muestre el equivalente octal del patrón de bits 101110010.

Solución

Cada grupo de tres bits se traduce a un dígito octal. El equivalente es 0562, o562 o 562_8.

Ejemplo 5

Muestre el equivalente octal del patrón de bits 1100010.

Solución

El patrón de bits se divide en grupos de tres bits (a partir de la derecha). En este caso, se añaden dos 0 más a la izquierda para hacer el número total de bits divisible entre 3. Así que usted tiene 001100010, lo cual se traduce a 0142, ol42 o 142_8.

Ejemplo 6

¿Cuál es el patrón de bits para 24_8?

Solución

Cada dígito octal se escribe como su patrón de bits equivalente para obtener 010100.

6. Representación de números

En las secciones previas mostramos cómo el texto, el audio, las imágenes y el video pueden representarse en un ordenador mediante patrones de bits. Pospusimos el análisis de la representación de los números porque ésta es muy diferente de la representación de los datos no numéricos. Algunas de las razones de esta diferencia son las siguientes:

- Un código de caracteres como el ASCII no es eficiente para representar números. ASCII puede representar 128 símbolos, pero el sistema decimal necesita sólo 10. (Obsérvese que si se consideran otros símbolos como +, — y el punto decimal, se necesitan aún más símbolos, pero todavía menos que 128.) Por ejemplo, si usted quiere almacenar el número 65535 usando ASCII, necesita cinco

bytes (un byte para cada dígito). Pero si el número se representa como un entero sin signo (usted verá esta representación posteriormente en esta sección), sólo necesita dos bytes.

- Las operaciones con los números (por ejemplo, la suma y la resta) son muy complicadas si los dígitos de un número se representan en un código de caracteres.
- La representación de la precisión de un número (por ejemplo, el número de lugares después del punto decimal) requiere muchos bytes. Por ejemplo, para almacenar 23454.00001 se requieren 11 bytes, pero si el mismo número se representa en un punto flotante (esta representación se verá más adelante en este tema), necesita sólo unos cuantos bytes.

7. Decimal y binario

Dos sistemas de numeración predominan actualmente en el mundo de la computación: decimal y binario. Analizaremos estos dos tipos distintos de sistemas antes de presentar cómo se representan los números mediante una ordenador.

7.1. Sistema decimal

Hoy día, el mundo utiliza el **sistema decimal** para los números desarrollado por matemáticos árabes en el siglo VIII. Los primeros en usar un sistema numérico decimal fueron los antiguos egipcios. Los babilonios mejoraron el sistema egipcio al dar un significado a las posiciones del sistema numérico. Todos comprendemos fácilmente el sistema numérico decimal.

De hecho lo hemos usado tanto que es básicamente intuitivo. Pero, ¿realmente entendemos por qué la segunda posición en el sistema decimal representa las decenas y la tercera, las centenas? La respuesta yace en las potencias de la base del sistema, que es 10 en el sistema decimal. De esta manera la primera posición es 10 elevado a la potencia 0, la segunda posición es 10 elevado a la potencia 1 y la tercera posición es 10 elevado a la potencia 2. La figura 2.12 muestra la relación entre las potencias y el número 243.

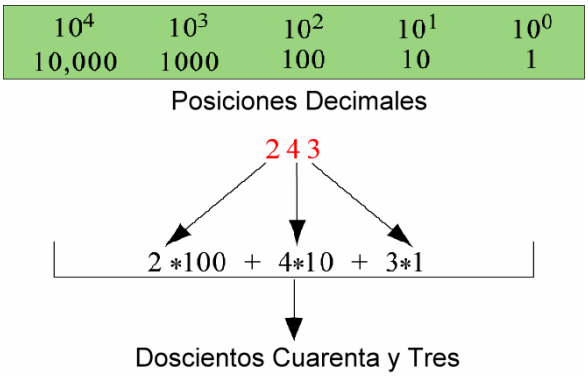


Figura 2.12. Sistema decimal

7.2. Sistema binario

Mientras que el sistema decimal se basa en 10, el **sistema binario** se basa en 2. Sólo hay dos dígitos en el sistema binario, 0 y 1. La figura 2.13 muestra los valores posicionales para un sistema binario y el dígito 243 en binario. En la tabla de posiciones, cada posición es el doble de la posición anterior. De nuevo, esto se debe a que la base del sistema es 2. Las potencias binarias deben memorizarse cuando menos hasta 2^{10}.

Figura 2.13. Sistema binario.

8. Conversión

8.1. Conversión de binario a decimal

Antes de estudiar cómo los números en forma de patrones de bits se almacenan dentro de una ordenador, debes comprender cómo convertir manualmente un número del sistema decimal al sistema binario y viceversa.

Comenzaremos por convertir un número del sistema binario al sistema decimal. Toma el número binario y multiplica cada dígito binario por el valor de su posición. Como cada bit binario puede ser sólo 0 o 1, el resultado será ya sea 0 o el valor posicional. Después de multiplicar todos los dígitos suma los resultados. La **conversión de binario a decimal** se muestra en la figura 2.14.

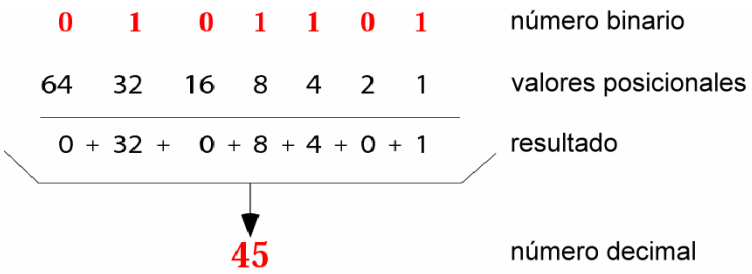


Figura 2.14. Conversión de binario a decimal

Ejemplo 1

Convierta el número binario 10011 a decimal.

Solución

Se escriben los bits y sus valores posicionales. Se multiplica el bit por su valor correspondiente y luego se anota el resultado. Al final, los resultados se suman para obtener el número decimal.

```
Binario            1     0       0       1       1
Valores           16     8       4       2       1
posicionales  ------------------------------------
                  16  +  0   +   0   +   2   +   1
 Decimal                                       19
```

8.2. Conversión de decimal a binario

Para convertir del sistema decimal al binario utilice la división repetitiva. El número original, 45 en el ejemplo, se divide por 2. El residuo (1) se vuelve el primer dígito binario y el segundo dígito se obtiene

dividiendo el cociente (22) por 2 para determinar la siguiente posición. Este proceso continúa hasta que el cociente es 0. La **conversión de decimal a binario** se muestra en la figura 2.15.

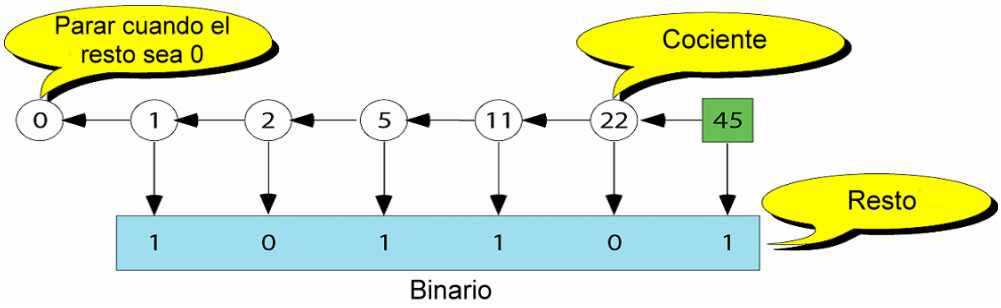


Figura 2.15. Conversión decimal a binario.

Ejemplo 2

Convierta el número decimal 35 a binario.

Solución

Se escribe el número en la esquina derecha. Se divide el número repetidamente por 2 y se anota el cociente y el residuo. Los cocientes se mueven a la izquierda y el residuo se anota bajo cada operación. Este proceso se suspende cuando el cociente es 0.

0	←	1	←	2	←	4	←	8	←	17	←	35	(Decimal)
		↓		↓		↓		↓		↓		↓	
Binario		1		0		0		0		1		1	

9. Representación de enteros

Ahora que sabe cómo convertir del sistema decimal al sistema binario, veamos cómo almacenar enteros dentro de una ordenador. Los **enteros** son **números íntegros** (es decir, números sin una fracción). Por ejemplo, 134 es un entero, pero 134.23 no lo es. Como otro ejemplo, —134 es un entero, pero -134.567 no lo es.

Un entero puede ser positivo o negativo. Un **entero negativo** varía del infinito negativo a 0; un **entero positivo** varía de 0 al infinito positivo (figura 2.16). Sin embargo, ninguna ordenador puede almacenar todos los enteros en este intervalo. Para hacerlo, requeriría un número infinito de bits, lo cual significa una ordenador con una capacidad de almacenamiento infinita.

Figura 2.16. Intervalo de enteros.

Para usar la memoria de una ordenador de manera más eficiente, se han desarrollado dos amplias categorías de representación de enteros: enteros sin signo y enteros con signo. Los enteros con signo también pueden representarse de tres maneras distintas (figura 2.17).

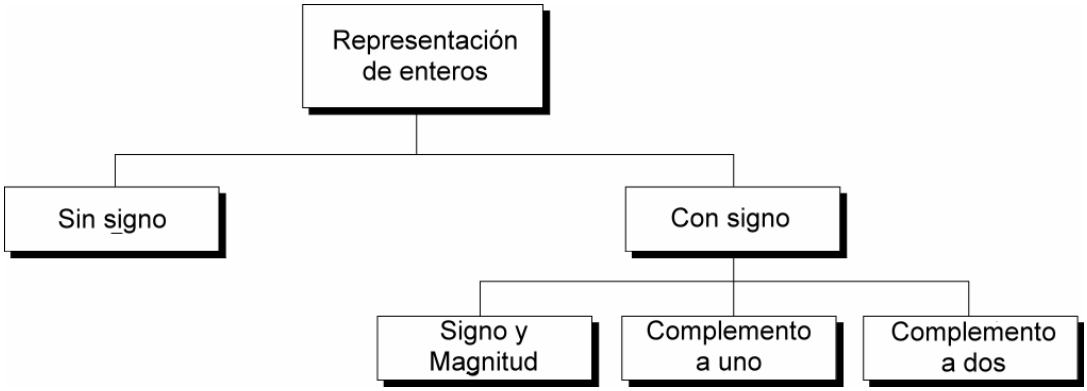


Figura 2.17. Taxonomía de enteros.

En la actualidad la representación de uso más común es el complemento a dos. Sin embargo, primero estudiamos las otras representaciones debido a que son más simples y sirven como una buena base para el complemento a dos.

Un **entero sin signo** es un entero que no tiene intervalo, su rango está entre 0 y el infinito positivo. No obstante, como no hay manera de que una ordenador represente a todos los enteros en este intervalo, la mayoría de los ordenadores define una constante llamada el entero máximo sin signo. Un entero sin signo varía entre 0 y esta constante. El entero máximo sin signo depende del número de bits que el ordenador asigna para almacenar un entero sin signo. A continuación se define el intervalo de los enteros sin signo en una ordenador, donde *N* es el número de bits asignado para representar un entero sin signo:

Intervalo: $0 \ldots (2^N - 1)$

La tabla 2.2 muestra dos intervalos comunes para los ordenadores de hoy.

Número de bits	Intervalo
8	0 ... 255
16	0 ... 65535

Tabla 2.2. Intervalo de enteros sin signo.

Representación

El almacenamiento de los enteros sin signo es un proceso sencillo según se esboza en los pasos siguientes:

1. El número cambia a binario.
2. Si el número de bits es menor que *N*, se añaden 0 a la izquierda del número binario de manera que haya un total de *N* bits.

Ejemplo 3

Almacene 7 en un registro de memoria de ocho bits.

Solución

Primero se cambia el número a binario: 111. Se añaden cinco 0 para hacer un total de N (8) bits: 00000111. El número se almacena en la memoria.

Ejemplo 4

Almacene 258 en un registro de la memoria de 16 bits.

Solución

Primero se cambia el número a binario: 100000010. Se añaden siete 0 para hacer un total de *N* (16) bits: 0000000100000010. El número se almacena en la localidad de la memoria.

La tabla 2.3 muestra cómo se almacenan los enteros no asignados en dos ordenadores diferentes: una usa registros de ocho bits y la otra usa registros de 16 bits. Observe que los números decimales 258 y 24760 no pueden almacenarse en una ordenador que use registros de ocho bits para un entero sin signo. El número decimal 1245678 no puede almacenarse en ninguna de estos dos ordenadores; a esta condición se le llama desbordamiento.

Decimal	Localidad de 8 bits	Localidad de 16 bits
7	00000111	0000000000000111
234	11101010	0000000011101010
258	Desbordamiento	0000000100000010
24760	Desbordamiento	0110000010111000
1245678	Desbordamiento	Desbordamiento

Tabla 2.3. Almacenamiento de enteros sin signo en dos ordenadores diferentes.

Interpretación

¿Cómo se interpreta una representación binaria sin signo en decimal? El proceso es simple. Cambie los *N* bits del sistema binario al sistema decimal como se mostró en una sección anterior.

Ejemplo 5

Interprete 00101011 en decimal si el número se almacenó como un entero sin signo.

Solución

Usando el procedimiento mostrado en la figura 3.3, el número decimal es 43.

Desbordamiento

Si se intenta almacenar un entero sin signo como 256 en un registro de memoria de ocho bits, se obtiene una condición llamada desbordamiento *(overflow).*

Aplicaciones

La representación de enteros sin signo puede mejorar la eficiencia del almacenamiento debido a que usted no necesita almacenar el signo de un entero. Esto significa que el registro de bits del entero puede utilizarse para almacenar el número. La representación de enteros sin signo puede usarse siempre que no se necesiten los enteros negativos. En seguida se listan algunos casos:

- Conteo. Cuando se cuenta, no se necesitan los números negativos. Usted comienza contando a partir de 1 (a veces de 0) y continúa.
- Direccionamiento. Algunos lenguajes de computación almacenan la dirección de un registro de memoria dentro de otro registro de memoria. Las direcciones son números positivos que comienzan a partir de 0 (el primer byte de memoria) y continúan hasta un número que representa la capacidad de memoria total en bytes. De nuevo, no se necesitan números negativos. Los enteros sin signo pueden hacer el trabajo fácilmente.

9.1. Formato de signo y magnitud

El almacenamiento de un entero en el formato de signo y magnitud requiere 1 bit para representar el signo (0 para positivo, 1 para negativo). Esto significa que en una asignación de ocho bits, sólo se pueden usar siete bits para representar el valor absoluto del número (número sin el signo). Por consiguiente, el máximo valor positivo es la mitad del valor sin signo. Lo siguiente define el intervalo de enteros de signo y magnitud en una ordenador, donde *N* es el número de bits asignados para representar a un entero de signo y magnitud:

Intervalo: $\quad -(2^{N-1} - 1) + \ldots + (2^{N-1} - 1)$

La tabla 2.4 muestra los intervalos comunes para los ordenadores actuales. Observe que en este sistema hay dos 0: +0 y —0.

En la representación de signo y magnitud hay dos 0: positivo y negativo. En una asignación de ocho bits:

```
+0   →   00000000
-0   →   10000000
```

Número de bits	Rango
8	-127 -0 +0 + 127
16	-32767 -0 +0 + 32 767
32	-2147483647 -0 +0 +2147483647

Tabla 2.4. Intervalo de enteros de signo y magnitud.

Representación

Almacenar enteros de signo y magnitud es un proceso sencillo:

1. El número se cambia a binario; el signo se ignora.
2. Si el número de bits es menor que *N — 1*, los 0 se añaden a la izquierda del número de manera que haya un total de *N — 1* bits.
3. Si el número es positivo, un 0 se añade a la izquierda (para volverlo de *N* bits). Si el número es negativo, se añade un 1 a la izquierda (para hacerlo de *N* bits).

En la representación de signo y magnitud, el bit en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

Ejemplo 6

Almacene +7 en un registro de memoria de ocho bits usando una representación de signo y magnitud.

Solución

Primero se cambia el número a binario: 111. Se añaden cuatro 0 para hacer un total de *N* — 1 (7) bits: 0000111. Luego se añade un cero más, que aquí se muestra en negritas, ya que el número es positivo. El resultado es **0**0000111.

Ejemplo 7

Almacene —258 en un registro de memoria de 16 bits usando una representación de signo y magnitud.

SOLUCIÓN

Primero se cambia el número a binario: 100000010. Se añaden seis 0 para hacer un total de *N* — 1 (15) bits: 000000100000010. Se añade un 1 más, que aquí se muestra en negrita, puesto que el número es negativo. El resultado es **1**000000100000010.

La tabla 2.5 muestra cómo se almacenan los números de signo y magnitud en dos ordenadores diferentes; una que usa registro de ocho bits y una que usa registro de 16 bits.

Decimal	Registro de 8 bits	Registro de 16 bits
+7	00000111	0000000000000111
-124	11111100	1000000001111100
+258	Desbordamiento	0000000100000010
-24 760	Desbordamiento	1110000010111000

Tabla 2.5. Almacenamiento de enteros de signo y magnitud en dos ordenadores diferentes.

Interpretación

¿Cómo se interpreta una representación binaria de signo y magnitud en decimal? El proceso es simple:

1. Se ignora el primer bit (el que está en el extremo izquierdo).
2. Se cambia los *N — 1* bits de binario a decimal como se explicó al principio.
3. Se agrega un signo + o — al número con base en el bit que está en el extremo izquierdo.

EJEMPLO 8

Interpretar 10111011 en decimal si el número se almacenó como un entero de signo y magnitud.

SOLUCIÓN

Al ignorar el bit que está en el extremo izquierdo, los bits restantes son 0111011. Este número en decimal es 59. El bit en el extremo izquierdo es 1, así que el número es —59.

Aplicaciones

Actualmente, la representación de signo y magnitud no se usa para que los ordenadores actuales almacenen números con signo. Hay cuando menos dos razones para ello. Primero, las operaciones como la suma y la resta no son sencillas para esta representación. Segundo, hay dos 0 en esta representación que vuelven las cosas difíciles para los programadores. Sin embargo, la representación de signo y magnitud tiene una ventaja: la transformación de decimal a binario y viceversa es muy fácil. Esto hace que esta representación sea conveniente para aplicaciones que no necesitan operaciones con números. Un ejemplo es el cambio de señales analógicas a señales digitales. Se muestrea la señal analógica, se asigna un número positivo o negativo a la muestra y se cambia a binario para enviarlo por canales de comunicación de datos.

9.2. El formato de complemento a uno

Tal vez se haya notado que la representación de un número en el sistema binario es una cuestión de convención. En la representación de signo y magnitud adoptamos la convención de que el bit que está en el extremo izquierdo representa el signo; este bit no es parte del valor.

Los diseñadores de la **representación del complemento a uno** adoptaron una convención diferente: para representar un número positivo, usan la convención adoptada para un entero sin signo. Y para representar un número negativo, complementan el número positivo. En otras palabras, +7 se representa justo como un número sin signo, mientras que —7 se representa como el complemento de +7. En el complemento a uno, el complemento de un número se obtiene al cambiar todos los 0 a 1 y todos los 1 a 0.

A continuación se define el intervalo de los enteros complemento a uno en una ordenador, donde *N* es el número de bits asignados para representar un entero complemento a uno:

Intervalo: $-(2^{N-1}-1)$ ... $+(2^{N-1}-1)$

Existen dos 0 en la representación del complemento a uno: positivo y negativo. En una asignación de 8 bits:

```
+0   →   00000000
-0   →   11111111
```

La tabla 2.6 muestra los intervalos comunes actuales para los ordenadores. Observe que en este sistema hay al menos dos 0: un +0 y un —0.

Número de bits	Intervalo			
8	-127	-0	+0	+ 127
16	-32767	-0	+0	+ 32767
32	-2147483647	-0	+0	+2147483647

Tabla 2.6. Rango de los enteros complemento de uno.

Representación

Para almacenar los enteros complemento de uno se siguen estos pasos:

1. Cambie el número a binario; el signo es ignorado.
2. Añada uno o varios 0 a la izquierda del número para hacer un total de *N* bits.
3. Si el signo es positivo, no se necesita ninguna otra acción. Si el signo es negativo, se complementa cada bit (se cambia 0 por 1 y 1 por 0).

En la representación del complemento de uno, el bit que está en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

Ejemplo 9

Almacene +7 en un registro de memoria de ocho bits usando la representación de complemento a uno.

Solución

Primero se cambia el número a binario: 111. Se añaden cinco 0 de modo que haya un total de *N(8)* bits: 00000111. El signo es positivo así que no se requiere realizar otra acción.

Ejemplo 10

Almacene —256 en un registro de memoria de 16 bits usando la representación del complemento de uno.

Solución

Primero se cambia el número a binario: 100000010. Se añaden siete 0 con el fin de que haya un total de *N*(16) bits: 0000000100000010. El signo es negativo, de manera que cada bit complementa. El resultado es 1111111011111101.

La tabla 2.7 muestra cómo se almacenan los números en complemento a uno en dos ordenadores distintas: una que usa registros de ocho bits y otra que usa registros de 16 bits.

Decimal	Localidades de 8 bits	Localidades de 16 bits
+7	00000111	0000000000000111
–7	11111100	1111111111111000
+ 124	01111100	0000000001111100
–124	10000011	1111111110000011
+ 24760	Desbordamiento	0110000010111000
–24760	Desbordamiento	1001111101000111

Tabla 2.7. Almacenamiento de enteros complemento a uno en dos ordenadores distintas.

Interpretación

¿Cómo se interpreta una representación binaria del complemento a uno en decimal? El proceso implica estos pasos:

1. Si el bit que está en el extremo izquierdo es 0 (número positivo),
   a. se cambia el número entero de binario a decimal.
   b. se pone un signo más (+) enfrente del número.
2. Si el bit que está en el extremo izquierdo es 1 (bit negativo),
   a. se complementa el número entero (cambiando todos los 0 a 1, y viceversa).
   b. se cambia el número entero de binario a decimal.
   c. se pone un signo negativo (—) enfrente del número.

Ejemplo 11

Interprete 11110110 en decimal si el número se almacenó como un entero complemento a uno.

Solución

El bit en el extremo izquierdo es 1, de modo que el número es negativo. Primero se complementa. El resultado es 00001001. El complemento en decimal es 9, de manera que el número original era —9. Obsérvese que el complemento de un complemento es el número original.

La operación complemento a uno significa invertir todos los bits. Si se aplica la operación complemento a uno a un número positivo, se obtiene el número negativo correspondiente. Si se aplica la operación complemento a uno a un número negativo, se obtiene el número positivo correspondiente. Si un número se complementa dos veces, se obtiene el número original.

Desbordamiento

Si se intenta almacenar un entero complemento a uno como +256 en una localidad de memoria de ocho bits, se obtiene una condición llamada desbordamiento.

Aplicaciones

Actualmente la representación del complemento a uno no se usa para almacenar números en ordenadores. Hay al menos dos razones para ello. Primero, las operaciones como la suma y *la resta no son sencillas para esta representación. Segundo, hay dos 0 en esta representación,* lo cual vuelve las cosas difíciles para los programadores. Sin embargo, esta representación tiene cierta relevancia. Primero, es la base para la siguiente representación (el complemento a dos). Segundo, tiene propiedades que la vuelven interesante para aplicaciones de comunicación de datos tales como la detección y corrección de errores.

9.3. Formato del complemento a dos

Como se mencionó previamente, la representación del complemento a uno tiene dos 0 (+0 y —0). Esto puede crear un poco de confusión en los cálculos. La **representación del complemento a dos** resuelve todos estos problemas.

El complemento a dos es la representación de enteros más común, más importante y de más amplio uso en la actualidad.

A continuación se define el intervalo de los enteros complemento a dos en una ordenador, donde *N* es un número de bits asignados a un entero complemento a dos:

$$\textbf{Intervalo: } -(2^{N-1}) \text{ a } +(2^{N-1}-1)$$

La tabla 2.8 muestra los intervalos comunes actuales para los ordenadores. Obsérvese que en este sistema hay sólo un 0 y que el principio del intervalo es 1 menos que aquel para el complemento a uno.

Número de bits	Intervalo			
8	-128	-0	+0	+ 127
16	-32 768	-0	+0	+ 32767
32	-2147483648	-0	+0	+2147483647

Tabla 2.8. Intervalo de números complemento a dos.

Representación

Para almacenar el complemento a dos se deben seguir estos pasos:

1. El número se cambia a binario; el signo se ignora.

2. Si el número de bits es menor que *N* se añaden 0 a la izquierda del número de manera que haya un total de *N* bits.
3. Si el signo es positivo, no se necesita una acción posterior. Si el signo es negativo, todos los 0 en el extremo derecho y el primer 1 permanecen sin cambios. El resto de los bits se complementa.

En la representación del complemento a dos, el bit en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

Ejemplo 12

Se almacena +7 en un registro de memoria de ocho bits usando la representación del complemento a dos.

Solución

Primero se cambia el número a binario: 111. Se añaden cinco 0 con el fin de que haya un total de *N(8)* bits: 00000111. El signo es positivo, así que no se requiere realizar ninguna otra acción.

Ejemplo 13

Almacene —40 en un registro de memoria de 16 bits usando la representación del complemento a dos.

Solución

Primero se cambia el número a binario: 101000. Se añaden diez 0 de modo que haya un total de *N(16)* bits: 0000000000101000. El signo es negativo, así que los 0 en el extremo derecho permanecen sin cambio hasta el primer 1 (inclusive) y el resto se complementa. El resultado es 1111111111011000.

La tabla 2.9 muestra cómo se almacenan los números complemento a dos en dos ordenadores distintos, una que usa registro de ocho bits y otra que usa registro de 16 bits.

Decimal	Localidad de 8 bits	Localidad de 16 bits
+7	00000111	0000000000000111
–7	11111001	1111111111111001
+ 124	01111100	0000000001111100
–124	10000100	1111111110000100
+24760	Desbordamiento	0110000010111000
–24760	Desbordamiento	1001111101001000

Tabla 2.9. Ejemplo de representaciones de complementos a dos en dos ordenadores.

Sólo hay un 0 en el complemento a dos. En una asignación de 8 bits:

0 → 00000000

Interpretación

¿Cómo se interpreta una representación binaria del complemento a dos en decimal? El proceso involucra los pasos siguientes:

1. Si el bit en el extremo izquierdo es 0 (número positivo),
   a. se cambia todo el número de binario a decimal.
   b. se pone un signo más (+) enfrente del número.
2. Si el bit en el extremo izquierdo es 1 (número negativo),
   a. se dejan los bits en el extremo derecho hasta el primer 1 (inclusive) como están.
   b. se cambia todo el número de binario a decimal.
   c. se pone un signo negativo (—) enfrente del número.

Ejemplo 14

Interprete 11110110 en decimal si el número se almacenó como un entero complemento a dos.

Solución

El bit en el extremo izquierdo es 1. El número es negativo. El 10 a la derecha se deja tal como está y el resto se complementa. El resultado es 00001010. El número complemento a dos es 10, así que el número original era —10.

La operación complemento a dos puede lograrse al invertir todos los bits, excepto los bits que están en el extremo derecho hasta el primer 1 (inclusive). Si se aplica la operación complemento a dos a un número positivo, se obtiene el número negativo correspondiente. Si se aplica la operación complemento a dos a un número negativo, se obtiene el número positivo correspondiente. Si un número se complementa dos veces, se obtiene el número original.

Aplicaciones

La representación del complemento a dos es la representación estándar actual para almacenar enteros en los ordenadores: esto ocurre cuando se considera la simplicidad de las operaciones que usan el complemento a dos.

9.4. Resumen de la representación de enteros

Para darse una idea general de los métodos de representación de números, examine la tabla 2.102.10. En esta tabla, suponga que *N* es 4. La localidad de memoria puede almacenar sólo cuatro bits. Si usted mira el contenido de la localidad de memoria, puede interpretar el número en una de las cuatro representaciones. Aun cuando la interpretación es la misma para los enteros positivos, es diferente para los enteros negativos.

Contenido de la memoria	Sin signo	Signo y magnitud	Complemento a uno	Complemento a dos
0000	0	+0	+0	+0
0001	1	+ 1	+ 1	+ 1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7

1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Tabla 2.10. Resumen de la representación de enteros.

10. Sistema Excess

Otra representación que permite almacenar tanto números positivos como negativos en una ordenador es el **sistema Excess.** En este sistema, es fácil transformar un número de decimal a binario, y viceversa. Sin embargo, las operaciones con los números son muy complicadas. Su única aplicación en uso actualmente es en el almacenamiento del valor exponencial de una fracción. Esto se estudia en la sección siguiente.

En una conversión Excess, un número positivo conocido como el número mágico se utiliza en el proceso de conversión. El número mágico normalmente es (2^{N-1}) o $(2^{N-1} - 1)$ donde *N* es la asignación de bits. Por ejemplo, si *N* es 8, el número mágico es ya sea 128 o 127. En el primer caso, llamamos a la representación Excess_128 y en el segundo, Excess_127.

Representación

Para representar un número en Excess. se utiliza el procedimiento siguiente:

1. Se suma el número mágico al entero.
2. Se cambia el resultado a binario y añada uno o varios 0 de modo que haya un total de *N* bits.

Ejemplo 15

Representar —125 en Excess_127 usando registros de ocho bits.

Solución

Primero se suma 127 a —25 y se obtiene 102. Este número en binario es 1100110. Se añade un bit para obtener una longitud de ocho bits. La representación es 01100110.

Interpretación

Para interpretar un número en Excess, utilice el siguiente procedimiento:

1. Cambie el número a decimal.
2. Reste el número mágico del entero.

Ejemplo 16

Interpretar **11111110** si la representación está en Excess_127.

Solución

Primero se cambia el número a decimal: 254. Luego se resta 127 del número. El resultado es 127 en decimal.

11. Representación en Punto Flotante

Para representar un **número en punto flotante** (un número que contiene un entero y una **fracción),** el número se divide en dos partes: el entero y la fracción. Por ejemplo, el número en punto flotante 14.234 tiene un entero de 14 y una fracción de 0.234.

11.1. Conversión a binario

Para convertir un número de punto flotante a binario, utilice el procedimiento siguiente:

1. Se convierte la parte entera a binario.
2. Se convierte la fracción a binario.
3. Se pone un punto decimal entre las dos partes.

Conversión de la parte entera

Este procedimiento es el mismo que se presentó en la sección 8.2.

Conversión de la parte fraccionaria

Para convertir una fracción a binario, se usa la multiplicación repetitiva. Por ejemplo, para convertir 0.125 a binario, se multiplica la fracción por 2; el resultado es 0.250. La parte entera del resultado (0) se extrae y se vuelve el dígito binario en el extremo izquierdo. Ahora se multiplica por 2 la parte fraccionaria (0.250) del resultado para obtener 0.50. De nuevo, se extrae la parte entera del resultado y se vuelve el siguiente dígito binario. Este proceso continúa hasta que la parte fraccionaria se vuelve 0 o cuando se llega al límite del número de bits que se pueden usar (figura 2.18).

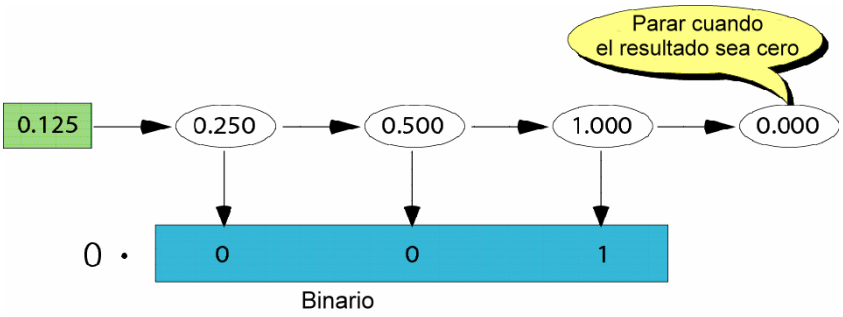


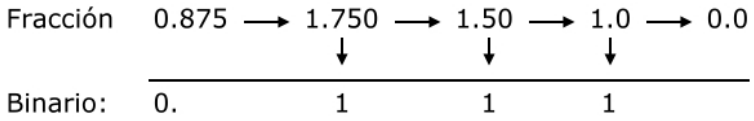
Figura 2.18. Cambio de fracciones a binario.

Ejemplo 17

Transformar la fracción 0.875 a binario.

Solución

La fracción se escribe en la esquina izquierda. El número se multiplica continuamente por 2 y se extrae la parte entera como dígito binario. El proceso se detiene cuando el número es 0.0.

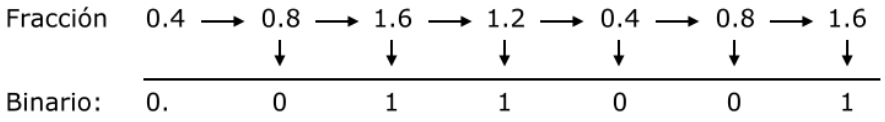


Ejemplo 18

Transformar la fracción 0.4 a un binario de 6 bits.

Solución

La fracción se escribe en la esquina izquierda. El número se multiplica continuamente por 2 y se extrae la parte entera como dígito binario. En este caso, no se puede obtener la representación binaria exacta debido a que reaparece la fracción original. Sin embargo, se puede continuar con el proceso hasta obtener seis bits.



11.2. Normalización

Para representar el número 71.3125 (+1000111.0101), en la memoria se almacena el signo, todos los bits y la posición del punto decimal. Aunque esto es posible, dificulta las operaciones con números. Se necesita una representación estándar para números de punto flotante. La solución es la normalización, es decir, el desplazamiento del punto decimal para que haya sólo un 1 a la izquierda del punto decimal.

1 . xxxxxxxxxxxxxxxxxx

Para indicar el valor original del número, éste se multiplica por 2^e, donde *e* es el número de bits en que se desplazó el punto decimal: positivo para el desplazamiento izquierdo y negativo para el desplazamiento derecho. Un signo positivo o negativo se añade luego dependiendo del signo del número original. La tabla 2.11 muestra ejemplos de la normalización.

Número original	Desplazamiento	Normalizado
+ 1010001.11001	← 6	$+2^{+6}$ x 1.01000111001
-111.000011	← 2	-2^{+2} x 1.11000011
+0.00000111001	6 →	$+2^{-6}$ x 1.11001
-0.001110011	3 →	-2^{-3} x 1.110011

Tabla 2.11. Ejemplos de la normalización.

11.3. Signo, exponente y mantisa

Después de que se normaliza un número, se almacenan sólo tres piezas de información del mismo: signo, exponente y mantisa (los bits a la derecha del punto decimal). Por ejemplo, + 1000111.0101 se convierte en:

$$+ \quad 2^6 \qquad\qquad x \quad 1.0001110101$$

Signo: + Exponente: 6 Mantisa: 0001110101

Obsérvese que el 1 a la izquierda del punto decimal no se almacena; esto es comprensible.

Signo

El signo de un número puede almacenarse usando 1 bit (0 o 1).

Exponente

El exponente (potencia de 2) define el movimiento del punto decimal. Observe que la potencia puede ser negativa o positiva. El método que se utiliza para almacenar el exponente es la representación de Excess. El número de bits asignado *N* define el intervalo de números que una ordenador puede almacenar.

Mantisa

La mantisa es el número binario a la derecha del punto decimal. Define la precisión del número. La mantisa se almacena como un entero sin signo.

11.4. Estándares IEEE

El Instituto de Ingenieros Electrónicos y Eléctricos (IEEE: *Institute of Electrical and Electronics Engineers*) ha definido tres estándares para almacenar números de punto flotante; dos de ellos se utilizan para almacenar números en la memoria (precisión simple y precisión doble). Estos formatos se muestran en la figura 2.19. Observe que el número dentro de los cuadros es el número de bits para cada campo. Analizamos el formato de precisión simple y dejamos el **formato de precisión doble** como un ejercicio.

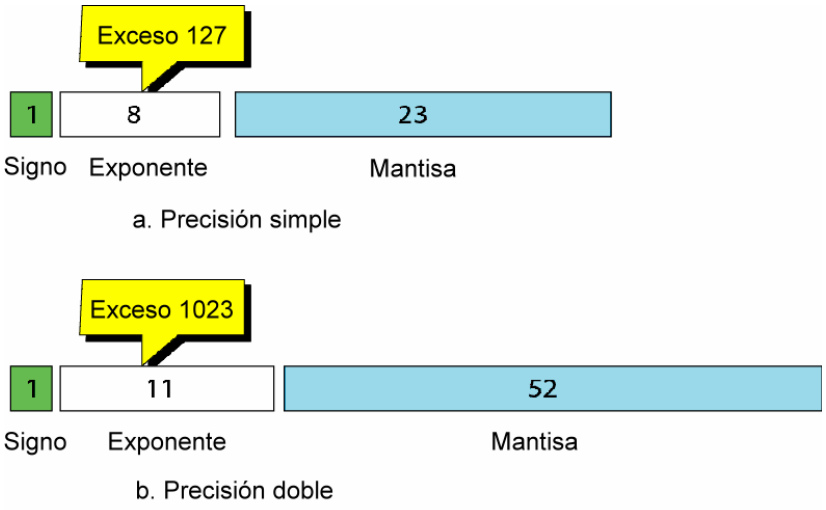


Figura 2.19. Estándares del IEEE para la representación de punto flotante.

Representación de precisión simple

El procedimiento para almacenar un número en punto flotante en la memoria mediante el uso del formato de precisión simple es el siguiente:

1. Se almacena el signo como 0 (positivo) o 1 (negativo).
2. Se almacena el exponente (potencia de 2) como Excess_127.
3. Se almacena la mantisa como un entero sin signo.

Ejemplo 19

Muestre la representación del número normalizado

$$+ \quad 2^6 \quad X \quad 1.01000111001$$

Solución

El signo es positivo y se representa como 0. El exponente es 6. En la representación Excess_127, se le suma 127 y se obtiene 133. En binario es 10000101. La mantisa es 01000111001. Cuando se aumenta la longitud a 23, se obtiene 01000111001000000000000. Adviértase que no se puede ignorar el 0 a la izquierda porque es una fracción, sin considerar que el 0 es lo mismo que multiplicar el número por 2. Obsérvese que los 0 se añadieron a la derecha (no a la izquierda) debido a que se trata de una fracción. La adición de 0 a la derecha de una fracción no modifica a la fracción, pero añadir 0 a la izquierda significa dividir el número entre una potencia de 2. El número en la memoria es un número de 32 bits como se muestra en seguida:

Signo	exponente	mantisa
0	**10000101**	01000111001000000000000

La tabla 2.12 exhibe más ejemplos de representación de punto flotante.

Número			Signo	Exponente	Mantisa
-2^2	x	1.11000011	1	10000001	11000011000000000000000
$+ 2^{-6}$	x	1.11001	0	01111001	11001000000000000000000
-2^{-3}	x	1.110011	1	01111100	11001100000000000000000

Tabla 2.12. Ejemplos de representación de punto flotante.

Interpretación de punto flotante para precisión simple

El procedimiento siguiente interpreta un número de punto flotante de 32 bits almacenado en la memoria.

1. Se usa el bit del extremo izquierdo como el signo.
2. Se cambian los siguientes ocho bits a formato decimal y se resta 127 del resultado. Éste es el exponente.
3. Se añade un 1 y un punto decimal a los siguientes 32 bits. Se puede ignorar cualquier 0 adicional a la derecha.
4. Se mueve el punto decimal a la posición correcta usando el valor del exponente.
5. Se cambia la parte entera a decimal.
6. Se cambia la parte fraccionaria a decimal.
7. Se combina las partes entera y fraccionaria.

Ejemplo 20

Interprete el siguiente número de punto flotante de 32 bits:

```
1 01111100 11001100000000000000000
```

Solución

El bit en el extremo izquierdo es el signo (—). Los siguientes ocho bits son 01111100. Este es 124 en decimal. Si se le resta 127, se obtiene el exponente —3. Los 23 bits siguientes son la mantisa. Si se ignoran

los 0 adicionales, se obtiene 110011. Después de que se añade un 1 a la izquierda del punto decimal, el número normalizado en binario es:

$$-2^{-3} \quad x \quad 1.110011$$

12. Resumen de representación de datos

- Los números, el texto, las imágenes, el audio y el video, todos son formas de datos. Los ordenadores necesitan procesar todo tipo de datos.
- Todos los tipos de datos se transforman en una representación uniforme llamada patrón de bits para su procesamiento por el ordenador.
- Un bit es la unidad más pequeña de datos que puede almacenarse en una ordenador.
- Un interruptor, con sus dos estados de encendido y apagado, puede representar un bit.
- Un patrón de bits es una secuencia de bits que pueden representar un símbolo.
- Un byte son ocho bits.
- La codificación es el proceso de transformar datos en un patrón de bits.
- ASCII es un código popular para los símbolos.
- EBCDIC es un código utilizado en los mainframes de IBM.
- Unicode es un código de 16 bits y la ISO ha desarrollado un código de 32 bits. Ambos códigos permiten un mayor número de símbolos.
- Las imágenes utilizan el método de gráficos de mapa de bits o gráficos de vectores para representación de datos. La imagen se divide en pixeles a los que luego pueden asignarse patrones de bits.
- Los datos de audio se transforman a patrones de bits a través del muestreo, la cuantificación y la codificación.
- Los datos de video son una serie de imágenes en secuencia.
- El sistema decimal tiene diez dígitos y se basa en potencias de 10.
- El sistema binario, utilizado por ordenadores para almacenar números, tiene dos dígitos, 0 y 1, y se basa en potencias de 2.
- La asignación de bits es el número de bits utilizado para representar a un entero.
- Los enteros pueden representarse como números enteros con signo y sin signo.
- Existen tres métodos principales de representación de números con signo: signo y magnitud, complemento a uno y complemento a dos.
- Los números sin signo se usan comúnmente para conteo y direccionamiento.
- En el método de signo y magnitud de la representación de enteros, un bit representa el signo de número; los bits restantes representan la magnitud.
- En el método de representación de enteros del complemento a uno, un número negativo se representa por medio de la operación complemento del número positivo correspondiente.
- Complementar un número significa convertir cada 1 a 0 y cada 0 a 1.
- En el método de representación de enteros del complemento a uno, un número negativo se representa al dejar todos los 0 en el extremo derecho y el primer 1 sin cambio y luego complementar los bits restantes.

- La mayoría de los ordenadores actuales utiliza el método de representación de enteros del complemento a dos.
- Tanto el método de signo y magnitud como el de complemento a uno tienen dos representaciones para el valor 0; el complemento a dos tiene sólo una representación para el valor 0.
- Un número de punto flotante es un número entero y una fracción. La conversión de la fracción a binario requiere que el denominador de la fracción se exprese como una potencia de 2. El sistema Excess_X se utiliza para almacenar esta potencia de 2.
- Una fracción se normaliza con la finalidad de que las operaciones sean más simples.
- Para almacenar una fracción en la memoria, se necesita su signo, exponente y mantisa.

13. Un poco de humor: La creación del ordenador

1. En el principio DIOS creó el Bit y el Byte. Y de ellos creó la Palabra.
2. Y había dos Bytes en la Palabra; y nada mas existía. Y Dios separó el Uno del Cero: y vió que era bueno.
3. Y Dios dijo: que se hagan los Datos; y así pasó. Y Dios dijo: Dejemos los Datos en sus correspondientes sitios. Y creó los disketes, los discos duros y los discos compactos.
4. Y Dios dijo: que se hagan los ordenadores, así habrá un lugar para poner los disketes, los discos duros y los discos compactos. Así Dios creó a los ordenadores, y les llamó hardware.
5. Pero aun no había software. Pero Dios creó los programas; grandes y pequeños... Y les dijo: Creced y multiplicaos y llenad toda la memoria.
6. Y Dios dijo: crearé el Programador; y el Programador creará nuevos programas y gobernará los ordenadores y los programas y los datos.
7. Y Dios creó al Programador; y lo puso en el Centro de Datos; y Dios le enseñó al Programador el Directorio y le dijo: Puedes usar todos los volúmenes y subdirectorios, pero NO USES Windows.
8. Y Dios dijo: no es bueno que el Programador esté solo. Cogió un hueso del cuerpo del Programador y creo una criatura que miraría al Programador; y admiraría al Programador; y amaría las cosas que el programador hiciese. Y Dios llamó a la criatura el Usuario.
9. Y el Programador y el Usuario fueron dejados en el desnudo DOS y eso era Bueno.
10. Pero Bill era más listo que todas las otras criaturas de Dios. Y Bill le dijo al Usuario: ¿Te dijo Dios realmente que no ejecutaras todos los programas?
11. Y el Usuario respondió: Dios nos dijo que podíamos usar cualquier programa y cualquier pedazo de datos, pero nos dijo que no ejecutásemos Windows o moriríamos.
12. Bill le dijo al Usuario: ¿Cómo puedes hablar de algo que incluso no has probado?. En el momento en que ejecutes Windows serás igual a Dios. Serás capaz de crear cualquier cosa que quieras con el simple toque del ratón.
13. Y el Usuario vio que los frutos del Windows eran más bonitos y fáciles de usar. Y el Usuario vio que todo conocimiento era inútil, ya que Windows podía reemplazarlo.
14. Así el Usuario instaló Windows en su ordenador; y le dijo al Programador que era bueno.
15. Y el Programador inmediatamente empezó a buscar nuevos controladores. Y Dios le pregunto: ¿que buscas? Y el Programador respondió: Estoy buscando nuevos controladores, porque no puedo encontrarlos en el DOS. Y Dios dijo: ¿quién te dijo que necesitabas nuevos controladores? ¿Acaso ejecutaste Windows? Y el Programador dijo: fue Bill quien nos lo dijo...
16. Y Dios le dijo a Bill: Por lo que hiciste, serás odiado por todas las criaturas. Y el Usuario siempre estará descontento contigo. Y siempre venderás Windows.

17. Y Dios le dijo al Usuario: por lo que hiciste, el Windows te decepcionará y se comerá todos tus recursos; y tendrás que usar malos programas; y siempre permanecerás bajo la ayuda del Programador.
18. Y Dios le dijo al Programador: por haber escuchado al Usuario nunca serás feliz. Todos tus programas tendrán errores y tendrás que corregirlos y corregirlos hasta el fin de los tiempos.
19. Y Dios los echó a todos del Centro de Datos y bloqueó la puerta con una password.

Anexo A: Tabla de códigos ASCII - Formato de caracteres estándares

ASCII = *American Standard Code for Information Interchange*: Estándar Americano de Codificación para el Intercambio de Información (pronunciación: as-ki)

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

Anexo B: Historia de la numeración

Introducción. El Concepto de Base

Cuando los hombres empezaron a contar usaron los dedos, guigarros, marcas en bastones, nudos en una cuerda y algunas otras formas para ir pasando de un número al siguiente. A medida que la cantidad crece se hace necesario un sistema de representación más práctico.

En diferentes partes del mundo y en distintas épocas se llegó a la misma solución, cuando se alcanza un determinado número se hace una marca distinta que los representa a todos ellos. Este número es la base. Se sigue añadiendo unidades hasta que se vuelve a alcanzar por segunda vez el número anterior y se añade otra marca de la segunda clase . Cuando se alcanza un número determinado (que puede ser diferente del anterior constituyendo la base auxiliar) de estas unidades de segundo orden, las decenas en caso de base 10, se añade una de tercer orden y así sucesivamente.

La base que más se ha utilizado a lo largo de la Historia es 10 según todas las apariencias por ser ese el número de dedos con los que contamos. Hay alguna excepción notable como son las numeración babilónica que usaba 10 y 60 como bases y la numeración maya que usaba 20 y 5 aunque con alguna irregularidad.

Desde hace 5000 años la gran mayoría de las civilizaciones han contado en unidades, decenas, centenas, millares etc. es decir de la misma forma que seguimos haciéndolo hoy. Sin embargo la forma de escribir los números ha sido muy diversa y muchos pueblos han visto impedido su avance científico por no disponer de un sistema eficaz que permitiese el cálculo.

Casi todos los sistemas utilizados representan con exactitud los números enteros, aunque en algunos pueden confundirse unos números con otros, pero muchos de ellos no son capaces de representar grandes cantidades, y otros requieren tal cantidad de simbolos que los hace poco prácticos.

Pero sobre todo no permiten en general efectuar operaciones tan sencillas como la multiplicación, requiriendo procedimientos muy complicados que sólo estaban al alcance de unos pocos iniciados. De hecho cuando se empezó a utilizar en Europa el sistema de numeración actual, los abaquistas, los profesionales del cálculo se opusieron con las más peregrinas razones, entre ellas la de que siendo el cálculo algo complicado en sí mismo, tendría que ser un metodo diabólico aquel que permitiese efectuar las operaciones de forma tan sencilla.

El sistema actual fue inventado por los indios y transmitido a Europa por los árabes;. Del origen indio del sistema hay pruebas documentales más que suficientes, entre ellas la opinión de Leonardo de Pisa (Fibonacci) que fue uno de los indroductores del nuevo sistema en la Europa de 1200. El gran mérito fue la introducción del concepto y símbolo del cero, lo que permite un sistema en el que sólo diez simbolos puedan representar cualquier número por grande que sea y simplificar la forma de efectuar las operaciones.

Sistemas de Numeración Aditivos

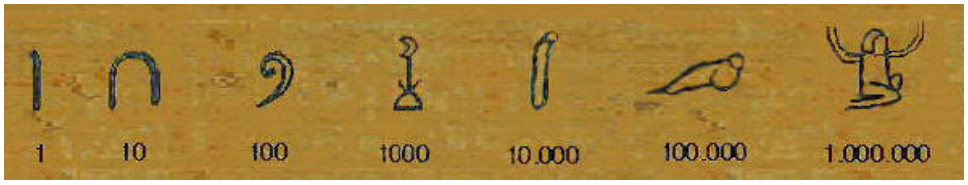
Para ver cómo es la forma de representación aditiva consideremos el sistema jeroglífico egipcio. Por cada unidad se escribe un trazo vertical, por cada decena un símbolo en forma de arco y por cada centena, millar, decena y centena de millar y millón un jeroglífico específico. Así para escribir 754 usaban 7 jeroglíficos de centenas 5 de decenas y 4 trazos. De alguna forma todas las unidades están físicamente presentes.

Los sistemas aditivos son aquellos que acumulan los símbolos de todas las unidades, decenas... como sean necesarios hasta completar el número. Una de sus características es por tanto que se pueden poner los símbolos en cualquier orden, aunque en general se ha preferido una determinada disposición.

Han sido de este tipo las numeraciones egipcia, sumaria (de base 60), hitita, cretense, azteca (de base 20), romana y las alfabéticas de los griegos, armenios, judios y árabes.

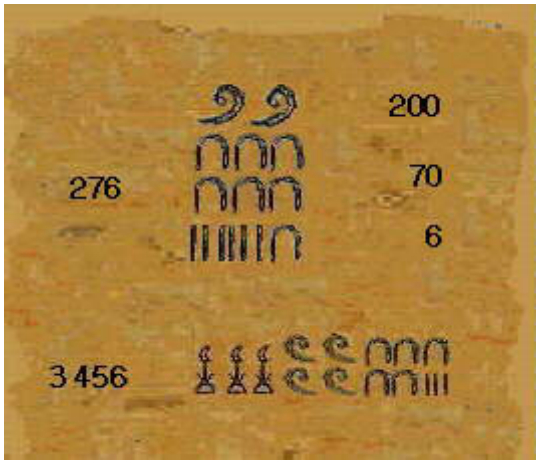
El Sistema de Numeración Egipcio

Desde el tercer milenio A.C. los egipcios usaron un sistema deescribir los números en base diez utilizando los geroglíficos de la figura para representar los distintos ordenes de unidades.



Se usaban tantos de cada uno cómo fuera necesario y se podian escribir indistintamente de izquierda a derecha, al revés o de arriba abajo, cambiando la orientación de las figuras según el caso.

Al ser indiferente el orden se escribían a veces según criterios estéticos, y solían ir acompañados de los geroglíficos correspondientes al tipo de objeto (animales, prisioneros, vasijas etc.) cuyo número indicaban. En la figura aparece el 276 tal y como figura en una estela en Karnak.



Estos signos fueron utilizados hasta la incorporación de Egipto al imperio romano. Pero su uso quedó reservado a las inscripciones monumentales, en el uso diario fue sustituido por la escritura hierática y demótica, formas más simples que permitian mayor rapidez y comodidad a los escribas.

En estos sistemas de escritura los grupos de signos adquirieron una forma propia, y asi se introdujeron símbolos particulares para 20, 30....90....200, 300.....900, 2000, 3000..... con lo que disminuye el número de signos necesarios para escribir una cifra.

El Sistema de Numeración Griego

El primer sitema de numeración griego se desarrolló hacia el 600 A.C. Era un sistema de base decimal que usaba los símbolos de la figura siguiente para representar esas cantidades. Se utilizaban tantas de ellas como fuera necesario según el principio de las numeraciones aditivas.

Para representar la unidad y los números hasta el 4 se usaban trazos verticales. Para el 5, 10 y 100 las letras correspondientes a la inicial de la palabra cinco (pente), diez (deka) y mil (khiloi). Por este motivo se llama a este sistema acrofónico.

Los símbolos de 50, 500 y 5000 se obtienen añadiendo el signo de 10, 100 y 1000 al de 5, usando un principio multiplicativo. Progresivamente este sistema ático fue reemplazado por el jónico, que empleaba las 24 letras del alfabeto griego junto con algunos otros símbolos según la tabla siguiente .

De esta forma los números parecen palabras, ya que están compuestos por letras, y a su vez las palabras tienen un valor numérico, basta sumar las cifras que corresponden a las letras que las componen. Esta circunstancia hizo aparecer una nueva suerte de disciplina mágica que estudiaba la relación entre los números y las palabras. En algunas sociedades como la judía y la árabe, que utilizaban un sistema similar, el estudio de esta relación ha tenido una gran importancia y ha constituido una disciplina aparte: la kábala, que persigue fines místicos y adivinatorios.

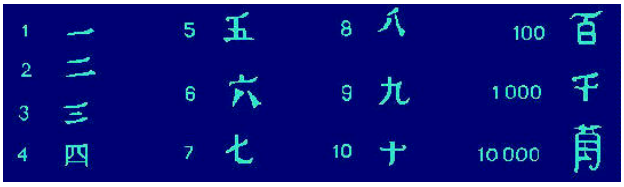
Sistemas de Numeración Híbridos

En estos sistemas se combina el principio aditivo con el multiplicativo. Si para representar 500 los sistemas aditivos recurren a cinco representaciones de 100, los híbridos utilizan la combinación del 5 y el 100. Pero siguen acumulando estas combinaciones de signos para los números más complejos. Por lo tanto sigue siendo innecesario un símbolo para el 0. Para representar el 703 se usa la combinación del 7 y el 100 seguida del 3.

El orden en la escritura de las cifras es ahora fundamental para evitar confusiones, se dan así los pasos para llegar al sistema posicional, ya que si los signos del 10, 100 etc se repiten siempre en los mismos lugares, pronto alguien piensa en suprimirlos, dándolos por supuestos y se escriben sólo las cifras correspondientes a las decenas, centenas etc. .Pero para ello es necesario un cero, algo que indique que algún orden de magnitud está vacío y no se confundan el 307 con 370, 3070 ...

Además del chino clásico han sido sistemas de este tipo el asirio, arameo, etíope y algunos del subcontinente indio cómo el tamil, el malayalam y el cingalés.

El Sistema de Numeración Chino



La forma clásica de escritura de los números en China se empezó a usar desde el 1500 A.C. aproximadamente. Es un sistema decimal estricto que usa las unidades y los distintas potencias de 10. Utiliza los ideogramas de la figura y usa la combinación de los números hasta el diez con la decena, centena, millar y decena de millar para según el principio multiplicativo representar 50, 700 ó 3000. El orden de escritura se hace fundamental,ya que 5 10 7 igual podría representar 57 que 75.

Tradicionalmente se ha escrito de arriba abajo aunque también se hace de izquierda a derecha como en el ejemplo de la figura. No es necesario un símbolo para el cero siempre y cuando se pongan todos los ideogramas, pero aún así a veces se suprimían los correspondientes a las potencias de 10.

Aparte de esta forma que podríamos llamar canónica se usaron otras. Para los documento importantes se usaba una grafía más complicada con objeto de evitar falsificaciones y errores. En los sellos se escribía de forma más estilizada y lineal y aún se usaban hasta dos grafías diferentes en usos domésticos y comerciales, aparte de las variantes regionales. Los eruditos chinos por su parte desarrollaron un sistema posicional muy parecido al actual que desde que incorporó el cero por influencia india en s. VIII en nada se diferencia de este.

Sistemas de Numeración Posicionales

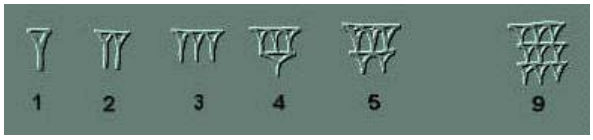
Mucho más efectivos que los sitemas anteriores son los posicionales. En ellos la posición de una cifra nos dice si son decenas, centenas ... o en general la potencia de la base correspondiente.

Sólo tres culturas además de la india lograron desarrollar un sistema de este tipo. Babilonios, chinos y mayas en distintas épocas llegaron al mismo principio. La ausencia del cero impidió a los chinos un desarrollo completo hasta la intraducción del mismo. Los sistemas babilónico y maya no eran prácticos para operar porque no disponían de simbolos particulares para los dígitos, usando para representarlos una acumulación del signo de la unidad y la decena. El hecho que sus bases fuese 60 y 20 respectivamente no hubiese representado en principio nigún obstáculo. Los mayas por su parte cometían una irregularidad a partir de las unidades de tercer orden, ya que detrás de las veintenas no usaban 20x20=400 sino 20x18=360 para adecuar los números al calendario, una de sus mayores preocupaciones culturales.

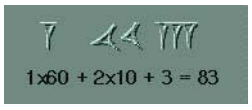
Fueron los indios antes del siglo VII los que idearon el sistema tal y como hoy lo conocemos, sin mas que un cambio en la forma en la que escribimos los nueve dígitos y el cero. Aunque con frecuencia nos referimos a nuestro sistema de numeración cómo árabe, las pruebas arqueológicas y documentales demuestran el uso del cero tanto en posiciones intermedias como finales en la India desde el sss. Los árabes transmitieron esta forma de representar los números y sobre todo el cáculo asociado a ellas, aunque tardaron siglos en ser usadas y aceptadas. Una vez más se produjo una gran resistencia a algo por el mero hecho de ser nuevo o ajeno, aunque sus ventajas eran evidentes. Sin esta forma eficaz de numerar y efectuar cálculos dificilmente la ciencia hubiese podido avanzar.

8. El Sistema de Numeración Babilónico

Entre la muchas civilizaciones que florecieron en la antigua Mesopotamia se desarrollaron distintos sistemas de numeración. En el ssss A.C. se inventó un sistema de base 10, aditivo hasta el 60 y posicional para números superiores.



Para la unidad se usaba la marca vertical que se hacía con el punzón en forma de cuña. Se ponían tantos como fuera preciso hasta llegar a 10, que tenía su propio signo.

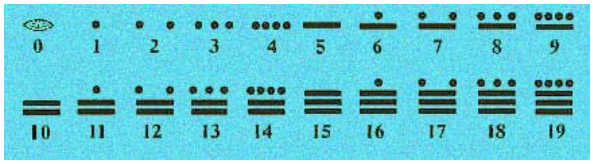


De este se usaban los que fuera necesario completando con las unidades hasta llegar a 60.

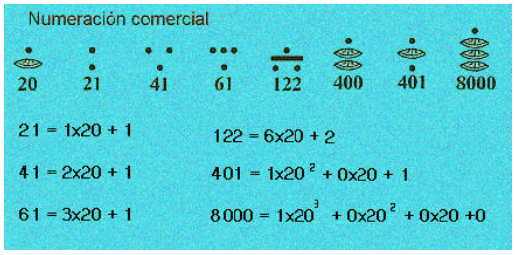
A partir de ahí se usaba un sistema posicional en el que los grupos de signos iban representando sucesivamente el número de unidades, 60, 60x60, 60x60x60 y asi sucesivamente como en los ejemplos que se acompañan.

El Sistema de Numeración Maya

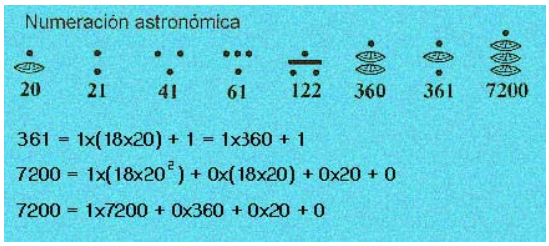
Los mayas idearon un sistema de base 20 con el 5 cómo base auxiliar. La unidad se representaba por un punto. Dos, tres, y cuatro puntos servían para 2, 3 y 4. El 5 era una raya horizontal, a la que seañadían los puntos necesarios para representar 6, 7, 8 y 9. Para el 10 se usaban dos rayas, y de la misma forma se continúa hasta el 20, con cuatro rayas.



Hasta aquí parece ser un sistema de base 5 aditivo, pero en realidad, considerados cada uno un solo signo, estos símbolos constituyen las cífras de un sistema de base 20, en el que hay que multiplicar el valor de cada cifra por 1, 20, 20x20, 20x20x20 ... según el lugar que ocupe, y sumar el resultado. Es por tanto un sistema posicional que se escribe a arriba abajo, empezando por el orden de magnitud mayor.



Al tener cada cifra un valor relativo según el lugar que ocupa, la presencia de un signo para el cero, con el que indicar la ausencia de unidades de algún orden, se hace imprescindible y los mayas lo usaron, aunque no parece haberles interesado el concepto de cantidad nula. Cómo los babilonios lo usaron simplemente para indicar la ausencia de otro número.



Pero los científicos mayas eran a la vez sacerdotes ocupados en la observación astronómica y para expresar los número correspondientes a las fechas usaron unas unidades de tercer orden irregulares para la base 20. Así la cifra que ocupaba el tercer lugar desde abajo se multiplicaba por 20x18=360 para completar una cifra muy próxima a la duración de un año.

El año lo consideraban dividido en 18 uinal que constaba cada uno de 20 días. Se añadían algunos festivos (uayeb) y de esta forma se conseguía que durara justo lo que una de las unidades de tercer orden del sistema numérico. Además de éste calendario solar, usaron otro de carater religioso en el que el año se divide en 20 ciclos de 13 días.

Al romperse la unidad del sistema éste se hace poco práctico para el cálculo y aunque los conocimiento astronómicos y de otro tipo fueron notables los mayas no desarrollaron una matemática más allá del calendario.