CAPÍTULO 1: Almacenamiento de datos en el computador

Una de las primeras cosas que se debe tener en cuenta para aprender a programar un computador es la forma cómo se almacena la información en la computadora. Cuando una persona ajena al mundo de la computación se pone delante de la máquina, ve un apantalla que contiene muchas imágenes, íconos, etc., y que por medio de una serie de dispositivos como el teclado, el mouse, o la misma pantalla, puede crear documentos, trazar una serie de gráficos, crear imágenes o hacer cálculos complejos. Aquellas personas, que simplemente usan el computador, no se dan cuenta lo que hay detrás de esa máquina, me refiero a la manera cómo ese texto, esa imagen o ese sonido puede estar almacenado y puede ser procesado por el computador de manera que podamos apreciarlos y sacarle el provecho que le sacamos cada día.

La programación de computadores implica la creación de aplicaciones que permita a las personas poder procesar información, esto es, crear aplicaciones para poder escribir texto, almacenarlos, recuperarlos y actualizarlos, poder trazar una gráfica, procesar imágenes, o hacer que se realicen cálculos complejos y reiterativos. Pues bien, antes de poder crear una aplicación lo primero que debemos conocer es cómo se almacena la información en el computador, también se tiene que entender que el computador es una máquina creada por el hombre y por lo tanto un elemento con muchas limitaciones, que un computador no piensa, que sólo se limita a repetir una serie de acciones y que esas acciones no son otra cosa que órdenes conglomeradas en lo que se conoce como un programa o una aplicación de computador.

En este capítulo se estudiarán en primer lugar los dispositivos que emplea el computador para almacenar la información, y en segundo lugar, donde nos explayaremos más, veremos cómo se almacena la información en la computadora, veremos cómo una máquina llena de cables y circuitos electrónicos puede guardar y procesar un número, un texto, una imagen, etc. y lo más importante, se las limitaciones que existen y cómo se hace para adaptarse a ellas.

Comencemos entonces analizando los medios que tiene le computador para almacenar información.

Memoria del computador

El dispositivo que permite almacenar información, vale decir, textos, números, imágenes, sonidos, en la computadora se denomina "memoria del computador". En un computador podemos distinguir dos tipos de memoria, la memoria principal y la secundaria.

Memoria Principal

La memoria principal del computador es el lugar donde el computador colocará la información que está procesando en ese momento. Si se quiere ejecutar un programa como una hoja de cálculo, un procesador de palabras, un manejador de imágenes, etc., el computador colocará primero el programa que se va a ejecutar en la memoria principal y lo ejecutará desde allí. Una vez que el programa esté en ejecución se podrá añadir información al computador, esto es, si estamos trabajando con un procesador de palabras se agregará el texto que se desea crear, este texto también es colocado en la memoria principal del computador.

La memoria principal tiene un problema, es "volátil", esto quiere decir que si el computador está desconectado la información que guardaba allí se pierde y no habrá forma de recuperarla. Es por esto que el computador requiere de otro dispositivo que permita almacenar información de modo que ésta no se pierda.

Memoria Secundaria

La memoria secundaria sirve para almacenar, de modo permanente, información procesada por el computador. Esto quiere decir que lo que se almacena en este dispositivo, no se pierde así el computador esté apagado, como sucede con el contenido de la memoria principal. El problema con la memoria secundaria es que la información que se guarda allí no se puede procesar, sólo se almacena; para poder procesar la información que se encuentra allí el computador tendrá que llevarla a la memoria principal.

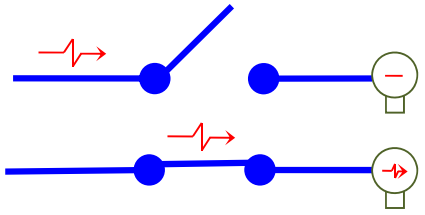
Esto último se refiere a que, una imagen (por ejemplo un fotografía), un directorio telefónico, un programa, etc., pueden ser almacenados en la memoria secundaria, sin embargo para que una imagen pueda ser vista en el monitor del computador o impresa en papel, ésta debe ser llevada a la memoria principal del computador. Si, dado el nombre de una persona deseamos obtener su número telefónico, el directorio, parcial o totalmente, debe ser llevado a la memoria principal para realizar la búsqueda. Si deseamos escribir un documento, primero debe ejecutarse el programa Word, para esto se debe llevar el programa a la memoria principal para su ejecución. Una vez escrito el documento, si se desea preservarlo para utilizarlo en otro momento, éste debe ser almacenado en la memoria secundaria.

Existen diferentes dispositivos que cumplen la función de memoria secundaria. Estos dispositivos son por ejemplo los discos duros, los discos compactos, las memorias flash o memorias USB, etc.

Codificación de la información para su almacenamiento

¿Cómo hace el computador para almacenar "un texto", "un número", "una fotografía", "un programa", etc.?, ¿Cómo hace para ejecutar un programa? Esta tarea no es sencilla y se ha requerido una solución muy compleja e ingeniosa.

Para entender mejor su funcionamiento, imaginemos un interruptor eléctrico como se muestra en la siguiente figura:



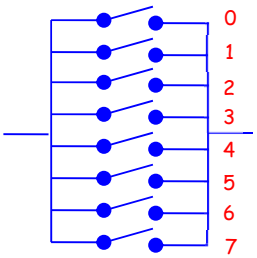
Este dispositivo puede estar colocado en dos estados, uno en el que la compuerta está abierta, por lo tanto no pasa la corriente y por lo tanto no es registrada por un detector y otro en el que la compuerta está cerrada y por lo tanto pasa la corriente y esta es detectada. Llamemos al primer caso "estado cero" y al segundo "estado uno".

Con este dispositivo se podría representar, al detectarse la corriente, de dos números, el cero (con el estado cero) y el uno (con estado uno), coincidentemente estos dos números o dígitos corresponden a los dígitos que se emplean para representar el sistema de numeración binario. A cada uno de estos estados se les denomina "bit" que deriva de la frase en inglés "binary digit".

En los computadores actuales, no existen estos interruptores eléctricos, en lugar de eso existen dispositivos electromagnéticos que permiten representar de estos dos estados y por lo tanto los dos dígitos binarios.

Siguiendo con la analogía, con un solo interruptor sólo se puede representar el 0 y el 1, ¿Qué pasa con los demás números p. e.: 23, 1623, 45.67, -12 -89.432, 0.000034? ¿Qué pasa con los textos, las imágenes, el sonido, los videos, etc.? ¿Cómo se almacenan o representan? La respuesta a estas preguntas requiere también soluciones complejas e ingeniosas.

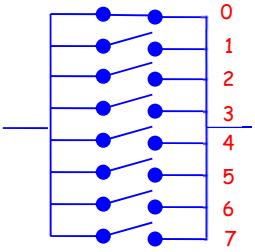
Si tuviéramos un dispositivo en el que se agruparan varios de estos interruptores, por ejemplo 8, se tendría un esquema como el que se muestra a continuación:



Cada interruptor podría encontrarse en un estado 0 o 1.

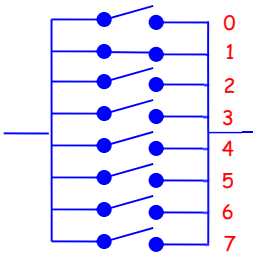
En la imagen anterior, todos están en 0, por lo tanto en conjunto el dispositivo pondrían estar representando el estado 0000 0000, o simplemente el estado 0.

Si se tuviera un esquema como el que se muestra a continuación, en el que el primer interruptor estuviera cerrado (estado 1) y los demás abiertos (estado 0):

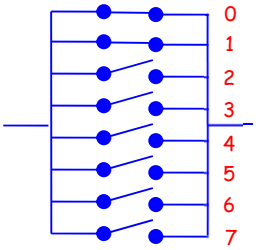


Se podría pretender representar el estado 0000 0001, o simplemente el estado 1.

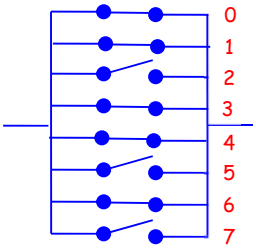
La siguiente figura representa alguno de los diferentes estados que podría adaptar el dispositivo:



Estado 0000 0010 o estado 2



Estado 0000 0011 o estado 3



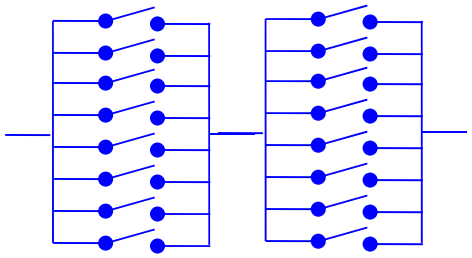
Estado 0101 1011 o estado 91

Como se puede apreciar en un dispositivo como este se pueden obtener 256 estados diferentes, por lo tanto se puede relacionar cada uno de estos estados con valores que van de 0 a 255. Dispositivos análogos a éste, conforman cada una de las celdas de memoria del computador.

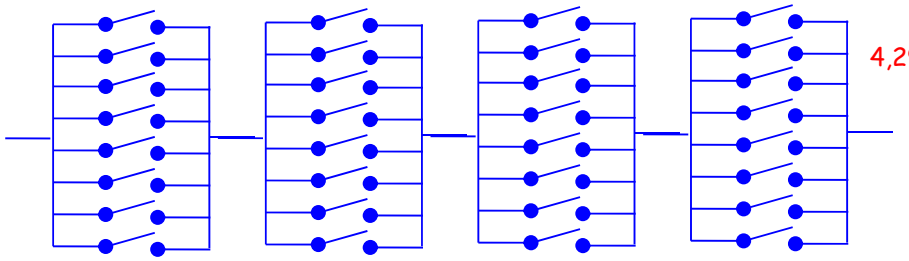
En términos generales se denomina "Byte" a una unidad de información compuesta por ocho bits.

Bueno, con este dispositivos podemos almacenar números que van de 0 a 255, ¿qué se hace para almacenar números de mayor tamaño?

La respuesta es simple, agrupando más de una celda de memoria, y luego tomando la representación del conjunto, así:



65,536 estados diferentes



4,294'967,296 estados diferentes

Con esto, prácticamente se está solucionando el problema de los números de gran tamaño, sin embargo se debe tomar en cuenta que los sistemas definidos en el computador no dan la libertad de representar cualquier número, los sistemas definen primero bajo qué esquema se representarán sus números, empleando un byte, dos bytes, cuatro bytes, etc., una vez definido esto, el usuario sólo podrá emplear un rango de números de acuerdo a ese esquema. Así, si el sistema define una representación de 2 bytes para sus números, no se podrá manejar por ejemplo el número 70,000.

Representación de números negativos

La pregunta que viene ahora es ¿qué pasa con los números negativos? Para solucionar este problema, se debe tener en cuenta, primero que el número se debe representar en un esquema de bytes, el signo "-" tiene que entrar en este esquema, y en segundo lugar, que operar dos números bajo esta representación debe ser una tarea extremadamente sencillas. Por último, si la operación da como respuesta un valor en el que se requieran más bits para representar el resultado, el valor resultante se almacenará sin el bit más significativo, éste se pierde o se acarrea según el sistema o el programa que se esté empleando.

Suma de bits:
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 Se acarrea 1

Suma de bytes:
0000 0000 + 0000 0001 = 0000 0001
0000 0001 + 0000 0001 = 0000 0010
1111 1111 + 0000 0001 = 0000 0000 Se acarrea 1

En este sentido, la representación de números negativos debe permitir realizar operaciones tan sencillas como la que se muestra en la tabla anterior.

A continuación se muestran tres maneras de representarlos, analizando la posibilidad de hacer con ellas operaciones sencillas.

Método 1: Empleo de un bit para representar el signo

Este método consiste en emplear un bit, de los ocho que tiene un byte, para representar el signo. Esto es 0 para números positivos y 1 para negativos. Los restantes bits se empalan para representar el número.

Lo primero que se debe apreciar es que en un byte ya no se podrán representar números que vayan de 0 a 255, sino que ahora sólo se podrá representar valores de 0 a 127, esto debido a que se cuenta con tan solo 7 bits, el octavo se emplea para el signo.

Bajo este esquema, el número 27 se representaría como 0001 1011 y -27 se representa como 1001 1011. El esquema de representación es sencillo, sin embargo al querer hacer operaciones con estos números se presentan estas situaciones:

```
 27 +      0001 1011 +
  9        0000 1001
----      -----------
 36        0010 0100    0010 0100 es la representación de 36, la operación es correcta.
```

```
 -27 +     1001 1011 +
  -9       1000 1001
----      ----------
 -36       0010 0100    El acarreo final se pierde.   0010 0100 representa 36 y no -36, la operación es incorrecta.
```

```
 27 +      0001 1011 +
 -9        1000 1001
----      ----------
 18        1010 0100    1010 0100 representa -36 y no 18.
```

Como se puede apreciar, no es tan sencillo sumar dos números en esta representación, se tendría que agregar condicionales para que la operación sea hecha en forma correcta, y esto no se puede aceptar.

Método 2: Complemento

Este método consiste en representar un número negativo como el complemento del número positivo, esto es 27 se representa como 0001 1011 y -27 como 1110 0100, observe que en el primer caso el bit más significativo es 0 y en el segundo caso es 1, aquí también se podrán representar números que van sólo de -128 a 127.

Veamos cómo se comporta esta representación cuando se desea hacer una suma:

27 +	0001 1011 +	
9	0000 1001	
----	----------	0010 0100 es la representación de 36, la
36	0010 0100	operación es correcta.
-27 +	1110 0100 +	
9	0000 1001	1110 1101 es el complemento de 0001 0010
----	----------	y esto representa 18, por lo tanto la
-18	1110 1101	operación es correcta.
27 +	0001 1011 +	
-9	1111 0110	
----	----------	0001 0001 representa 17 y no 18. La
18	0001 0001	operación es incorrecta
-27 +	1110 0100 +	
-9	1111 0110	1101 1010 es el complemento de 0010 0101
----	----------	y esto representa 37 y NO 36 por lo tanto la
-36	1101 1010	respuesta no es correcta.

Nos encontramos en un caso similar al anterior, la representación de los números es sencilla pero las operaciones con ellos no.

Método 3: Complemento a 2

Este método consiste en representar un número negativo como el complemento del número positivo pero a esta representación se le suma 1, esto es 27 se representa como 0001 1011 y -27 como 1110 0101 (a diferencia de la representación anterior que daba 1110 0100), observe que, de la misma forma, en el primer caso el bit más significativo es 0 y en el segundo caso es 1 y por lo tanto se podrán representar números que van de -128 a 127.

Veamos cómo se comporta esta representación cuando se desea hacer una suma:

27 +	0001 1011 +	
9	0000 1001	
----	----------	0010 0100 es la representación de 36, la
36	0010 0100	operación es correcta.
-27 +	1110 0101 +	
9	0000 1001	1110 1110 es el complemento a 2 de
----	----------	0001 0010 y esto representa 18, por lo
-18	1110 1110	tanto la operación es correcta.
27 +	0001 1011 +	
-9	1111 0111	
----	----------	0001 0010 representa 18, la operación es
18	0001 0010	correcta.
-27 +	1110 0101 +	
-9	1111 0111	1101 1100 es el complemento a 2 de
----	----------	0010 0100 y esto representa 36 por lo tanto
-36	1101 1100	la respuesta es correcta.

Esta es una representación sencilla en la que se pueden hacer operaciones sencillas sin que, como en los otros casos, tengan que introducir condicionales.

La representación de números enteros en complemento a 2 es un estándar por lo que todos los lenguajes de programación lo emplean.

Representación de números reales o de punto flotante

Una característica de los número reales es que si tomamos dos números cualesquiera que pueden estar muy juntos, se puede probar que existen infinitos números entre esos dos números. Así por ejemplo, si tomamos el 2 y el 3, podemos encontrar el 2.1, 2.2, 2.3,… 2.9, pero también el 2.01, 2.02,… 2.09, 2.11, 2.12,… y así sucesivamente.

Esta propiedad hace que para representar números reales se haya tenido que definir una forma mucho más compleja que para los números enteros. Más aun, los diferentes autores no han podido ponerse de acuerdo sobre el formato final que se les da a los números reales de modo que los diferentes lenguajes de programación emplean distintos formatos para el manejo de sus números reales, por ejemplo el Pascal emplea 6 bytes para representar un número real, mientras que C lo hace con 4, por esta razón la información que sale de un lenguajes no puede ser empleada con facilidad por otro lenguaje. Esto no se da con los enteros ya que allí si se emplea un estándar.

De lo que sí se puede hablar de común que existen entre los formatos de números reales es en el principio que se emplea para representarlos en memoria. El método utilizado se base en la "notación científica normalizada", esta notación se basa en que un número se multiplique o divida varias veces por la base en la que esté representado hasta que todos los dígitos menos uno aparezcan a la derecha del punto decimal, de modo que la parte entera del número sea mayor que cero.

Por ejemplo:

El número $157.381_{(10)}$ puede ser representado como 1.57381×10^{2}
El número $0.00007281_{(10)}$ puede ser representado como 7.281×10^{-5}
El número $-7800000000.0_{(10)}$ como -7.8×10^{9}
El número $-0.00000000000456_{(10)}$ como -4.56×10^{-15}
El número $10101.1101_{(2)}$ como 1.01011101×2^{4}
El número $0.00F3A_{(16)}$ como $F.3A \times 16^{-3}$
etc.

En términos generales se puede afirmar que un número cualquiera R expresado en una base b, se puede representar como:

$$R = m \times b^{e}$$

Donde "m" es un número cuya parte entera es de un solo dígito mayor que cero, "b" es la base del sistema y "e" es un número entero que denota cuántas veces se desplazó el punto decimal, a la derecha (+) o izquierda (-), para obtener el valor de "m" a partir de "R". En otras palabras el número está compuesto por una "mantisa multiplicada por una "base" elevada a una potencia entera denominada "exponente".

La ventaja de este método es que se pueden representar números muy grandes y también extremada mente pequeños como 1.3456×10^{23} ó 8.09234×10^{-56}.

Veamos ahora cómo esta notación científica puede ser empleada por los computadores para representar los números de punto flotante. Primero se decide cuantos bytes se van a utilizar en esta representación, por ejemplo Pascal emplea 6 bytes para sus datos de tipo real, por otro lado el lenguaje C emplea 4 bytes para representar el tipo float.

En el caso del lenguaje C, la representación de 4 bytes (32 bits), se distribuirían de la siguiente manera: un bit para el signo del número, 8 bits para representar el exponente, el resto (23) queda para la mantisa.

En esta representación no se emplea el complemento a dos para la mantisa por lo que el bit de signo indicará positivo con un cero (0) y negativo con un uno (1). Por otro lado debido a que el sistema binario (en base 2) sólo se emplea los dígitos 0 y 1, la parte entera de la mantisa siempre será 1, este dígito no se lleva a la representación en el computador como se verá más adelante.

Para manejar el exponente, tampoco se emplea la notación en complemento a dos debido a que complicaría las operaciones con los números de ponto flotante. Por esto debido, a que el exponente estaría en el rango de -126 á 127, éste se desplaza en 127, con lo que se obtienen valores entre 1 y 254. El exponente cero (0) se emplea para denotar el número 0 mientras que el valor 255 para denotar el valor infinito (∞).

Ejemplos:

1) El número 734.8267 de representaría de la siguiente manera:

Bit de signo: 0 (número positivo)

Mantisa: 734.8267 $_{(10)}$ = 1011011110.11010011101000101001110001 1… $_{(2)}$

Mantisa normalizada: $1.01101111011010011101000100111000111110 01\ldots \times 2^{9}$

Exponente: 9, normalizado = $136_{(10)}$ (9+127) ⇨ $10001000_{(2)}$

Por lo tanto el número quedará representado como:

Signo	Exponente	Mantisa
0	10001000	01101111 01101001 1101001

Separado en bytes tendíamos:

01000100	00110111	10110100	11101001

o, en hexadecimal:

44	37	B4	E9

2) El número -0.000008267 de representaría de la siguiente manera:

Bit de signo: 1 (número negativo)

Mantisa: 0.000008267 $_{(10)}$ =
0.00000000000000001000101010110010011111110100000011101

Mantisa normalizada: $1.00010101011001001111111010000001 1101\ldots \times 2^{-17}$

Exponente: -17, normalizado = $110_{(10)}$ (-17+127) ⇨ 01101110 $_{(2)}$

Por lo tanto el número quedará representado como:

Signo	Exponente	Mantisa
1	01101110	00010101 01100100 1111111

Separado en bytes tendíamos:

10110111	00001010	10110010	01111111

o, en hexadecimal:

B7	0A	B2	7F

En el caso del lenguaje Pascal, los números de punto flotante se representan en 6 bytes (48 bits), los cuales se distribuyen de la siguiente manera: primero va el exponente de 8 bits para, le sigue un bit para el signo del número y el resto (39 bits) para la mantisa.

Nótese que la primera diferencia con respecto al C, sin contar el número de bits empleados, es la posición del signo, en C va primero mientras que en Pascal va después del exponente.

En esta representación tampoco se emplea el complemento a dos para la mantisa por lo que el bit de signo indicará positivo con un cero (0) y negativo con un uno (1).

Para la mantisa se emplea otro estándar, en éste, la notación se basa en que un número se multiplique o divida varias veces por la base en la que esté representado hasta que todos los dígitos aparezcan a la derecha del punto decimal, esto es, el número $157.381_{(10)}$ se representa como 0.157381×10^3 a diferencia del otro método en el que se obtiene 1.57381×10^2. La mantisa se toma a partir del punto decimal.
Finalmente, en la representación del número se toma a partir del segundo dígito, esto porque el primer dígito siempre será 1.

Para manejar el exponente, a éste se le suma 128, en lugar de 127 como se hace en C.

Ejemplos:

1) El número 734.8267 de representaría de la siguiente manera:

Mantisa: $734.8267_{(10)} = 1011011110.11010011101000101001110001111 0\ldots_{(2)}$

Mantisa normalizada: $0.10110111101101001110100010100111000111 10\ldots \times 2^{10}$

Exponente: 10, normalizado = $138_{(10)}$ (10+128) ⇨ $10001010_{(2)}$

Bit de signo: 0 (número positivo)

Por lo tanto el número quedará representado como:

Exponente	Signo	Mantisa
10001010	0	01101111 01101001 11010001 01001110 0011110

Separado en bytes tendíamos:

10001010	00110111	10110100	11101000	10100111	00011110

o, en hexadecimal:

8A	37	B4	E8	A7	1E

2) El número -0.000008267 de representaría de la siguiente manera:

Mantisa: $0.000008267_{(10)} =$

$0.00000000000000001000101010110010011111101010000001111000$

Mantisa normalizada: $0.10001010101100100111111010100000001111000\ldots \times 2\text{-}16$

Exponente: -16, normalizado = $112_{(10)}$ (-16+120) ⇨ $01110000_{(2)}$

Bit de signo: 1 (número negativo)

Por lo tanto el número quedará representado como:

Exponente	Signo	Mantisa
01110000	1	00010101 01100100 11111101 01000000 1111000

Separado en bytes tendíamos:

01110000	10001010	10110010	01111110	10100000	01111000

o, en hexadecimal:

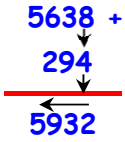
70	8A	B2	7E	A0	78

Almacenamiento invertido o "back-words"

Como hemos podido apreciar en detalle hasta este momento, como se representan los valores numéricos en el computador, hemos apreciado que se requiere muchas veces más de un byte para poder representarlos. Sin embargo nos falta un concepto que es muy importante cuando se estudia la representación interna de datos, este concepto se denomina "almacenamiento invertido de palabras o back-words.

Cuando nosotros visualizamos un número éste lo leemos de izquierda a derecha, desde la cifra más significativa, esto es por ejemplo el número setecientos cuarenta y nueve (749) la cifra de mayor valor (la más significativa), el 7, se encuentra a la izquierda del número, mientras que la de menos valor, el 9, se encuentra a la derecha.

Esto no tiene nada de particular puesto que así estamos acostumbrados a representarlos, sin embargo, podemos darnos cuenta que ese no es el orden en que operaríamos dos números, las operaciones se hacen de derecha a izquierda, empezando de la cifra menos significativa, esto es:



Para en computador, leer un número en un sentido y luego operarlo en el otro resulta una tarea muy complicada, por esta razón en el computador los números se almacenan empezando de la parte menos significativa hasta la más significativa. Esto quiere decir que, por ejemplo, si el computador trabajara en base 10 el número setecientos cuarenta y nueve se almacenaría como 947 (y se leería nueve cuarenta setecientos) a esta forma de almacenamiento o manejo de los números se denomina back-words. De ese modo se operaría el número en el mismo sentido al que se lee. Por ejemplo:

El número entero 8049653 representado en binario empleando 4 bytes como 00000000 01111010 11010011 11110101 ($00\ 7A\ D3\ F5_{(16)}$) se almacena en la memoria del computador como 11110101 11010011 01111010 00000000 ($F5\ D3\ 7A\ 00_{(16)}$), fíjese que la representación inversa es a nivel de bytes y no de bits.

En el caso de los números reales, en los ejemplos se tenía:

Para el número 734.8267 se obtenía para el lenguaje C, 44 37 B4 E9, en memoria se almacenaría como E9 B4 37 44. El mismo número en Pascal se obtenía 8A 37 B4 E8 A7 1E su representación en memoria será 8A 1E A7 E8 B4 37, obsérvese que en el caso del lenguaje C todo el número se escribe al revés, sin embargo en el Pascal, sólo se invierte la mantisa del número.

De igual manera, el número -0.000008267 que en C se representaría como B7 0A B2 7F en memoria se almacenaría como 7F B2 0A B7, y en Pascal su representación sería 70 8A B2 7E A0 78 y en memoria se almacena 70 78 A0 7E B2 8A.

Representación de caracteres

El computador sólo puede representar códigos binarios, lo hemos visto con los números. El caso de los caracteres no es una excepción, para almacenar en la memoria del computador un caracter, como la letras A, el símbolo #, etc., se requiere codificar el caracter en un formato binario.

A lo largo del tiempo se han venido utilizando diferentes sistemas de codificación, se empezó con el sistema BCDIC o Binary Coded Decimal Interchange Code (código binario de intercambio decimal), en él se agruparon los 64 caracteres más comunes y a cada uno se le relacionó con un número entero de 0 a 63, el valor binario del número corresponde al código del caracter. Como se puede deducir este sistema se representaba en 6 bits.

Por los años setenta se termina de implementar otro sistema de codificación denominado ASCII o American Standard Code for Information Interchange este sistema empezó empleando 7 bits para representar 128 caracteres que incluían las letras mayúsculas y minúsculas del alfabeto inglés así como caracteres de puntuación y de control. Sin embargo, con el paso del tiempo se amplió el código ASCII a 8 bits para poder incluir caracteres internacionales como la Ñ, á, ü, æ, ç etc. El código ASCII extendido, o por costumbre simplemente ASCII, se popularizó muchísimo y hasta hoy se viene empleando en forma masiva en todo el mundo, sin embargo se debe tener en cuenta que si bien los 128 primeros caracteres son siempre los mismos, el segundo grupo puede variar de acuerdo a la localidad donde se emplee. El juego de caracteres ASCII más común, se muestra en la tabla que se presenta más adelante. (Norton 1987: 450).

Nótese que, en la tabla, cada caracter viene acompañado de un código numérico (dado como un número decimal y como un número hexadecimal). Es este código y no el caracter es el que se almacena en la memoria del computador como cualquier valor entero, así por ejemplo un la letra A se almacena en la memoria como el número 65 codificado en binario.

Usted se preguntará entonces cómo se hace para diferenciar un caracter de un número entero almacenado en la memoria, la respuesta en es que no hay forma, serán las funciones de los lenguajes de programación las que harán la diferencia. Por ejemplo en el lenguaje C, a una variable entera asignada con el valor de 65 puede mostrarse en la pantalla como el número 65, como el número 41h o como el caracter A y esto sólo depende de la manera cómo se llame a la función de salida. Esto es:

```
x=65;
printf("%d", x); →→→→→→  muestra el número decimal 65
printf("%x", x); →→→→→→  muestra el número hexadecimal 41
printf("%c", x); →→→→→→  muestra el caracter A
```

Observe también que en la tabla los caracteres vienen agrupados, esto es las letras mayúsculas vienen juntas desde la posición 65 a la 90, igual pasa con las letras minúsculas y los dígitos. Note también que entre el caracter "A" y el caracter "a" existen 32 posiciones. Esto se ha hecho para poder manipular los caracteres en forma sencilla, si se tiene almacenado el código de la letra H (72) fácilmente se puede obtener el caracter I, para esto sólo hay que sumarle uno al código de la letra H, por otro lado si le sumamos 32 al código de la letra H obtenemos el caracter h.

CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER	CÓDIGO DEC	CÓDIGO HEX	CARACTER
0	00		32	20		64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
2	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
5	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
8	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	♀	44	2C		76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	░	208	D0	╨	240	F0	≡
17	11	◄	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	∟	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	Pts	190	BE	╛	222	DE	▐	254	FE	■
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

TABLA DE CARACTERES ASCII

Como se dijo anteriormente, el código ASCII tiene un problema, este es que los caracteres de la segunda mitad de la tabla no son siempre los mismos lo que hace muy difícil su transporte a otras localidades. Por los años 90 se propuso un nuevo sistema de codificación denominado **Unicode** en los que cada caracter es codificado en dos bytes (16 bits). Este sistema permite la representación de 65535 caracteres con lo que con esto queda prácticamente cubierto todos los símbolos del planeta. Unicode se ha convertido en un sistema de codificación cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático.

> Unicode cubre la mayor parte de las escrituras usadas actualmente, incluyendo: Árabe, Armenio, Bengalí, Braille, Sílabas aborígenes canadienses, Cheroqui, Copto, Cirílico, Devanāgarī, Esperanto, Ge'ez, Georgiano, Griego, Guyaratí, Gurmukhi, Hangul (Coreano), Han (Kanji, Hanja y Hanzi), Japonés (Kanji, Hiragana y Katakana), Hebreo, Jémer (Camboyano), Kannada (Canarés), Lao, Latino, Malayalam, Mongol, Burmese, Oriya, Syriac, Tailandés (Thai), Tamil, Tibetano, Yi, Zhuyin (Bopomofo).
>
> Unicode ha ido añadiendo escrituras y cubrirá aún más, incluyendo escrituras históricas menos utilizadas, incluso aquellas extinguidas, para propósitos académicos: Cuneiforme, Griego antiguo, Linear B, Fenicio, Rúnico, Sumerio, Ugarítico (Wikipedia 2010 UNICODE).

Representación de cadenas de caracteres

La representación de cadenas de caracteres no está estandarizada, esto quiere decir que los diferentes lenguajes de programación no manejan las cadenas de caracteres de la misma manera, a pesar que el manejo de cadenas se basa en la aglomeración de caracteres, los cuales como hemos visto si están estandarizados.

En Pascal por ejemplo las cadenas de caracteres se basan en un tipo de datos denominado **string**. Este tipo de datos define las cadenas como un arreglo de 256 caracteres, cuyos índices van de 0 a 255. A partir de la posición 1 se colocan los caracteres que se quiere almacenar en la cadena, en la posición 0 se almacena un caracter cuya representación numérica corresponde a la cantidad de caracteres que se almacenó en la cadena. Por ejemplo:

Si se quiere almacenar la siguiente cadena de caracteres:

'Rodolfo Alexander López Rojas'

Esto se haría de la siguiente manera:

↔	R	o	d	o	l	f	o		A	l	e	x	a	n	d	e	r		L	ó	p	e	z		R	o	j	a	s	…	
0	1	2	3	4	5	6	7	8	9	01	1	2	3	4	5	6	7	8	9	02	1	2	3	4	5	6	7	8	9	03	

Nótese que en la posición 0 del arreglo se coloca el caracter '↔' que corresponde al código 29 en la tabla ASCII.

Cualquier operación que se realice sobre la cadena afectará en forma automática la posición cero del arreglo de modo que en cada momento se tenga la cantidad de caracteres válidos almacenados.

Es bueno notar que debido a que la longitud de la cadena se denota por un solo caracter es imposible que esta representación pueda almacenar más de 255 caracteres.

Otros lenguajes de programación manejan sus cadenas de caracteres de manera muy diferente. Por ejemplo el lenguaje C, la representación de cadenas se hace por medio de arreglos, pero también por medio de direcciones de memoria (punteros) a espacios de memoria gestados dinámicamente (en tiempo de ejecución de un programa). Aquí los caracteres son colocados a la manera de un arreglo pero no se coloca la longitud en el inicio, más bien se coloca un caracter especial de terminación al final de los caracteres. En este caso se coloca el caracter cuyo código ASCII es cero. Todo trabajo efectuado sobre la cadena se empieza desde el inicio de la misma, hasta encontrar el caracter de terminación. Por ejemplo la cadena:

'Rodolfo Alexander López Rojas'

Se almacenaría de la siguiente manera:

R	o	d	o	l	f	o		A	l	e	x	a	n	d	e	r		L	ó	p	e	z		R	o	j	a	s	'\0'	…	
0	1	2	3	4	5	6	7	8	9	01	1	2	3	4	5	6	7	8	9	02	1	2	3	4	5	6	7	8	9	03	

'\0' representa al caracter cuyo código ASCII es cero

Esto permite que las cadenas no tengan límite de tamaño, lo que puede ser una gran ventaja en algunos casos.

Otras representaciones más complejas

Veamos ahora cómo se representan elementos más complejos en el computador, por ejemplo una imagen.

Una figura, una fotografía, un dibujo, etc., para que pueda almacenarse en el computador tendrán que de alguna manera llevarse a una representación basada en ceros y unos.

Si cogemos una fotografía y la observamos a través de un lente de aumento podemos observar que ésta no es impresa de manera continua como cuando se coge un pincel y se hace un trazo con él. Lo que observamos es que la fotografía está compuesta por una serie de puntos muy pequeños colocados lo suficientemente juntos como para que, vista la imagen a una distancia prudencial, parezca que esté formada por trazos continuos.

Si observamos un monitor de computador o televisión podemos observar el mismo esquema. Las imágenes se descomponen en pequeños elementos a los que se le asignan un color determinado, el conjunto visto desde una distancia prudencial forma una imagen.

Precisamente del inglés "picture element" deriva el nombre píxel, conocido como el elemento más pequeño en que se descompone una imagen.

La pregunta ahora es saber cómo representamos un píxel en términos de ceros y unos. Empecemos imaginándonos una figura en blanco y negro, para representar cada píxel sólo necesitaremos dos valores, uno para representar el blanco y otro para el negro, podemos asignar al blanco el valor de cero y uno al negro. De esta forma sólo necesitamos un bit para representar un píxel y un byte puede almacenar la representación de 8 píxeles. A una imagen que tuviera por ejemplo 4 colores, por ejemplo azul, verde, rojo y amarillo, podríamos asignarle cero al azul, uno al verde, dos al rojo y tres al amarillo. Aquí podemos observar que requerimos dos cifras binarias para representar cada color (azul = 00, verde = 01, rojo = 10, amarillo = 11), por lo tanto dos bits por cada píxel.

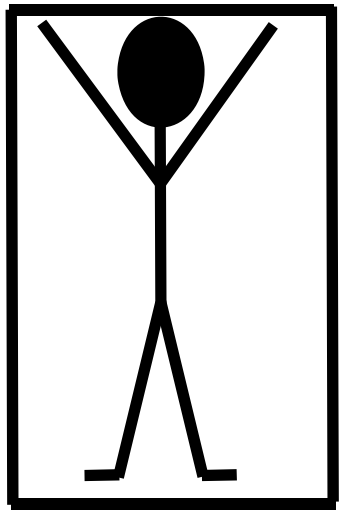
Si quisiéramos almacenar una imagen que tenga millones de colores quizá requiramos cuatro o más "bytes" para representar cada píxel.

En el ejemplo de la página siguiente vemos dos figuras y a su lado la misma imagen pero descompuesta en los valores que tendría cada píxel.

En el primer caso la figura es en blanco y negro, en la otra empleamos hasta 8 colores diferentes. Mientras que en la primera se requiere sólo un digito binario para representar cada píxel en la segunda se requieren tres dígitos binarios por cada píxel.

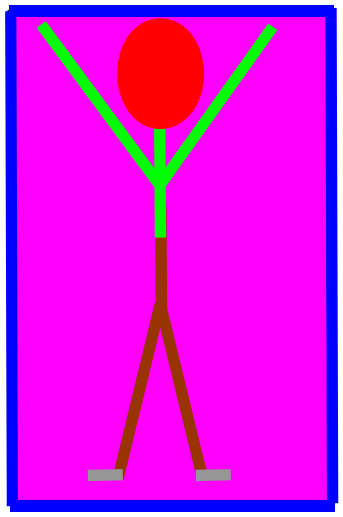
Ya tenemos definido la manera que se codifica cada píxel, sin embargo aun no podemos decir que el problema está solucionado, no es suficiente con esta información. Un programa que pueda manipular imágenes debe ser capaz de procesar diferentes tipos de imágenes, de lo contrario se tendría que tener un programa para leer imágenes en blanco y negro, otro para imágenes con cuatro colores, otro para imágenes de 8 colores y así sucesivamente. Por otro lado se debería tener un programa por cada tamaño de imagen que se quisiera procesar (combinado con la cantidad de colores). Se podría pensar en miles de programas. Esto es ilógico, un solo programa debe ser capaz de procesar cualquier imagen.

Es por esto que el formato que se emplee para almacenar una imagen en el computador debe incluir de alguna manera la cantidad de colores o el número de bits que se requieren para representar un píxel y el número de píxeles que tiene cada línea y el número de líneas que tiene la imagen. En este sentido se podría almacenar en el primer byte la cantidad de bits que se requieren para representar cada píxel, en los dos bytes siguientes se puede almacenar un número entero sin signo con la cantidad de píxeles por fila que tiene la imagen y en los dos bytes siguiente, con el mismo formato, la cantidad de filas de la imagen. Con esa información un programa podría reconstruir la imagen.



Valor de cada píxel: 1 píxel = 1 bit

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,1
1,0,0,1,0,0,1,1,1,1,1,0,0,1,0,0,0,1
1,0,0,0,1,0,1,1,1,1,1,0,1,0,0,0,0,1
1,0,0,0,0,1,0,1,1,1,0,1,0,0,0,0,0,1
1,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,1
1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1
1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1
1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1
1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1
1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1



Valor decimal de cada píxel.

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,3,2,3,3,3,3,4,4,4,3,3,3,3,2,3,3,1
1,3,3,2,3,3,4,4,4,4,4,3,3,2,3,3,3,1
1,3,3,3,2,3,4,4,4,4,4,3,2,3,3,3,3,1
1,3,3,3,3,2,3,4,4,4,4,3,2,3,3,3,3,1
1,3,3,3,3,3,2,3,4,3,2,3,3,3,3,3,3,1
1,3,3,3,3,3,3,2,2,2,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,2,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,2,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,2,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,6,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,6,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,3,6,3,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,6,3,6,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,6,3,6,3,3,3,3,3,3,3,1
1,3,3,3,3,3,3,6,3,6,3,3,3,3,3,3,3,1
1,3,3,3,3,3,6,3,3,3,6,3,3,3,3,3,3,1
1,3,3,3,3,3,6,3,3,3,6,3,3,3,3,3,3,1
1,3,3,3,3,3,6,3,3,3,6,3,3,3,3,3,3,1
1,3,3,3,7,7,7,3,3,3,7,7,7,3,3,3,3,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

Valor de cada píxel: 1 píxel = 3 bits

001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001
001,011,010,011,011,011,011,100,100,100,011,011,011,011,010,011,011,001
001,011,011,010,011,011,100,100,100,100,100,011,011,010,011,011,011,001
001,011,011,011,010,011,100,100,100,100,100,011,010,011,011,011,011,001
001,011,011,011,011,010,011,100,100,100,011,010,011,011,011,011,011,001
001,011,011,011,011,011,010,011,100,011,010,011,011,011,011,011,011,001
001,011,011,011,011,011,011,010,010,010,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,010,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,010,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,010,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,110,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,110,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,110,011,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,011,110,011,110,011,011,011,011,011,011,001
001,011,011,011,011,011,011,110,011,110,011,011,011,011,011,011,011,001
001,011,011,011,011,011,011,110,011,110,011,011,011,011,011,011,011,001
001,011,011,011,011,011,110,011,011,011,110,011,011,011,011,011,011,001
001,011,011,011,011,011,110,011,011,011,110,011,011,011,011,011,011,001
001,011,011,011,011,011,110,011,011,011,110,011,011,011,011,011,011,001
001,011,011,011,111,111,111,011,011,011,111,111,111,011,011,011,011,001
001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001,001

A continuación se muestra cómo quedaría la representación de cada figura.

Figura 1:

Byte 1	Bytes 2 y 3		Bytes 4 y 5		Byte 6	Byte 7	Byte 8	Byte 9	Byte 10
0000 0001	0000 0000	0001 0010	0000 0000	0001 0101	1111 1111	1111 1111	1110 1000	0111 0000	1001 1001
1 bit x píxel	18 píxeles x fila		21 filas		Valor de cada pixel				

Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20
0011 1110	0100 0110	0010 1111	1010 0001	1000 0101	1101 0000	0110 0000	1010 1000	0001 1000	0001 1100

Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30
0000 0110	0000 0010	0000 0001	1000 0000	1000 0000	0110 0000	0010 0000	0001 1000	0000 1000	0000 0110

Byte 31	Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39	Byte 40
0000 0010	0000 0001	1000 0000	1000 0000	0110 0000	0101 0000	0001 1000	0001 0100	0000 0110	0000 0101

Byte 41	Byte 42	Byte 43	Byte 44	Byte 45	Byte 46	Byte 47	Byte 48	Byte 49	Byte 50
0000 0001	1000 0010	0010 0000	0110 0000	1000 1000	0001 1000	0010 0010	0000 0110	0011 1000	1110 0001

Byte 51	Byte 52	Byte 53
1111 1111	1111 1111	1100 0000

Figura 2

Byte 1	Bytes 2 y 3		Bytes 4 y 5		Byte 6	Byte 7	Byte 8	Byte 9	Byte 10
0000 0011	0000 0000	0001 0010	0000 0000	0001 0101	0010 0100	1001 0010	0100 1001	0010 0100	1001 0010
3 bits x píxel	18 píxeles x fila		21 filas		Valor de cada pixel				

Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20
0100 1001	0010 0100	1011 0100	1101 1011	0111 0010	0100 0110	1101 1011	0100 1101	1001 0010	1101 1010

Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30
0110 1110	0100 1001	0010 0011	0110 1001	1011 0110	0100 1011	0110 1101	0011 1001	0010 0100	1000 1101

Byte 31	Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39	Byte 40
0011 0110	1101 1001	0010 1101	1011 0110	1001 1100	1001 0001	1010 0110	1101 1011	0110 0100	1011 0110

Byte 41	Byte 42	Byte 43	Byte 44	Byte 45	Byte 46	Byte 47	Byte 48	Byte 49	Byte 50
1101 1011	0100 1110	0011 0100	1101 1011	0110 1101	1001 0010	1101 1011	0110 1101	1010 0100	1001 1011

Byte51	Byte 52	Byte 53	Byte 54	Byte 55	Byte 56	Byte 57	Byte 58	Byte 59	Byte 60
0110 1101	1011 0110	0100 1011	0110 1101	1011 0110	1101 0011	0110 1101	1011 0110	1101 1001	0010 1101

Byte 61	Byte 62	Byte 63	Byte 64	Byte 65	Byte 66	Byte 67	Byte 68	Byte 69	Byte 70
1011 0110	1101 1011	0100 1101	1011 0110	1101 1011	0110 0100	1011 0110	1101 1011	0110 1101	0011 0110

Byte 71	Byte 72	Byte 73	Byte 74	Byte 75	Byte 76	Byte 77	Byte 78	Byte 79	Byte 80
1101 1011	0110 1101	1001 0010	1101 1011	0110 1101	1011 1100	1101 1011	0110 1101	1011 0110	0100 101

Byte 81	Byte 82	Byte 83	Byte 84	Byte 85	Byte 86	Byte 87	Byte 88	Byte 89	Byte 90
1011 0110	1101 1011	0111 1001	1011 0110	1101 1011	0110 1100	1001 0110	1101 1011	0110 1101	1110 0110

Byte 91	Byte 92	Byte 93	Byte 94	Byte 95	Byte 96	Byte 97	Byte968	Byte 99	Byte 90
1101 1011	0110 1101	1011 0010	0101 1011	0110 1101	1011 1100	1111 0011	0110 1101	1011 0110	1100 1001

Byte 101	Byte 102	Byte 103	Byte 104	Byte 105	Byte 106	Byte 107	Byte 108	Byte 109	Byte 110
0110 1101	1011 0110	1111 0011	1100 1101	1011 0110	1101 1011	0010 0101	1011 0110	1101 1011	1100 1111

Byte 111	Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119	Byte 120
0011 0110	1101 1011	0110 1100	1001 0110	1101 1011	0111 1001	1011 0111	1001 1011	0110 1101	1011 0010

Byte 121	Byte 122	Byte 123	Byte 124	Byte 125	Byte 126	Byte 127	Byte 128	Byte 129	Byte 130
0101 1011	0110 1101	1110 0110	1101 1110	0110 1101	1011 0110	1100 1001	0110 1101	1011 0111	1001 1011

Byte 131	Byte 132	Byte 133	Byte 134	Byte 135	Byte 136	Byte 137	Byte 138	Byte 139	Byte 140
0111 1001	1011 0110	1101 1011	0010 0101	1011 0111	1111 1111	0110 1101	1111 1111	1101 1011	0110 1100

Byte 141	Byte 142	Byte 143	Byte 144	Byte 145	Byte 146	Byte 147
1001 0010	0100 1001	0010 0100	1001 0010	0100 1001	0010 0100	1001 0010

Observe que la segunda imagen sólo tiene 8 colores, y es una figura pequeña; una imagen que ocupe toda la pantalla del computador en una resolución baja se maneja con 800x600 píxeles empleando millones de colores, imagínese el tamaño de esos archivos si se almacenara en un formato como el que describimos anteriormente.

El formato empleado es muy simple pero no es óptimo, en el mercado existen muchos formatos para almacenar imágenes, algunos se basan en algoritmos que permiten compactar las imágenes de modo que ocupen menos espacio en memoria. Entre los formatos conocidos podemos mencionar BMP, JPEG, GIF, PDF, PCX, TIFF etc.

Por último, imagínese usted cómo se puede almacenar en un computador un video, o sonidos, pues al igual que en los otros casos analizados, se tiene que dividir cada medio en elementos simples y estos llevarlos a representaciones binarias, no es una tarea fácil, pero si factible.