ARQUITECTURA DE COMPUTADORAS
UNIDAD 1

SISTEMAS DE NUMERACIÓN

INTRODUCCIÓN

Los **números naturales** aparecen al contar los objetos de un conjunto. En la primera infancia se empieza por aprender a contar, luego más tarde se coordinan conjuntos prescindiendo del orden. Lo mencionado parece justificar la introducción del número natural mediante la **formulación axiomática** que revele en qué consiste la operación de contar.

Con posterioridad a la invención del número natural surge la necesidad de ampliar el concepto de número. Surge así el número entero, el racional, el irracional y el número real.

Aceptando la existencia del número podemos definir un **Sistema de Numeración** como el conjunto de reglas y convenios que permiten la representación de todos los números mediante varios signos, o varias palabras.

Existen Sistemas de Numeración muy conocidos como el **Romano** que descompone el número en suma o diferencia de otros varios, cada uno de los cuales está representado por un símbolo especial:

I, V, X, L, C, D, M

Otro sistema es el **Decimal** que en vez de introducir nuevos símbolos para estos diversos sumandos, utiliza el principio del *valor relativo* (**posicional**), es decir una misma cifra representa valores distintos según el lugar que ocupa. Estos de sistemas son los que representan interés matemático. El Sistema Decimal que fue inventado en la India en el siglo IV antes de Cristo y llevado a Europa por los árabes en la Edad Media, está fundado en el número fijo que llamamos **diez** (habiendo elegido este y no otro porque tenemos diez dedos en las manos).

Toda combinación de operaciones fundamentales efectuadas con números cualesquiera que da origen a un nuevo número, se llama **algoritmo de numeración**. Para los sistemas de numeración basados en el *valor relativo* (posicionales), el algoritmo de numeración consiste en un polinomio:

$$N = a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + ... + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + ... + a_{-k} b^{-k}$$

Donde:

N : Número

a_i : número natural menor que b (símbolo)

b : base del sistema (cantidad de símbolos diferentes del sistema, es un número natural mayor que 1)

n + 1: cantidad de cifras enteras.

k : cantidad de cifras fraccionarias.

Obsérvese que la parte entera del número se corresponde a los términos del polinomio que tienen la base b elevada a exponentes cero o positivos mientras que la parte fraccionaria se corresponde con los exponentes negativos.

Puede apreciarse el principio de valor relativo, por ejemplo el número 24,2 en el conocido sistema de numeración decimal tiene el siguiente polinomio:

$$24,2 = 2 \times 10^{1} + 4 \times 10^{0} + 2 \times 10^{-1}$$

Obsérvese que 2 tiene un valor que depende de la posición que ocupa en el número.

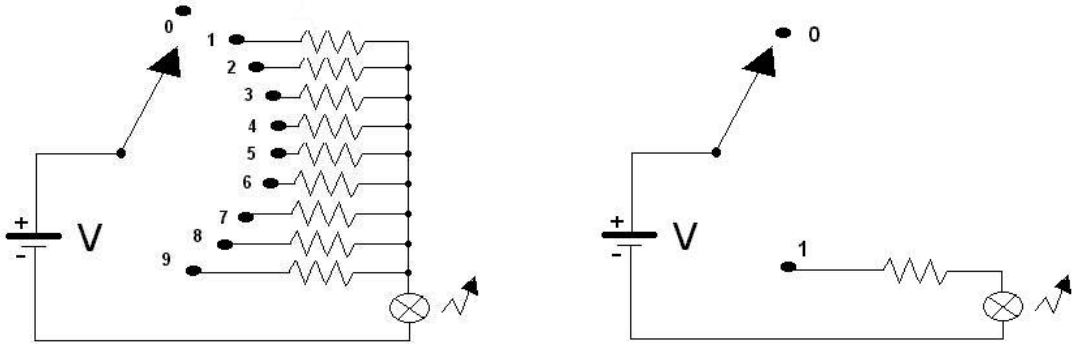
Las propiedades de los números, que se estudian en Matemáticas, son válidas cualquiera sea la base utilizada, siempre que se utilice el mismo algoritmo de numeración. Es decir que si elegimos otra base podríamos realizar operaciones algebraicas (suma, resta, multiplicación, división, etc.) sin ningún problema ya que la axiomática y teoremas parten del algoritmo de numeración. Como se dijo anteriormente la elección de 10 (diez) como base, fue que tenemos diez dedos en las manos, pero podemos hacernos esta pregunta:
¿Es el Sistema de Numeración Decimal el más adecuado para ser usado en sistemas físicos de representación y tratamiento de la información?
La respuesta resulta de consideraciones de costo y confiabilidad del *Sistema Físico* a construir.

Confiabilidad

Podemos decir que un sistema es más confiable cuanto más independiente sea de la **temperatura** en la que trabaja, el **envejecimiento** y la **dispersión** de los componentes que lo forman.

Los componentes que se utilizan en la construcción de los sistemas físicos de procesamiento de información, son eléctricos y electrónicos (transistores, diodos, resistores, capacitores, inductores, etc.), todos ellos cambian sus parámetros con el envejecimiento y la temperatura. Además es imposible construir dos componentes idénticos, a esto se lo llama dispersión. Los fabricantes especifican el *porcentaje de dispersión máximo* de los componentes que comercializan.

A fin de aclarar lo mencionado comparemos dos sistemas físicos (ver fig. 1) que representan números en distintas bases: *base 10 y base 2*:



a) b)

FIG. 1

La *fig.1a* indica un circuito eléctrico capaz de representar un número decimal (*base 10*), y la *fig.1b* indica un circuito eléctrico capaz de representar un número binario (*base 2*). Ambos consisten en un foco que se enciende con distintas intensidades luminosas al mover la llave (*diez en el caso a y dos en el caso b*), y suponen que un observador tiene que ser capaz de determinar el número en cuestión apreciando la intensidad luminosa.

Como primera conclusión resulta evidente que es más fácil determinar el estado del foco en caso b (prendido o apagado) que en el caso a (diez intensidades posibles).

Como segunda conclusión. Supongamos que aumenta la temperatura ambiente o que el circuito ha envejecido, esto modifica el valor de las resistencias y de la intensidad emitida por el foco. La temperatura y el envejecimiento afectan mucho más al caso a que al b.

Como tercera conclusión. Supongamos que tenemos que reponer el foco porque se quemó. El nuevo foco, debido a la dispersión, tendrá seguramente parámetros diferentes al anterior. (Mismas consideraciones si debiésemos reemplazar una resistencia). La dispersión afecta más al caso a que al b.

Como conclusión final puede decirse que el circuito que trabaja en binario es más confiable que el decimal. Podemos decir que es más confiable que cualquier circuito que trabaje en cualquier base.

Si bien los sistemas digitales no se construyen con focos y llaves, lo considerado es aplicable a los componentes que se utilizan en la realidad.

Costo

Nuevamente observando la fig.1 desde el punto de vista del costo podemos afirmar que es más costoso construir el circuito en decimal que en binario: **el *Costo* es proporcional a la base del sistema de numeración** utilizado, sin embargo necesitamos más circuitos binarios para representar una misma cantidad, (*por ejemplo para representar el número 25 necesitaríamos dos circuitos decimales como el de la fig.1a y cinco circuitos binarios como el de la fig.1b*). : **el *Costo* es proporcional a la cantidad de cifras**. Entonces podemos escribir:

$$Costo = k \,.\, b \,.\, (n+1)$$

Entre b y (n + 1) existe una relación:

$$b^{(n+1)} = M$$

Donde M : Módulo del Sistema (máximo número representable con *n+1* cifras)

$$(n+1).Ln\ b = Ln\ M$$

$$n + 1 = (Ln\ M / Ln\ b)$$

reemplazando

$$Costo = k\ Ln\ M\ \ (b / Ln\ b)$$

Haciendo la derivada respecto a la base (*dCosto/db= 0*) e igualando a cero obtenemos:

$$b_{óptima} = e$$

Concluimos que la base óptima, desde el punto de vista del costo, está entre 2 y 3. En realidad,la expresión del costo planteada es aproximada y teniendo en cuenta la confiabilidad, es el **Sistema Binario** el que se utiliza en la construcción de los Sistemas Digitales.

Sistema de Numeración Binario

En el Sistema Binario los símbolos utilizados son el 0 y el 1, llamados dígitos binarios (bits o bitios). Es posible, aplicando el algoritmo de numeración, obtener los números binarios correspondientes a las primeras 16_{10} (el subíndice 10 indica que el número esta en Sistema Decimal) cantidades. Igualmente podríamos obtener los números en el Sistema Octal y en el Sistema Hexadecimal. La fig.2 indica las correspondencias.

DECIMAL $b = 10_{10}$	BINARIO $b = 2_{10}$	OCTAL $b = 8_{10}$	HEXADECIMAL $b = 16_{10}$
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Obsérvese que en el Sistema Hexadecimal de han agregado las letras A, B, C, D, E y F para obtener los 16 símbolos distintos necesarios.

La razón de tener en cuenta este Sistema es que $2^4 = 16$, por lo tanto a cada 4 cifras binarias se corresponderá 1 cifra hexadecimal. También en el Sistema Octal se da algo parecido ya que $2^3 = 8$, por lo tanto cada tres cifras binarias se corresponde una Octal. Esto último nos permite encontrar rápidamente las equivalencias entre binario, Octal y hexadecimal, por ejemplo:

$$0010\ 1001\ 1110\ 0101_2 = 29E5_{16}$$
$$110\ 111\ 111\ 001\ 000\ 110_8 = 677106_8$$

Un grupo de **cuatro bits recibe el nombre de nible** y un **grupo de ocho bits recibe el nombre de octeto o byte.**

Es usual en estos temas utilizar los prefijos kilo, mega, giga, tera de una manera similar que en el Sistema Métrico Decimal, pero con algunas diferencias, a saber:

$$1\ Kilo = 1K = 1024 = 2^{10}$$
$$1\ Mega = 1M = 1024K = 2^{20}$$
$$1\ Giga = 1G = 1024M = 2^{30}$$
$$1\ Tera = 1T = 1024G = 2^{40}$$

Así, cuando decimos por ejemplo 1Mbit, estamos indicando que se trata de 1.048.576 bits (2^{20}), o si decimos 1Kbyte estamos indicando 1024 bytes (2^{10})

Todas las operaciones aritméticas estudiadas en el álgebra aplicadas a números al Sistema Decimal, son aplicables a los números binarios por cuanto ambos Sistemas responden al mismo algoritmo de numeración.

Conversión entre números de distintas bases

Para convertir las expresiones de números entre distintas bases puede usarse el siguiente procedimiento general:

a) *Parte entera del número* (términos del polinomio con la base elevada a exponentes cero o positivos)

Supongamos que el primer miembro de la siguiente igualdad esta expresado en un Sistema de Numeración en base $b = b_1$, y el segundo miembro en base $b = b_2$

$$N_{b1} = a_n\, b_2^{\,n} + a_{n-1}\, b_2^{\,n-1} + \ldots\ldots + a_1\, b_2^{\,1} + a_0\, b_2^{\,0}$$

Si dividimos miembro a miembro entre b_1 vemos que el último término del segundo miembro será a_0, que es el resto de la división y la cifra menos significativa del número en base b_2.

Repitiendo la división entre el cociente anterior y b_2, obtendremos como resto a_1, y así sucesivamente hasta llegar a a_n. Un ejemplo aclarará lo expuesto:

Ejemplo:
$b_1 = 10_{10}$
$N_{b1} = 25_{10}$
$b_2 = 2_{10}$

25	2				
Resto = 1	*12*	2			
	Resto = 0	*6*	2		
		Resto = 0	*3*	2	
			Resto = 1	**Resto = 1**	2

De acuerdo a lo mencionado, tenemos:

$$25_{10} = 11001_2$$

b) *Parte fraccionaria del número* (términos del polinomio con la base elevada a exponentes negativos)

Supongamos que el primer miembro de la siguiente igualdad esta expresado en un Sistema de Numeración en base $b = b_1$, y el segundo miembro en base $b = b_2$

$$N_{b1} = a_{-1}\, b_2^{\,-1} + a_{-2}\, b_2^{\,-2} + \ldots\ldots + a_{k+1}\, b_2^{\,k+1} + a_{-k}\, b_2^{\,-k}$$

Si multiplicamos miembro a miembro por b_1 observamos que el primer término del segundo miembro es a_{-1}, es decir la primera cifra más significativa de la parte fraccionaria. Reiterando la multiplicación sólo sobre la parte fraccionaria del resultado, vamos obteniendo las sucesivas cifras menos significativas.

Un ejemplo aclarará lo expuesto:

Ejemplo:

$b_1 = 10_{10}$
$N_{b1} = 0{,}21_{10}$
$b_2 = 2_{10}$

0,21 x 2 = ***0****,42*
0,42 x 2 = ***0****,84*
0,84 x 2 = ***1****,68*
0,68 x 2 = ***1****,36*
0,36 x 2 = ***0****,72*
0,72 x 2 = ***1****,44*
0,44 x 2 = ***0****,88*
.....
.....

De acuerdo a lo mencionado tenemos:

$$0{,}21_{10} = 0{,}0011010\ldots\ldots_2$$

Si tenemos números con parte entera y fraccionaria, aplicamos la parte a) y la parte b).

Complementos

Complemento a la base:

El complemento a la base de un número N que posee m cifras enteras, se define como:

$$C_b(N) = b^m - N$$

Complemento a la base disminuida:

El complemento a la base disminuida de un número N que posee m cifras enteras, se define como:

$$C_{b-1}(N) = b^m - N - 2^{-k}$$

Donde *k* es la cantidad de cifras fraccionarias.

Si bien los complementos se aplican a Sistemas de Numeración de cualquier base, los utilizaremos especialmente en el Sistema de Numeración Binario, en este caso los llamaremos Complemento a Dos y Complemento a Uno. Veamos algunos ejemplos:

Ejemplo de *Complemento a 2:*

$C_2(10011_2) = 10_2^{1012} - 10011_2 = 100000_2 - 10011_2 = 01101_2$

Ejemplo de *Complemento a 1:*

$C_1(10011_2) = 10_2^{1012} - 10011_2 - 1_2 = 100000_2 - 10011_2 - 1_2 = 01100_2$

De ahora en adelante se omitirán los *subíndices 2* para indicar números binarios, a menos que no se pueda interpretar correctamente.

Observando los ejemplos vemos que el **Complemento a 1 puede obtenerse cambiando ceros por unos y viceversa**. Esto es consecuencia de restar a *m* unos un número de *m* bits. Como consecuencia de lo anterior el **Complemento a 2 de un número binario puede obtenerse cambiando unos por ceros y viceversa y sumando uno**, de acuerdo a la definición de Complemento a 2.

Para el caso de números binarios con parte fraccionaria (por ejemplo 1100011,110) la definición y reglas mencionadas siguen siendo válidas.

Representación de los números negativos en binario

Los números negativos se pueden representar de distintas formas, a saber:

a) Valor Absoluto y Signo:

Supongamos un número binario de n bits, el bit más significativo se reserva para el signo (*bit de signo Bs*) y los restantes para el valor absoluto (*Mantisa M*), esta es la forma en que representamos los números negativos en decimal. Bit de Signo 0 (cero) : + (positivo), 1 (uno) : - (negativo).

b) Mediante el Convenio de Complemento a 2:

El bit más significativo es el *bit de signo*, los restantes (*Mantisa M*), si el número es negativo, son el complemento a 2. . Bit de Signo

0 : + (positivo),
1 : - (negativo).

c) Mediante el Convenio de Complemento a 1:

El bit más significativo es el *bit de signo*, los restantes (*Mantisa M*), si el número es negativo, son el complemento a 1. Bit de Signo

0 : + (positivo),
1 : - (negativo).

d) Mediante el Convenio Exceso 2^{n-1}:

El bit más significativo es el *bit de signo*, los restantes (*Mantisa M*), si el número es negativo, son el complemento a 2. . Bit de Signo

0 : - (negativo),
1 : + (positivo).

Veamos un ejemplo de los distintos Convenios para un número de 3 bits. (n = 3).

Obsérvese que:

- ✓ Usando Complemento a 2 se tiene un solo cero lo que puede resultar una ventaja como después se verá.
- ✓ Usando Exceso 4 a menores combinaciones binarias absolutas se corresponden menores equivalentes decimales. Esto es útil para la representación de los exponentes en Punto Flotante, como veremos luego.
- ✓ En Unidades posteriores se verá la conveniencia del uso de los Convenios que usan Complementos ya que de esta forma la operación resta se puede realizar mediante una suma. Esto simplifica los circuitos lógicos.

Combinación Binaria (n = 3)		*Valor absoluto y Signo*	*Usando Complemento a 2*	*Usando Complemento a 1*	*Exceso 4*
Bs	*Mantisa*				
0	*00*	*+0*	*0*	*+0*	*-4*
0	*01*	*+1*	*+1*	*+1*	*-3*
0	*10*	*+2*	*+2*	*+2*	*-2*
0	*11*	*+3*	*+3*	*+3*	*-1*
1	*00*	*-0*	*-4*	*-3*	*0*
1	*01*	*-1*	*-3*	*-2*	*+1*
1	*10*	*-2*	*-2*	*-1*	*+2*
1	*11*	*-3*	*-1*	*-0*	*+3*

A excepción del Valor Absoluto y Signo, la utilización de estas representaciones para realizar operaciones aritméticas requiere de definir reglas apropiadas que se irán exponiendo más adelante. No obstante adelantaremos algunos conceptos.

En la tabla de arriba puede observarse que para el caso de los Convenios que usan Complemento a 2 y Complemento a 1 se puede incluir el bit de signo como un bit más del número, así por ejemplo para encontrar la combinación binaria correspondiente a *-3* en el Convenio que usa Complemento a 2, basta con partir de la combinación correspondiente a +3 (*011*) y encontrar su Complemento a 2 incluyendo el Bit de Signo, es decir $C_2(011) = 1000 - 011 = 101$. De esta forma el Bit de Signo se utiliza como un bit más. Veamos el siguiente ejemplo:

$$+3 + (-3) = 0$$

será usando el Convenio de Complemento a 2:

$$\begin{array}{r} 0\ 1\ 1 \\ + \\ 1\ 1\ 1 \\ \hline \not{1}\ 0\ 0\ 0 \end{array}$$

Donde el 1 de la izquierda se rechaza por tratarse de números de 3 bits.

Si usamos el Convenio de Complemento a 1 será:

$$+1 + (-3) = -2$$

$$\begin{array}{r} 0\ 0\ 1 \\ + \\ 1\ 0\ 0 \\ \hline 1\ 0\ 1 \end{array}$$

Como se ve hemos usado el bit de signo como si se tratara de un bit más.

Punto Fijo y Punto Flotante

Lamentablemente no todos los números son *enteros,* así surgen representaciones binarias con una, dos, tres o más dígitos (bits) fraccionarios según la necesidad. La llamada Notación en Punto Fijo resuelve ese problema, por ejemplo un formato podría ser:

1101001,101

en este caso disponemos de 10 bits para representar el número de los cuales 7 son enteros y 3 fraccionarios. Al construir un Sistema Digital debemos fijar el mismo formato para todos los números.

Además en las ciencias es usual la necesidad de representar números muy pequeños y a la vez números muy grandes. Con una cantidad limitada de dígitos resulta imposible abarcar un amplio rango de cantidades sin usar la llamada *Notación en Punto Flotante*. Esta Notación, muy similar a la Notación Científica, divide al número en tres campos, a saber:

- ✓ ***Signo de la Mantisa***
- ✓ ***Exponente****: Signo del Exponente*
  Mantisa del Exponente
- ✓ ***Mantisa***

Signo de la Mantisa	Exponente	Mantisa

Signo de la Mantisa (S)

Es un campo de 1 bit.

0 positivo (+)
1 negativo (-)

Exponente (E)

Es un campo de varios bits con la ***Convención Exceso 2^{n-1}***, por ejemplo el Exceso 64 (7 bits) y el Exceso 128 (8 bits).

Mantisa(M)

Es un campo de varios bits fraccionarios. Para que el número esté *normalizado*, el bit más significativo debe ser 1, es decir que la mantisa debe ser mayor o igual a 1/2 y menor que 1.

Operaciones Aritméticas

Las operaciones aritméticas con números en punto flotante tienen ciertas particularidades. Sean $S_1E_1M_1$ y $S_2E_2M_2$ dos números en punto flotante normalizados y en Exceso 64.

Multiplicación

El producto de los dos números será $S_pE_pM_p$, donde:

Signo de la mantisa

$S_p = 0$ *si* $S_1 = S_2$

$S_p = 1$ *si* $S_1 \neq S_2$

Exponente

$E_p = E_1 + E_2 - 64 - K$, *siendo K el valor que satisface* $1/2 <= M_1 \times M_2 \times 2^K < 1$

Mantisa

$M_p = M_1 \times M_2 \times 2^K$

División

La división de los dos números será $S_dE_dM_d$, donde:

Signo de la mantisa

$S_d = 0$ *si* $S_1 = S_2$

$S_d = 1$ *si* $S_1 \neq S_2$

Exponente

$E_d = E_1 - E_2 + 64 - K$, *siendo K el valor que satisface* $1/2 <= (M_1 / M_2)\ 2^K < 1$

Mantisa

$M_d = (M_1 / M_2)\ 2^K$

Suma y resta

Para sumar o restar los dos números debemos igualar sus exponentes. Debido a que los exponentes están en Exceso y en esta convención menores números se corresponden con menores combinaciones binarias absolutas, la comparación es muy sencilla. Supongamos que:

$E_1 > E_2$

Entonces encontramos *j*, tal que

$j = E_1 - E_2$

Modificamos $S_2E_2M_2$ a fin de igualar los exponentes:

$S_2(E_2 + j)M_2/2^j$

Procedemos a sumar o restar las mantisas según sea el caso. El resultado debe ser luego normalizado como se expuso en la división y el producto.

Base de Exponenciación

A fin de aumentar el rango de representación la base de exponenciación en vez de 2 puede ser 4, 8 o 16. Por ejemplo si en una representación en punto flotante la base de exponenciación es 8, la mantisa debe interpretarse en Octal (grupos de 3 bits partiendo desde la coma hacia la derecha). La normalización en estos casos se entiende cuando el dígito más significativo (grupo de tres bits) de la mantisa no es cero.

Norma IEEE 754

A fin de estandarizar la representación en punto flotante el IEEE dictó la Norma 754 (1985) la cual es respetada actualmente por la mayoría de los fabricantes de procesadores.

- ✓ Esta norma establece tres tipos de representación:

Simple Precisión (32 bits)

1 bit	8 bits	23 bits

Doble Precisión (64 bits)

1 bit	11 bits	52 bits

Precisión Extendida (80 bits)

1 bit	15 bits	64 bits

✓ Los exponentes mínimo y máximo se utilizan para casos especiales.

Ya que las mantisas normalizadas siempre empiezan con 1, no es necesario almacenarlo y la coma implícita se coloca a la derecha del 1, entonces la mantisa, ahora llamada **significando**, es mayor o igual a 1 y menor que 2.

✓ Se consideran 5 clases de números, a saber:

1. Normalizados

+/-	0 < E < Máximo	Cualquier patrón de bits

2. Desnormalizados (el bit 1 a la izquierda de la coma del significando, ahora es 0)

+/-	0	Cualquier patrón de bits excepto cero

3. Cero (el bit 1 a la izquierda de la coma del significando, ahora es 0)

+/-	0	0

4. Infinito

+/-	1111....1111111	0

5. No numero (Indica el cociente Infinito/Infinito)

+/-	1111....1111111	Cualquier patrón de bits excepto cero

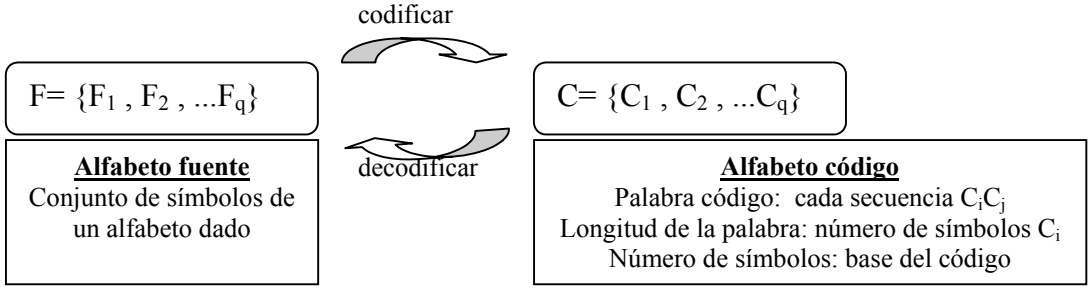
CÓDIGOS

Definición:

CÓDIGO: ley de correspondencia biunívoca entre los elementos de dos conjuntos.

Código es una representación unívoca de algún conjunto de elementos de tal forma que, a cada uno de estos, se le asigna una combinación de símbolos determinados y viceversa.

Para "comunicarse" hay que conocer el código utilizado.



Códigos numéricos: son cuando los elementos de los dos conjuntos son números

CÓDIGOS BINARIOS

Los sistemas de numeración (binario, octal, hexadecimal, etc) estudiados constituyen códigos de representación de las cantidades.

CÓDIGOS BINARIOS: la base del código es 2, son los utilizados por los sistemas digitales.

El sistema de numeración binario recibe el nombre de código binario natural:

decimal	*binario natural*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Con n cifras binarias (bits) se pueden obtener 2^n combinaciones diferentes. Cada combinación se puede asignar a un elemento (o cantidad) distinto.
Es decir, que:
• Un grupo de *n* bits binarios es un código de *n* bits que puede asumir 2^n combinaciones distintas de *1´*s y *0´*s. Para que el código sea útil no se puede asignar ninguna combinación a más de un elemento, y varían de 0 a (2^n – 1)
• Aunque el *mínimo* número de bits requerido para escribir un código con 2^n elementos distintos es *n*, el máximo número de bits no está especificado.
• Con estas combinaciones podríamos tener 2^n! (las permutaciones de las 2^n combinaciones) códigos distintos; aunque solo se verán algunos con características particulares.

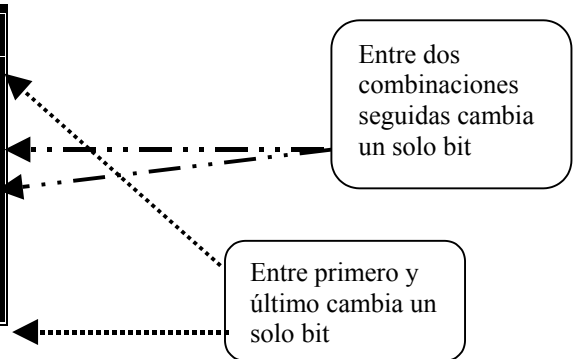
En el estudio de los códigos binarios existen algunos con ciertas características, como:

Códigos binarios continuos (o progresivos): son aquellos que las combinaciones correspondientes a números decimales consecutivos son adyacentes. Se llaman combinaciones binarias adyacentes las que difieren solamente en un bit.

Códigos binarios cíclicos: son aquellos códigos continuos que la última combinación es adyacente a la primera.

Ejemplos:

decimal	*continuo*	*cíclico*
0	**0000**	**0000**
1	**0001**	**0001**
2	**0011**	**0011**
3	**0110**	**0111**
4	**0111**	**1111**
5	**1111**	**1110**
6	**1110**	**1100**
7	**1100**	**1000**



El ejemplo más típico de un código continuo y cíclico es el llamado de GRAY, el cual puede tener 2, 3, 4 ó más bits de acuerdo a la necesidad. También llamado "reflejado" porque para construir el código de n bits, se "reflejan" las combinaciones del código n-1.
Se utiliza principalmente en convertidores muy rápidos y en codificadores. Como de una combinación binaria a la contigua sólo cambia un bit, se aumenta la velocidad y se elimina el riesgo de transiciones intermedias. Para ampliar palabras codificadas en este código, en un bit más, basta con repetir simétricamente las combinaciones (como si fuera un espejo) y agregar un bit a la izquierda, siendo cero para la primera mitad y uno para la segunda mitad de las combinaciones posibles.
Representamos a continuación el código de Gray para 2, 3 y 4 bits.

decimal	GRAY		
	2 bits	3 bits	4 bits
0	0 0	0 0 0	0 0 0 0
1	0 1	0 0 1	0 0 0 1
2	1 1	0 1 1	0 0 1 1
3	1 0	0 1 0	0 0 1 0
4		1 1 0	0 1 1 0
5		1 1 1	0 1 1 1
6		1 0 1	0 1 0 1
7		1 0 0	0 1 0 0
8			1 1 0 0
9			1 1 0 1
10			1 1 1 1
11			1 1 1 0
12			1 0 1 0
13			1 0 1 1
14			1 0 0 1
15			1 0 0 0

Otro ejemplo de código binario continuo y cíclico es el de Johnson, el cual representamos a continuación con 5 bits:

decimal	Código JOHNSON
0	0 0 0 0 0
1	0 0 0 0 1
2	0 0 0 1 1
3	0 0 1 1 1
4	0 1 1 1 1
5	1 1 1 1 1
6	1 1 1 1 0
7	1 1 1 0 0
8	1 1 0 0 0
9	1 0 0 0 0

Códigos BCD:

Los números puros se representan con la notación octal o con la hexadecimal, por su facilidad de conversión. Sin embargo, la conversión binaria a decimal es bastante difícil, y en las calculadoras, los juegos electrónicos, y los instrumentos digitales, en los que que generalmente es común la entrada y salida de números en notación decimal, se emplea un código especial para representar esta notación . A este código se le llama *CODIGO BCD*: decimal codificado en binario (binary - coded - decimal).

Con esta técnica es posible convertir cada numero decimal a su equivalente en binario, en lugar de convertir el valor decimal entero a su forma binaria pura. Para poder representar los diez dígitos decimales (del 0 al 9) son necesarios 4 bits. De las 16 combinaciones posibles (2^4) que se obtienen con los 4 bits, sólo se utilizan diez.

Los códigos BCD se pueden clasificar en:

Ponderados: poseen una regla de formación que adjudica un cierto "peso" a los "unos" binarios según la posición que ocupan en el conjunto de los 4 bits, debiéndose verificar que la suma algebraica de los pesos de cada combinación sea igual al número decimal representado. Los ejemplos más tìpicos son el ***BCD natural*** (que tiene los pesos 8 4 2 1) y el ***AIKEN*** (que tiene los pesos 2 4 2 1).

Libres o no ponderados: en estos la correspondencia decimal-binaria es arbitraria con respecto a que no existen "pesos" ni sumas algebraicas, aunque en general las combinaciones se forman según ciertas reglas específicas para cada código. El ejemplo típico es el ***Exceso 3*** que se forma partiendo del BCD natural y sumándole 3 a cada combinación.

decimal	Binario natural	AIKEN	Exceso 3
	8 4 2 1	2 4 2 1	n + 3
0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 1 0 0	1 0 0 1
7	0 1 1 1	1 1 0 1	1 0 1 0
8	1 0 0 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 0 0
	ponderados		Libre

La conversión de un número decimal a un código BCD se realiza simplemente expresando cada dígito mediante la combinación binaria que le corresponde de acuerdo al código especificado.
Por ejemplo el número decimal 926 lo representamos en distintos códigos:

Número decimal	9 2 6
BCD natural	1000 0010 0110
BCD AIKEN	1111 0010 1100
BCD exceso 3	1100 0101 1001

Códigos Alfanuméricos

Muchas aplicaciones de computadoras digitales, requieren manejar datos que consisten no solamente de números sino también de letras. Por ejemplo una compañía de seguros con millones de clientes pueden usar un computador digital para procesar sus historias. Para representar el nombre del dueño de una póliza en forma binaria, es necesario tener un código binario para el alfabeto. Además el mismo código binario puede representar números decimales y algunos otros caracteres especiales.
Un código alfanumérico es un código binario de un grupo de elementos consistente de los diez números decimales, los 26 caracteres del alfabeto y de cierto número de símbolos especiales tales como el $
Uno de estos códigos es conocido como ASCII (American Standard Code for Information Interchange = Código normalizado americano para el intercambio de información).
A continuación se muestra una lista del código ASCII de 7 bits para los números, letras mayúsculas y signos de puntuación. El código ASCII completo tiene códigos para letras minúsculas y signos de control.
ASCII incluye 256 códigos divididos en dos conjuntos, estándar y extendido, de 128 cada uno. Estos conjuntos representan todas las combinaciones posibles de 7 u 8 bits. El conjunto ASCII básico, o estándar, utiliza 7 bits para cada código, lo que da como resultado 128 códigos de caracteres desde 0 hasta 127 (00H hasta 7FH hexadecimal). El conjunto ASCII extendido utiliza 8 bits para cada código, dando como resultado 128 códigos adicionales, numerados desde el 128 hasta el 255 (80H hasta FFH extendido).

En el conjunto de caracteres ASCII básico, los primeros 32 valores están asignados a los códigos de control de comunicaciones y de impresora —caracteres no imprimibles, como retroceso, retorno de carro y tabulación— empleados para controlar la forma en que la información es transferida desde una computadora a otra o desde una computadora a una impresora. Los 96 códigos restantes se asignan a los signos de puntuación corrientes, a los dígitos del 0 al 9 y a las letras mayúsculas y minúsculas del alfabeto latino.

Los códigos de ASCII extendido, del 128 al 255, se asignan a conjuntos de caracteres que varían según los fabricantes de computadoras y programadores de *software*. Estos códigos no son intercambiables entre los diferentes programas y computadoras como los caracteres ASCII estándar. Por ejemplo, IBM utiliza un grupo de caracteres ASCII extendido que suele denominarse conjunto de caracteres IBM extendido para sus computadoras personales. Apple Computer utiliza un grupo similar, aunque diferente, de caracteres ASCII extendido para su línea de computadoras Macintosh. Por ello, mientras que el conjunto de caracteres ASCII estándar es universal en el *hardware* y el *software* de los microordenadores, los caracteres ASCII extendido pueden interpretarse correctamente sólo si un programa, computadora o impresora han sido diseñados para ello.

CODIFICACION PARA DETECCION Y CORRECCION DE ERROR

En la transmisión de información es posible que se produzcan errores debido a diferentes factores, como la presencia de ruido en el proceso, la avería de alguno de los componentes, etc.
Especialmente cuando la información es numérica, no se aceptan errores, por lo que es necesario poder detectar la presencia de los mismos si se han producido. Si se detecta la presencia de un error es necesario retransmitir la información, mientras que si se puede detectar y corregir el error producido, no se produce la retransmisión.
Cuando en un código binario se utilizan todas las combinaciones posibles, no se pueden detectar errores; debido a que si cambia algunos de sus bits por error de una combinación válida, daría por resultado otra combinación de bits válida. Por lo tanto, podemos decir que "*para detectar errores es necesario no utilizar todas las combinaciones posibles*". Esto último es una condición necesaria pero no suficiente.
Para poder efectivamente detectar y/o corregir errores se da el concepto de distancia mínima de un código:
Distancia de un código entre dos combinaciones binarias es el número de bits que cambian para pasar de una combinación a la otra.
Distancia mínima de un código es la menor de las distancias entre dos combinaciones cualesquiera pertenecientes al código. La distancia mínima de los códigos vistos hasta ahora es la unidad, y por lo tanto no se pueden detectar errores.
Por lo anterior, se deduce que, para poder detectar y/o corregir errores, es imprescindible que:

Distancia mínima > 1

Este concepto nos da una idea de la capacidad de detección y corrección de errores de un código:

Distancia mínima	Significado
1	Sin detección ni corrección
2	Detección de error simple
3	Detección de error doble y/o corrección de error simple
4	Detección de error triple y/o corrección de error doble

CÓDIGOS DETECTORES DE ERROR:

La manera más simple de codificar un mensaje binario para permitir la detección de un error (un bit errado en cada combinación) es contar el número de 1´s (unos binarios) contenidos en el mensaje y agregarle un dígito binario (un bit) de forma tal que el mensaje tenga un número par de 1´s (o un número impar de 1´s). A esto se llama *código con paridad* y al uno que se adiciona *bit de paridad*. Si el número de 1´s es par se trabaja con *paridad par* y sino es impar. Antes de enviar una información es necesario ponerse de acuerdo en cual de las dos convenciones se trabaja. Al adicionarle un bit al código se consigue que la distancia mínima sea por lo menos de 2.
Ejemplo del código BCD natural:

decimal	*BCD natural*	*BCD natural c/paridad par*	*BCD natural c/ paridad impar*
0	**0000**	**0000 0**	**0000 1**
1	**0001**	**0001 1**	**0001 0**
2	**0010**	**0010 1**	**0010 0**
3	**0011**	**0011 0**	**0011 1**
4	**0100**	**0100 1**	**0100 0**
5	**0101**	**0101 0**	**0101 1**
6	**0110**	**0110 0**	**0110 1**
7	**0111**	**0111 1**	**0111 0**
8	**1000**	**1000 1**	**1000 0**
9	**1001**	**1001 0**	**1001 1**

Distancia mínima entre cualquier combinación mayor que 1

Bit de paridad

Esta característica se puede realizar con cualquier código con el cual se trabaje.
La detección de errores en estos códigos consiste en comprobar si el número de unos de cada combinación cumple con la convención adaptada. Es decir, si se trabaja con paridad par, se debe chequear que el dato recibido cumpla con esta condición; si no es así existe un error, y se debe solicitar una nueva transmisión del dato.

Otros códigos detectores de error son los de cantidad constante de 1´s, como:

decimal	*2 entre 5* *7 4 2 1 P*	*biquinario* *5 0 4 3 2 1 0*
0	**1 1 0 0 0**	**0 1 0 0 0 0 1**
1	**0 0 0 1 1**	**0 1 0 0 0 1 0**
2	**0 0 1 0 1**	**0 1 0 0 1 0 0**
3	**0 0 1 1 0**	**0 1 0 1 0 0 0**
4	**0 1 0 0 1**	**0 1 1 0 0 0 0**
5	**0 1 0 1 0**	**1 0 0 0 0 0 0**
6	**0 1 1 0 0**	**1 0 0 0 0 1 0**
7	**1 0 0 0 1**	**1 0 0 0 1 0 0**
8	**1 0 0 1 0**	**1 0 0 1 0 0 0**
9	**1 0 1 0 0**	**1 0 1 0 0 0 0**

Estos códigos tienen dos unos en cada combinación (o sea paridad par) y se forman de acuerdo a los pesos asignados.

CÓDIGOS CORRECTORES DE ERROR

Un código corrector detecta si la información codificada presenta o no errores, y en caso afirmativo determina la posición del bit o bits erróneos, de manera de poder corregirlos por inversión (recordar que es el sistema binario, así que si un bit tiene error, se cambia por su complemento).
Los códigos de distancia mínima dos no permiten la correccción de errores, porque al producirse un error la combinación obtenida puede poseer dos adyacentes pertenenecientes a código, y no es posible saber cuál es la correcta. Por ejemplo si del código *2 entre 5* se detecta la combinación errónea 01000 (que no es válida), no es posible conocer si el error se produjo en el primer bit (01001) o en el segundo bit (01010), ya que ambas combinaciones son válidas. Por lo tanto se deduce que para corregir un bit de error es necesario una distancia mínima mayor a dos.
En general para corregir bits de error se necesita una distancia dada por:

$$dm = 2n + 1 \qquad \text{I}$$

siendo n la cantidad de bits a corregir.

Un ejemplo típico es el ***código de Hamming***:
Su formación parte de un código cualquiera de *n bits* al que le adicionan *p bits* formando un nuevo código de *n + p bits.* En este código se realizan *p* detecciones de paridad, obteniéndose un bit de paridad uno o cero (dependiendo si el número de bits es par o impar). El conjunto de los *p* bits de paridad forma un número en el sistema binario natural cuyo equivalente decimal nos indica la posición del bit erróneo. Si no hay error el número obtenido debe ser cero.
Dado que con *p* bits se obtienen 2^p combinaciones, se debe cumplir la siguiente realción:

$$2^p >= n + p + 1 \qquad \text{II}$$

Como ejemplo tomaremos el código AIKEN, al cual queremos poder detectar y corregir un bit de error. Como n = 4 hay que agregarle p = 3 bits para cumplir con la *condición II,* lo que nos queda un código de 7 bits. Para detectar los siete posibles errores que se pueden producir en una combinación (7 errores considerando 7 posiciones en cada combinación) y la ausencia de error, son necesarias ocho combinaciones binarias (correctoras de error). Dichas combinaciones correctoras de error se obtiene mediante 3 bits (*c1, c2* y *c3*), y el número decimal formado por ellos indica la posición del bit erróneo.

Nº decimal equivalente	C3	C2	C1
0	0	0	0
1	0	0	**1**
2	0	**1**	0
3	0	**1**	**1**
4	**1**	0	0
5	**1**	0	**1**
6	**1**	**1**	0
7	**1**	**1**	**1**

Las combinaciones de estos bits son:

El bit c1 toma el valor 1 si se produce un error en los bits b1, b3, b5 y b7 de la combinación del código formado. Si el número de unos existentes en esas cuatro posiciones es siempre par, un error en uno cualquiera de esos cuatro bits lo convierte en impar. Por lo tanto, c1 vale uno si el número de unos en las posiciones dadas es impar, y cero en caso contrario. Si lo representamos en una función lógica:

$$C1 = b1 \oplus b3 \oplus b5 \oplus b7$$

donde $\oplus$ es el símbolo de la función OR-Exclusiva (que se estudiará después), y que da como resultado 0 si el nùmero de unos es par, y uno si el número de unos es impar.
De igual manera obtenemos c2 y c3:

$$C2 = b2 \oplus b3 \oplus b6 \oplus b7$$

$$C3 = b4 \oplus b5 \oplus b6 \oplus b7$$

Para generar los 3 bits (p bits) a agregarle al código general consideremos que los bits b1, b2 y b4 aparecen una sola vez en las fórmulas anteriores, con lo cual los generamos a partir de los otros, ya que por ejemplo b1 debe valer uno si el número de unos de b3, b5 y b7 es impar; por lo tanto:

$b1 = b3\ b5 \oplus b7 \oplus$

$b2 = b3\ b6 \oplus b7 \oplus$

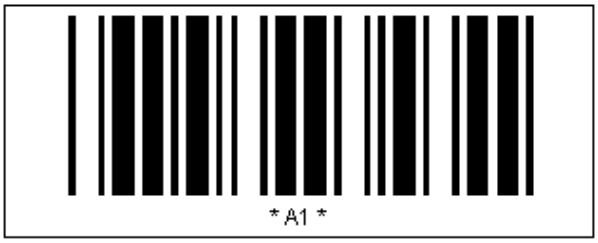
$b4 = b5\ b6 \oplus b7 \oplus$

de todo esto se deduce que el Código Hamming resultante a partir del AIKEN es:

Nº decimal equivalente	b7 b6 b5	**b4**	b3	**b2 b1**
0	0 0 0	**0**	0	**0 0**
1	0 0 0	**0**	1	**1 1**
2	0 0 1	**1**	0	**0 1**
3	0 0 1	**1**	1	**1 0**
4	0 1 0	**1**	0	**1 0**
5	1 0 1	**0**	1	**0 1**
6	1 1 0	**0**	0	**0 1**
7	1 1 0	**0**	1	**1 0**
8	1 1 1	**1**	0	**0 0**
9	1 1 1	**1**	1	**1 1**

B3, b5, b6 y b7 son los bits del código original AIKEN, mientras que los bits restantes se forman de acuerdo a las condiciones de paridad (en este caso paridad par)

OTROS CODIGOS UTILIZADOS:
CÓDIGO DE BARRAS



El Código de Barras es un arreglo en paralelo de barras y espacios que contiene información codificada en las barras y espacios del símbolo. Esta información puede ser leída por dispositivos ópticos (lectores de código de barras), los cuales envían la información leída hacia una computadora como si la información fuera una entrada de teclado.
Otra definición es: símbolos hechos de patrones de barras y espacios blancos y negros.
Dentro de los códigos de barra se codifican bits de información. Los datos son leídos por scanners especiales de códigos de barra y se usan muy a menudo en conjunto con bases de datos. Los códigos de barra no requieren ingreso manual por el ser humano ya que pueden ser leídos automáticamente por los scanners y son virtualmente libres de error
Los Scanners de códigos de barra leen el patrón de barras blancas y negras (o mejor dicho, claras y oscuras) y decodifican el código, convirtiéndolo en un "string de caracteres" que generalmente se guarda en una base de datos.

Ventajas:
Algunas de sus ventajas sobre otros procedimientos de colección de datos son:

- Se imprime a bajos costos
- Permite porcentajes muy bajos de error
- Los equipos de lectura e impresión de código de barras son flexibles y fáciles de conectar e instalar.

Beneficios
El código de barras es una técnica de entrada de datos, como son la captura manual, el reconocimiento óptico y la cinta magnética.
Se considera que la tecnología de código de barras es la mejor tecnología para implementar un sistema de colección de datos mediante identificación automática, y presenta muchos beneficios, entre otros.

- Virtualmente no hay retrasos desde que se lee la información hasta que puede ser usada
- Se mejora la exactitud de los datos
- Se tienen costos fijos de labor más bajos
- Se puede tener un mejor control de calidad, mejor servicio al cliente
- Se pueden contar con nuevas categorías de información.
- Se mejora la competitividad.

Aplicaciones
Las aplicaciones del código de barras cubren prácticamente cualquier tipo de actividad humana, tanto en industria, comercio, instituciones educativas, instituciones médicas, gobierno, etc.

- Control de material en proceso
- Control de inventario
- Control de tiempo y asistencia
- Punto de venta
- Control de calidad
- Control de inventario
- Embarques y recibos
- Control de documentos
- Facturación
- Bibliotecas
- Bancos de sangre
- Hospitales
- Control de acceso
- Control de tiempo y asistencia

Características de un código de barras

Un símbolo de código de barras puede tener, a su vez, varias características, entre las cuales podemos nombrar:

- **Densidad:**
  Es la anchura del elemento (barra o espacio) más angosto dentro del símbolo de código de barras. Está dado en mils (milésimas de pulgada). Un código de barras no se mide por su longitud física sino por su densidad.
- **WNR: (Wide to Narrow Ratio)**

Es la razón del grosor del elemento más angosto contra el más ancho. Usualmente es 1:3 o 1:2.

- **Quiet Zone:**
  Es el área blanca al principio y al final de un símbolo de código de barras. Esta área es necesaria para una lectura conveniente del símbolo.

Simbologías

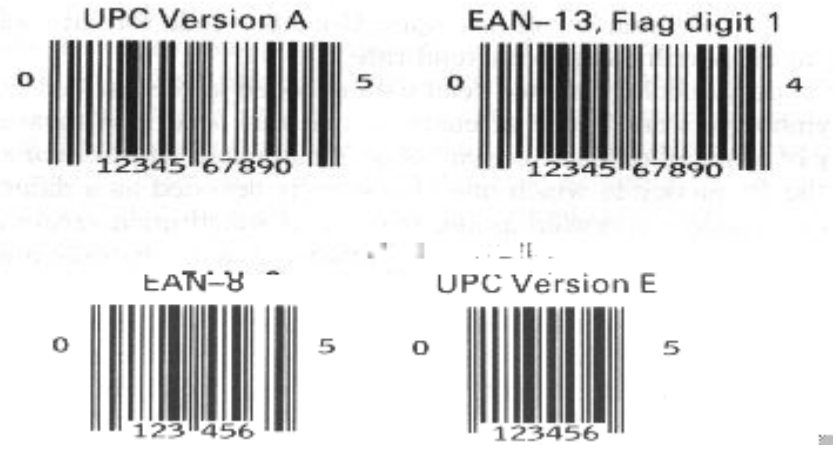
Un **símbolo de código de barras** es la impresión física de un código de barras.
Una **Simbología** es la forma en que se codifica la información en las barras y espacios del símbolo de código de barras.
Existen diferentes simbologías para diferentes aplicaciones, cada una de ellas con sus propias características. Las principales características que definen una simbología de código de barras son las siguientes:

- Numéricas o alfanuméricas
- De longitud fija o de longitud variable
- Discretas o continuas
- Número de anchos de elementos
- Autoverificación.

Las simbologías más usadas son:

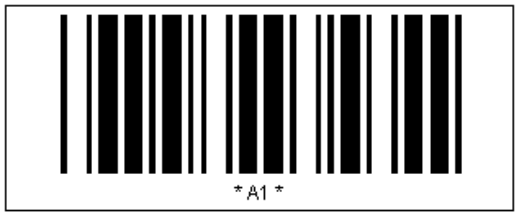
EAN/UPC

Comercio detallista, autoverificable, numérico, longitud fija.



Código 39

Industrial, alfanumérico, 44 caracteres

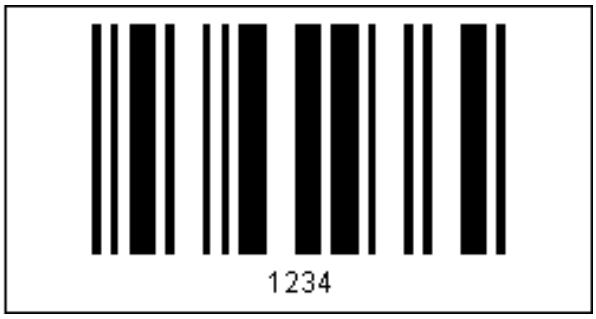


Codabar

Bancos de sangre, bibliotecas

I 2/5

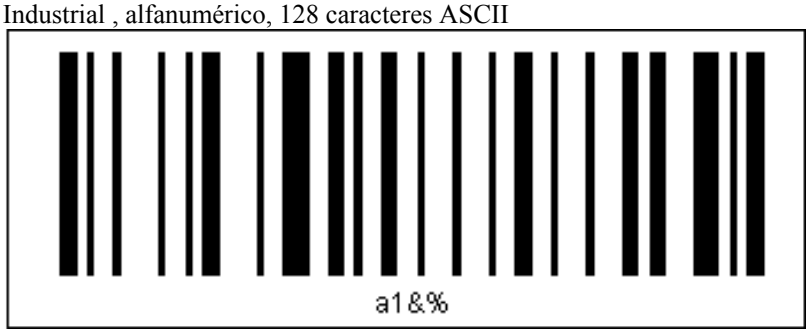
Aplicaciones numéricas, aerolíneas, numérico



Código 93

Complementa al código 39, alfanumérico

Código 128
Industrial , alfanumérico, 128 caracteres ASCII



Características de un código de barras

Un símbolo de código de barras puede tener, a su vez, varias características, entre las cuales podemos nombrar:

Densidad:
Es la anchura del elemento (barra o espacio) más angosto dentro del símbolo de código de barras. Está dado en mils (milésimas de pulgada). Un código de barras no se mide por su longitud física sino por su densidad.

WNR: (Wide to Narrow Ratio)
Es la razón del grosor del elemento más angosto contra el más ancho. Usualmente es 1:3 o 1:2.

Quiet Zone:
Es el área blanca al principio y al final de un símbolo de código de barras. Esta área es necesaria para una lectura conveniente del símbolo.

ENCRIPTACIÓN O CIFRADO

Con la introducción de las computadoras, especialmente en las empresas, fue evidente la necesidad de herramientas automáticas para proteger los archivos y otras informaciones almacenadas en su memoria. El nombre genérico del tema que trata las herramientas diseñadas para proteger los datos y frustrar a los usuarios no autorizados informáticos es la Seguridad en Computadoras. Esta unidad temática ha necesitado desarrollarse por la introducción de redes y facilidades de comunicación para transportar datos. La tecnología esencial en todas las redes automáticas y en las aplicaciones de seguridad en computadoras es la Encriptación o Cifrado.

La medida más efectiva en contra de la amenaza de usuarios no autorizados es el encriptado o cifrado de datos, que debe interpretarse como el almacenamiento o transmisión de datos sensibles en forma cifrada. Dentro de la terminología utilizada en este campo, se destaca el nombre de texto plano asignado a los datos originales. El texto plano es cifrado sometiéndolo a un algoritmo de cifrado, cuyas entradas son el texto plano y la clave de cifrado, y a la salida de este algoritmo (la forma cifrada del texto plano) se le llama texto cifrado.

Aunque los detalles del algoritmo normalmente son de dominio público, la clase de cifrado se mantiene en secreto. El texto cifrado que debe ser ininteligible para cualquiera que posea la clave de cifrado, es lo que se guarda en las computadoras principales de la organización o se transmite por las líneas de comunicaciones de las redes. El esquema empleado debería ser tal, que el trabajo involucrado en romperlo sobrepase cualquier ventaja potencial que pudiera obtenerse al hacerlo.

Existen dos técnicas diferenciadas que pueden utilizar los algoritmos: la sustitución y la permutación. La sustitución es uno de los enfoques básicos del cifrado, como se practica tradicionalmente, donde se usa una clave de cifrado para determinar, para cada carácter del texto plano), un caracter de texto cifrado que va a sustituir a ese carácter. Con la técnica de permutación, los caracteres de texto plano son simplemente reorganizados en una secuencia diferente, bajo la influencia de la clave de cifrado.

Por ejemplo, podríamos usar una técnica de sustitución, en un algoritmo elemental, para cifrar el siguiente texto plano y clave de cifrado:

Texto plano: ESTE ES UN TEXTO DEMO
Clave: PRUEBA

Suponemos, por simplicidad, que los únicos caracteres de datos que tenemos que manejar son las letras mayúsculas y los espacios en blanco. Y que el algoritmo de cifrado por sustitución sea el siguiente:

1. Dividimos el texto plano en bloques de longitud igual a la clave de cifrado.

   E S T E + E S + U N + T E X T O + D E M O

   (los espacios en blanco son mostrados explícitamente como "+").

2. Reemplazamos cada carácter del texto plano por un entero que esté en el rango de 00 a 26, usando espacio en blanco = 00, A = 01, ..., Z = 26.

   E S T E + E S + U N + T E X T O + D E M O

   05 19 20 05 00 05 19 00 21 14 00 20 05 24 21 15 00 04 05 13 15

3. Repetimos el paso 2 para la clave de cifrado.

   P R U E B A

   16 18 21 05 02 01

4. Para cada bloque de texto plano reemplazamos cada carácter por la suma módulo 27 de su codificación de enteros más la codificación de enteros del carácter correspondiente de la clave de cifrado:

05 19 20 05 00 05	19 00 21 14 00 20	05 24 21 15 00 04	05 13 15
16 18 21 05 02 01	16 18 21 05 02 01	16 18 21 05 02 01	16 18 21
21 10 14 10 02 06	08 18 15 19 02 21	21 15 15 20 02 05	21 04 09

5. Reemplazamos cada codificación de enteros del resultado del paso 4 por su equivalente en caracteres:

   U J N J B F H R O S B U U O O T B E U D I

:

El procedimiento de descifrado para este ejemplo es directo, siempre y cuando se tenga la clave. En este caso, pareciera que no sería tan difícil para un posible infiltrado determinar la clave sin ningún conocimiento previo, teniendo el texto plano y el texto cifrado. Aunque, es obvio que existen esquemas mucho más sofisticados.

Ninguna de las técnicas de sustitución y permutación es particularmente segura en sí misma, pero los algoritmos que combinan a las dos pueden proporcionar una alto grado de seguridad. Uno de estos algoritmos es el DES (Estándar de cifrado de datos) que usa clave de 64 bits. A través de los años muchas personas han sugerido que probablemente el DES no sea tan seguro para ciertas aplicaciones. Alternativamente, ha aparecido otros algoritmos que han ampliado el tamaño de la clave, y/o usan dos claves (una para encriptar y otra para desencriptar) como el RSA.

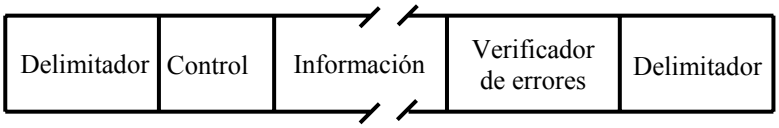
CODIGOS CICLICOS EN LA TRANSFERENCIA DE DATOS

Una vez definida la conexión física para poder transferir información entre los dispositivos o sistemas debe existir un formato para los datos y una estrategia de sincronización de como se envía y recepcionan los mensajes, incluyendo la detección y corrección de los errores.
En un enlace de datos se presentan bloques que cumplen diferentes funciones.

DTE: EquipoTerminal de Datos
DCE: Equipo de Comunicación de Datos

La trasferencia ordenada de información en un enlace de comunicación se logra estableciendo una conexión entre los DTE, identificando el emisor y el receptor, asegurando que todos los mensajes se transfieran correctamente sin errores, controlando toda la transferencia de información.
Los equipos (teléfono, transmisor, modem), las conexiones, los cables, repetidoras, etc., constituyen el soporte físico que permiten el enlace de datos.

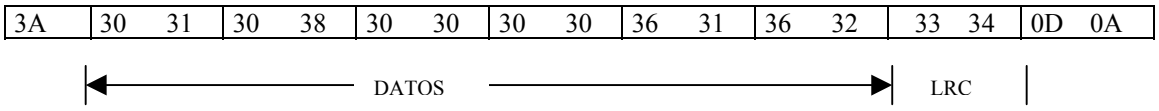
Un elemento básico a considerar es la estructura del mensaje, constituyendo una unidad de información denominada Cuadro, Bloque, Trama o Datagran. En general tiene la estructura siguiente



Teniendo en cuenta que los equipos y el canal esta expuestos a ruidos internos y externos que pueden alterar la información que se transmite, resulta de fundamental importancia detectar y corregir errores. Hay varias formas de verificar los errores que se producen en la trama durante una transmisión.

Códigos Cíclicos
Resulta de interés la utilización de códigos cíclicos, que son aquellos que al realizar una rotación cíclica de una palabra se produce otra palabra que pertenece al mismo código. La ventaja de emplear códigos cíclicos es que se puede implementar fácilmente su generación a partir de del empleo de registros de desplazamientos con realimentación

Chequeo de Redundancia Longitudinal (LRC)
Es un código detector de error en un bit. Utilizado en la adquisición de datos y control en la industria en la comunicación entre dispositivos y con un PC..
Tomando como ejemplo una trama con información en ASCII:



Donde 3A es **:** / 0D es CR / 0A es LF (Identifican el comienzo y final de la Trama)
El resto es un binario equivalente: 01 08 00 00 61 62
El cálculo del LRC consiste en realizar la suma hexadecimal de los datos y calcular el complemento a 2.

$$01 + 08 + 00 + 00 + 61 + 62 = CC = 0011\ 0011$$
$$+\ 1$$
$$0011\ 0100 \text{ en hexadecimal: } 33\ 34$$

Luego, LRC = 33 34

Chequeo de Redundancia Cíclica (CRC)
Es un código detector de error en un bit. A cada bloque de datos se le agrega una cantidad de bits redundantes. Utilizado en la verificación de datos grabados en disquetes y almacenamiento en memoria RAM
La trama de los datos es asociada a un polinomio de datos D(X)de grado n - 1 (Cantidad de bits menos uno). Los coeficientes se corresponden a los valores binarios de la trama. A cada posición de bit le corresponde un termino del polinomio
Para el ejemplo 01 08 00 00 61 62 corresponde una estructura binaria de 6 bytes (48 bits), es decir 48 – 1 = 47 El grado del polinomio asociado será de X^47:
0000 0001 0000 1000 0000 0000 0000 0000 0110 0001 0100 0010

$$D(X) = X^{40} + X^{35} + X^{14} + X^{13} + X^{8} + X^{6} + X^{1}$$

Los bits de redundancia se obtienen mediante un polinomio generador G(X). El polinomio asociado mas la redundancia debe ser divisible por el polinomio G(X), es decir *resto nulo.*

Los polinomios generadores G(X) están normalizados internacionalmente.

CRC-8 = X^8 + X^5 + X^4 + 1 (Redundancia de 8 bits)

CRC-12 = X^12 + X^11 + X^3 + X^2 + X^1 + 11 (Redundancia de 16 bits)

CRC-16 = X^16 + X^15 + X^2 + 1 (Redundancia de 16 bits)

Si se emplea el CRC-8, el procedimiento consiste en elevar a potencia de 8 el bloque de datos (desplazando a la izquierda 8 lugares). Luego se divide por X^8 + X^5 + X^4 + 1.
La parte entera del cociente se elimina y resto en 8 bit se agrega al final del mensaje como CRC

No se desarrolla el fundamento y procedimiento matemático del método. Cabe aclarar que se pueden implementar por hardware mediante registros de desplazamiento o por software.
En general se realiza por hardware para reducir el tiempo de cálculo.