TEMA 1 REPRESENTACIÓN INTERNA DE LOS DATOS.

1.1 INTRODUCCIÓN

Los datos no serán representables directamente en el ordenador. Para su uso en sistemas informáticos la información los representaremos mediante símbolos (no son la información en sí) y serán almacenados en diferentes tipos de soportes.

1.2 CONCEPTO DE CÓDIGO

Correspondencia entre un dato y su representación. Implica unas reglas que especifiquen sin ambigüedad dicha correspondencia.

Distinguiremos entre códigos **numéricos** y **alfanuméricos**.

El conjunto de todos los símbolos posibles que componen un determinado código se denomina **Alfabeto**. Y mediante este alfabeto es posible construir palabras y frases de dicho código.

En general para codificar un número de **N** informaciones necesitaremos al menos $\mathbf{log_2 N}$ bits.

La codificación que se elija debería cumplir las siguientes características:

- **Simplicidad**. Cuanto más complicado mayor será el porcentaje de errores.
- **Flexibilidad**. Que pueda admitir casos nuevos.
- **Amplitud**. Las variables con algo en común deben tener una parte del código en común.
- **Operatividad**. Facilidad de intercalar, clasificar y facilidad de reconocer el dato.
- **Concreción**. Facilidad de asignar, escribir, transcribir y registrar el dato.
- **Unicidad**. Cada dato un único código y viceversa.
- **Clasificación**. Los códigos de datos componente de un grupo, forman así mismo otro grupo. (facilidad de encontrar sinónimos)
- **Ordenación**. Los códigos deben quedar en el mismo orden que el orden lógico de los datos originales.

1.3 REPRESENTACIÓN NUMÉRICA

1.3.1 SISTEMA BINARIO

El sistema binario tiene como base el número 2, y dispone por tanto de dos símbolos distintos: 0 y 1. Estos símbolos se denominan abreviadamente **bits** (acrónimo de Binary digit). Este sistema de numeración es muy importe en el campo de la informática, porque en él se basa toda la lógica interna de una computadora.

Para pasar de binario a decimal sumamos los pesos de los dígitos con valor 1. Ej:

pesos	16	8	4	2	1
	1	0	1	1	0

El número en decimal sería: 16 + 4 + 2 = 22

Para la operación inversa (de decimal a binario) descomponemos el número decimal en función de los pesos de los dígitos binarios.

1.3.2 SISTEMA OCTAL

En bastantes ocasiones, por comodidad, es mejor manejar los datos en otros sistemas en lugar del binario. Los sistemas octal y hexadecimal son los más utilizados, debido a la facilidad para pasar desde ellos un número a binario y viceversa. Facilidad que viene motiva por ser la base de ambos una potencia de la base binaria (2).

El octal es un sistema posicional con base 8 (símbolos 0, 1, 2, 3, 4, 5, 6 y 7). Cada dígito octal se corresponde con 3 dígitos binarios.

OCTAL	0	1	2	3	4	5	6	7
BINARIO	000	001	010	011	100	101	110	111

- **Conversión de octal a binario**.

Para convertir un número de octal a binario se sustituye cada cifra octal por sus correspondientes tres dígitos binarios. Ej:

Pasar a binario el número $127,57_{(8}$

OCTAL	1	2	7	,	5	7
BINARIO	001	010	111	,	101	111

Resultado: $001010111,101111_{(2}$

- **Conversión de binario a octal**.

Para convertir un número de binario a octal se sustituyen cada 3 cifras binarias (comenzando por la derecha) por su correspondiente dígito octal.

Ej: Pasar a octal el número $011100101,000001_{(2}$

OCTAL	011	100	101	,	000	001
BINARIO	3	4	5	,	0	1

Resultado: $345,01_{(8}$

1.3.3 SISTEMA HEXADECIMAL

El hexadecimal es un sistema posicional con base 16 (símbolos 0, 1, 2, 3, 4, 5, 6, 7, A., B, C, D, E, F). Cada dígito hexadecimal se corresponde con 4 dígitos binarios.

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

- **Conversión de hexadecimal a binario**.

Para convertir un número de hexadecimal a binario se sustituye cada cifra hexadecimal por sus correspondientes cuatro dígitos binarios. Ej:

Pasar a binario el número $A2F,E4_{(16}$

HEXAD.	A	2	F	,	E	4
BINARIO	1010	0010	1111	,	1110	0100

Resultado: $101000101111,11100100_{(2}$

- **Conversión de binario a hexadecimal**.

Para convertir un número de binario a hexadecimal se sustituyen cada 4 cifras binarias (comenzando por la derecha) por su correspondiente dígito hexadecimal. Ej:

Pasar a hexadecimal el número $101110100101,00001001_{(2}$

OCTAL	1011	1010	0101	,	0000	1001
BINARIO	B	A	5	,	0	9

Resultado: $BA5,09_{(16}$

1.3.4 REPRESENTACIÓN DE NÚMEROS ENTEROS

Un ordenador maneja números en binario con una limitación de longitud referida al número de bits, y además necesita considerar el signo para poder operar con números negativos. Las cuatro formas habituales que se utilizan para estas representaciones son:

- Módulo y signo
- Complemento a 1 (C-1)
- Complemento a 2 (C-2)
- Exceso a 2 elevado a N – 1 (exceso 2^{N-1})

representaremos el <u>número de dígitos disponibles</u> por **N**.

Rango de Representación en un sistema es el conjunto de números representables en el mismo.

• MÓDULO Y SIGNO.

En este sistema de representación, el bit que está situado más a la izquierda representa el signo, y su valor es 0 para el signo + y 1 para el signo -. El resto de bits (**N-1**) representan el **módulo** del número.

Ej, disponiendo de 8 bits (N=8)

Nº 23

signo	64	32	16	8	4	2	1
0	0	0	1	0	1	1	1

Nº -115

signo	64	32	16	8	4	2	1
1	1	1	1	0	0	1	1

El **rango** de representación es:

$$-2^{N-1} + 1 <= X <= 2^{N-1} - 1$$

Ventaja: Rango simétrico.

Inconveniente: dos representaciones para el 0. Ej., con 8 bits:

+0

signo	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

-0

signo	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

• COMPLEMENTO A UNO (C-1).

En este sistema de representación, el bit que está situado más a la izquierda representa el signo, y su valor es 0 para el signo + y 1 para el signo -. Para los números positivos, los N-1 bits de la derecha representan el módulo del número. El negativo se obtiene complementando todos los dígitos del positivo correspondiente (cambiando ceros por unos y viceversa) incluido el bit de signo.

Ej, disponiendo de 8 bits (N=8)

Nº 23

signo	64	32	16	8	4	2	1
0	0	0	1	0	1	1	1

Nº -115

signo	64	32	16	8	4	2	1
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0

El **rango** de representación es:

$$-2^{N-1} + 1 <= X <= 2^{N-1} - 1$$

Ventaja: Rango simétrico.

Inconveniente: dos representaciones para el 0. Ej., con 8 bits:

+0	signo	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0

-0	signo	64	32	16	8	4	2	1
	1	1	1	1	1	1	1	1

• COMPLEMENTO A 2 (C-2).

En este sistema de representación, el bit que está situado más a la izquierda representa el signo, y su valor es 0 para el signo + y 1 para el signo -. Para los números positivos, los N-1 bits de la derecha representan el módulo del número. El negativo se obtiene en dos pasos:

1- Se complementa el número positivo en todos sus bits incluido el de signo.

2- Al resultado obtenido se le suma 1 (en binario), despreciando el acarreo si existe.

Ej, disponiendo de 8 bits (N=8)

Nº 23	signo	64	32	16	8	4	2	1
	0	0	0	1	0	1	1	1

Nº -115	signo	64	32	16	8	4	2	1
	0	1	1	1	0	0	1	1
	1	0	0	0	1	1	0	0
	+							1
	1	0	0	0	1	1	0	1

El **rango** de representación es:

$$-2^{N-1} <= X <= 2^{N-1} - 1$$

Ventaja: una única representación para el 0. Ej., con 8 bits:

+0	signo	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0

-0	signo	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1
	+							1
	0	0	0	0	0	0	0	0

(despreciamos el acarreo final)

Inconveniente: Rango asimétrico.

- **EXCESO 2 ELEVADO A N-1.**

En este sistema de representación, el bit que está situado más a la izquierda representa el signo, y su valor es **1** para el signo **+** y **0** para el signo **-**. Para obtener la representación de un número x en este sistema, se suma a dicho número N-1, y se escribe en binario natural. Se observa que todo número representado en exceso es igual a su correspondiente representación en C-2 con el primer dígito de la izquierda cambiado.
Ej, disponiendo de 8 bits (N=8)

Nº 23
N-1 = 7 $2^{N-1} = 128$
Resultado: 23 + 128 = 151

signo	64	32	16	8	4	2	1
1	0	0	1	0	1	1	1

Nº -115
N-1 = 7 $2^{N-1} = 128$
Resultado: -115 + 128 = 13

signo	64	32	16	8	4	2	1
0	0	0	0	1	1	0	1

El **rango** de representación es:

$$-2^{N-1} <= X <= 2^{N-1} - 1$$

Ventaja: una única representación para el 0. Ej., con 8 bits:

+0 (0+128)

signo	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

-0 (-0+128)

signo	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

Inconveniente: Rango asimétrico.

1.3.5 CÓDIGOS BCD

El código BCD (Decimal Codificado en Binario) utiliza un cuarteto o **nibble** (4 bits) para la representación de cada cifra decimal (del 0 al 9). Existen varias versiones de este código, algunos de los cuales son:

- BCD Natural
- BCD Aiken
- BCD Exceso 3

- **BCD Natural.**

Es un sistema que codifica cifra a cifra del 0 al 9 con 4 bits utilizando para cada una su correspondiente valor binario.

DECIMAL	BCD NATURAL
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

E: el número 14 en binario puro y en BCD Natural:.

Binario: 1111

BCD natural 0001 0101

- **BCD Aiken.**

En este sistema se trata de conseguir una simetría en la representación (esto hace que sea un código muy apropiado para operaciones de resta y división. Cada cifra es el complemento a 1 de su cifra simétrica (el 5 del 4, el 6 del 3, etc.).

DECIMAL	BCD AIKEN
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

E: el número 15 en binario puro y en BCD Aiken:

Binario: 1111

BCD aiken 0001 1011

- **BCD Exceso 3.**

En este sistema se trata también de conseguir simetría en la representación Cada cifra se codifica en sumándole tres a su valor y escribiendo el resultado en binario.

DECIMAL	BCD exceso 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

E: el número 15 en binario puro y en BCD Exceso 3:

Binario: 1111

BCD aiken 0001 1000

Sistemas de codificación basados en BCD.

- **DESEMPAQUETADO**

Para representar un número decimal se almacena cada dígito de dicho número en un byte. En cada byte el nibble de la izquierda se denomina **ZONA** y está compuesto por cuatro unos, salvo el del dígito menos representativo, cuyo nibble izquierdo se utiliza para almacenar el signo del número: $\mathbf{C_{(16}}$ si es positivo o $\mathbf{D_{(16}}$ si es negativo.

Ej. $-2354_{(10}$

zona	2	zona	3	zona	5	signo	4
1111	0010	1111	0011	1111	0101	1101	0100

- **EMPAQUETADO**

En este sistema se elimina el nibble de zona, salvo el de la última cifra. Ahora en cada byte representaremos dos dígitos del número decimal, salvo en el último, en el cual el nibble derecho se utilizará para representar el signo. El posible nibble sobrante por la izquierda, lo rellenamos con ceros.

Ej. $-2354_{(10}$

	2	3	5	4	signo
0000	0011	0011	0101	0100	1101

1.3.6 REPRESENTACIÓN DE NÚMEROS FRACCIONARIOS

Vamos a ver la representación de números fraccionarios utilizando el formato de **coma flotante**. Este formato está basado en la notación científica o exponencial, permitiendo el tratamiento de números muy grandes y muy pequeños.

En notación científica un número se representa en la forma:

$$\mathbf{N^o = mantisa \times base\ exponenciación}^{\text{exponente o característica}}$$

Donde la **mantisa** será un número fraccionario con signo, el **exponente** un número entero con signo.

Hay muchas formas de representar un número siguiendo esta notación, pero la que se suele utilizar es la **normalizada**.

Decimos que una representación en notación científica está **Normalizada** cuando la mantisa no tiene parte entera y el primer dígito a la derecha de la coma es distinto de cero (salvo que el número a representar sea el cero).

Ej: $125{,}34_{(10}$

$$125{,}34 = 12{,}534 \times 10 = 1{,}2534 \times 10^2 = 0{,}125 \times 10^3$$

por tanto la representación normalizada sería: **$0{,}125 \times 10^3$**

Hay además muchos formatos de representación en coma flotante, en función de la longitud, la base de exponenciación, el número de bits reservados para la mantisa y para el exponente, el sistema elegido para representar la mantisa y el exponente, etc.

Normalmente se suele seguir las siguientes reglas:

- El **exponente** se representa en MS o en exceso a 2N-1 (siendo **N** el número de bits reservados para su representación)
- La **mantisa** es la parte fraccionaria del número representado normalizado, con la coma implícita a la izquierda, representada en MS, C-1 o C-2. El signo se suele representar en el bit más a la izquierda.
- La **base** de exponenciación es una potencia de 2 (2, 8 o 16).

El **rango** de representación es:

Donde:

- **mNN** es el mínimo número negativo representable. Cuyo valor será:

$$mNN = -\text{máxima mantisa} \times base^{\text{máximo exponente}}$$

- **MNN** es el máximo número negativo representable. Cuyo valor será:

$$MNN = -\text{mínima mantisa} \times base^{-\text{máximo exponente}}$$

- **mNP** es el mínimo número positivo representable. Cuyo valor será:

$$mNP = \text{mínima mantisa} \times base^{-\text{máximo exponente}}$$

- **MNP** es el máximo número positivo representable. Cuyo valor será:

$$MNP = \text{máxima mantisa} \times base^{\text{máximo exponente}}$$

Las zonas que están rayadas en el dibujo representan el intervalo de número representables (además del cero). Hay que tener en cuenta, sin embargo, que estas zonas no son continuas, es decir existen huecos o números no representables en ellas, que son aquellos que superan la precisión del formato escogido (esto es, si utilizamos 32 bits

para representar la mantisa de los números en coma flotante, todos aquellos con una mantisa superior perderían precisión al ser representados).

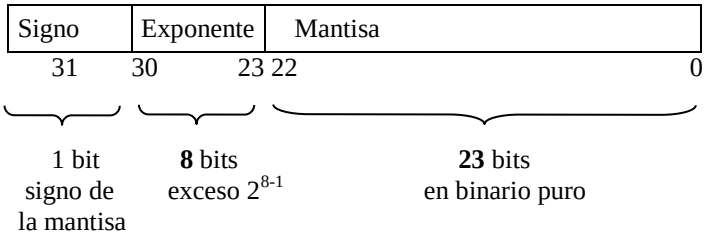
Hay además zonas de las que no se pueden representar ningún número, que toman los nombres:

- **Overflow** o Desbordamiento **positivo**: Números mayores que MNP
- **Underflow** o Subdesbordamiento **positivo**: números entre 0 y mNP
- **Underflow** o Subdesbordamiento **negativo**: números entre MNN y 0
- **Overflow** o Desbordamiento **negativo**: Números menores que mNN

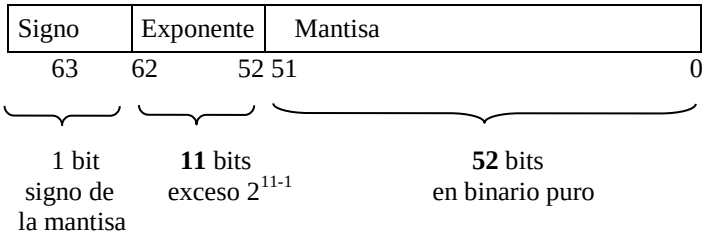
FORMATOS ESTÁNDAR

Existen unos formatos normalizados de coma flotante para ordenadores de **32 bits**. Es el estándar **IEEE P754**.

- Para **simple precisión**



- Para **doble precisión**



Ej: representar $-125{,}56_{(8}$ en simple precisión y $FF345{,}BAAB34567_{(16}$ en doble precisión.

Primero pasamos los número a base 2:

$-125{,}56_{(8} = -001\ 010\ 101\ ,\ 101\ 110_{(2}$

$FF345{,}BAAB34567_{(16} = 1111\ 1111\ 0011\ 0100\ 0101\ ,\ 1011\ 1010\ 1010\ 1011\ 0011\ 0100\ 0101\ 0110\ 0111_{(2}$

Lo siguiente es normalizarlos:

$-125{,}56_{(8} = -0\ ,\ 1\ 010\ 101\ 101\ 110\ \times\ 10^{111}{}_{(2}$

$FF345{,}BAAB34567_{(16} = 0,\ 11111111001101000101101110101010101100110100010101100111 \times 10^{10100}{}_{(2}$

Nos quedaría entonces:

Simple precisión:

-0 , 1 010 101 101 110 x $10^{111}_{(2}$

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

Los bits sobrantes por la derecha en la representación de la mantisa se rellenan con ceros.

Doble precisión:

0, 1111111100110100010110111010101010110011010001010111 x $10^{10100}_{(2}$

0	10000010100	1111111100110100010110111010101010110011010001010110

Tenemos 56 bits en la mantisa, pero sólo podemos representar 52 en este formato, con lo cual perdemos precisión al desechar los cuatro menos significativos.

1.4 REPRESENTACIÓN ALFANUMÉRICA

La **información** (conjunto de datos con significado propio constituido por cadenas de caracteres -cifras, letras y caracteres especiales-) con la que trabajamos necesita ser transformada para que pueda ser tratada por medio de ordenadores.

La información que queremos que pueda ser tratada por un ordenador se presenta en un determinado sistema de representación que utiliza un alfabeto (**alfabeto de entrada**), y por medio de un sistema de codificación la transformaremos en una información codificada que utiliza su correspondiente **alfabeto de salida** y que será directamente reconocible y tratable por la máquina.

En los ordenadores se utilizan sistemas de codificación binarios numéricos, como por ejemplo los BCD del punto 2.6, y alfanuméricos.

Los **códigos alfanuméricos** nos permiten codificar información no exclusivamente numérica. En ellos solemos tener las siguientes características:

- **Conjunto de caracteres**:

– Las 10 cifras del sistema decimal (del 0 al 9).
– Las letras del alfabeto.
– Los signos de puntuación (., : ; ...).
– Caracteres de control.

- **Longitud de un código binario**. Es el número de bits que utiliza para codificar un Carácter.

- **Número máximo** del conjunto de caracteres. $\mathbf{2^{longitud}}$

Un código en el que no se utilicen todas las combinaciones posibles ($2^{longitud}$) se dice que es **REDUNDANTE.**

Los primeros códigos utilizados fueron los de 6 bits. Permitían por tanto la representación de 64 caracteres.
Como ejemplo de estos tenemos:

• CODIGO ASCII

Significa American Estándar Code for Information Interchange. Es uno de los más utilizadso y tiene versiones de 7 y de 8 bits. Vamos a ver la de 7 bits.

- Letras mayúsculas y minúsculas.
- Las diez cifras decimales.
- 32 caracteres de control
- caracteres especiales (signos de puntuación, etc.).

Bits 3210 / 654 000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P	`	P
0001	SOH	DC1	!	1	A	Q	a	Q
0010	STX	DC2	“	2	B	R	b	R
0011	ETX	DC3	#	3	C	S	c	S
0100	EOT	DC4	$	4	D	T	d	T
0101	ENQ	NAK	%	5	E	U	e	U
0110	ACK	SYN	&	6	F	V	f	V
0111	BEL	ETB	‘	7	G	W	g	W
1000	BS	CAN	(	8	H	X	h	X
1001	HT	EM	)	9	I	Y	i	Y
1010	LF	SUB	*	:	J	Z	j	Z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Como se puede ver en la tabla, la representación de los dígitos decimales se corresponde con su codificación BCD natural, anteponiendo los bits 011

El significado de los caracteres de control es el siguiente:

NULL	Null (carácter nulo)
SOH	Start Of Heading (comienzo de cabecera)
STX	Start Of Text (comienzo de texto)
ETX	End Of Text (fin de texto)
EOT	End Of Transmission (fin de transmisión
ENQ	Enquery (petición de transmisión)
ACK	Acknowledge (reconocimiento de transmisión)
BEL	Bell (señal acústica)
BS	Backspace (retroceso)
HT	Horizontal Tabulation (tabulación horizontal)
LF	Line Feed (avance de línea)
VT	Vertical Tabulation (tabulación vertical)
FF	Form Feed (avance de página)
CR	Carriage Return (retorno de carro)
SO	Shift Out (quitar desplazador de bits)
SI	Shift In (poner desplazador de bits)
DLE	Data Link Escape (escape de enlace de datos)
DC1	Device Control 1 (control de dispositivo 1)
DC2	Device Control 2 (control de dispositivo 2)
DC3	Device Control 3 (control de dispositivo 3)
DC4	Device Control 4 (control de dispositivo 4)
NAK	Negative Acknowledge (transmisión negativa)

SYN	Synchronous idle (espera síncrona)
ETB	End of Transmission Block (fin bloque de transmisión)
CAN	Cancel (cancelar)
EM	End of Medium (final de medio)
SUB	Substitute (sustitución)
ESC	Escape (escape)
FS	File Separator (separador de ficheros)
GS	Group Separator (separador de grupos)
RS	Record Separator (separador de registros)
US	Unit Separator (separador de unidades)
DEL	Delete (borrar)

Ej. De codificación en ASCII.

1000001	1001101	1000001	1000100	1001111	1010010
A	M	A	D	O	R

Con 8 bits, se denomina ASCII extendido, y surge el concepto de **Páginas de Códigos.** La 850 para españa y la 437 para USA en MS/DOS, o la ISO8859-1 o Latín1 o ANSI en otros S.O.

- **UNICODE**

Un problema con sistemas como el ASCII es precisamente el hecho de que haya diferentes sistemas de codificación de caracteres para cada idioma, cada uno usando los mismos números (0-255) para representar los caracteres de ese lenguaje. Esto hace que intercambiar documentos entre estos sistemas sea difícil porque no hay manera de saber con certeza qué esquema de codificación de caracteres había usado el autor del documento; el ordenador sólo ve dígitos binarios, los cuales pueden significar muchas cosas. Almacenar estos documentos en un mismo sitio (como en una tabla de una base de datos); necesitaría almacenar además el tipo de codificación junto con cada texto, y asegurarse de adjuntarlo con el texto cada vez que accediese a él.

Unicode es un sistema de representación que asigna un único número pada cada carácter, independientemente del programa, plataforma o idioma en el que estemos trabajando.

Unicode define tres formas de codificación bajo el nombre UTF o Formato de Transformación Unicode (Unicode Transformation Format):8

- **UTF-8** codificación orientada a byte con símbolos de longitud variable.
- **UTF-16** codificación de 16 bits de longitud.
- **UTF-32** codificación de 32 bits de longitud fija.

UTF-8 (8-bit Unicode Transformation Format) es un formato de codificación de caracteres que utiliza símbolos de longitud variable

Sus características principales son:

- Es capaz de representar cualquier carácter Unicode.
- Usa símbolos de longitud variable (de 1 a 4 bytes por carácter Unicode).
- Incluye la especificación US-ASCII de 7 bits, por lo que cualquier mensaje ASCII se representa sin cambios.
- Incluye sincronía. Es posible determinar el inicio de cada símbolo sin reiniciar la lectura desde el principio de la comunicación.
- No superposición. Los conjuntos de valores que puede tomar cada byte de un caracter multibyte, son disjuntos, por lo que no es posible confundirlos entre sí.

1.5 DETECCIÓN Y CORRECCIÓN DE ERRORES

A la hora de transmitir información entre ordenadores, en las líneas de transmisión, así como al codificar y decodificar la información, pueden aparecen errores que es necesario detectar y, en la medida de lo posible, corregir. Para ello se utilizan una serie de códigos detectores y correctores de errores.

En cualquiera de estos códigos vamos a utilizar el concepto de "Distancia":

Distancia entre dos combinaciones binarias es el número de bits que deben ser modificados en una de las combinaciones para obtener la otra.

Ej:

	b7	b6	b5	b4	b3	b2	b1
A	0	1	0	1	0	1	1
B	0	0	0	1	1	1	0

La distancia es 3, pues varían tres bits (b6, b3 y b1).

Distancia de un código binario es la menor de las distancias entre dos combinaciones binarias cualesquiera del mismo.

Para que un código pueda detectar errores su distancia tiene que ser superior a la unidad, ya que si es l, los errores de un bit pueden llevar a otra combinación válida sin que se pueda detectar.

Para que un código sea capaz de corregir errores de 1 bit, debe tener una distancia de 3. En general, para corregir **errores** de ***n* bits** se necesitará un código cuya distancia sea al menos de ***2n* + l**.

Vamos a ver un ejemplo de códigos detectores: Códigos de paridad. Y un ejemplo de códigos correctores: Códigos Hamming.

• CODIGOS DE PARIDAD

Son códigos detectores de errores cuya distancia es **2**, por lo que permiten controlar errores de 1 bit. Para obtenerlos partimos de otro código al que añadiremos un bit más, que se denomina bit de paridad. El valor de este bit en cada combinación del código, estará en función del número de bits puestos a uno en dicha combinación. Podemos seguir dos criterios:

- **Paridad par**: Si el número de unos es impar, ponemos un uno en el bit de paridad y un cero en caso contrario.
- **Paridad impar**: Si el número de unos es impar, ponemos un cero en el bit de paridad y un cero en caso contrario.

Ej. **Código BCD Natural con paridad par**:

Decimal	BCD Natural	Paridad
0	0000	0
1	0001	1
2	0010	1
3	0011	0
4	0100	1
5	0101	0
6	0110	0
7	0111	1
8	1000	1

9	1001	0

- **CODIGOS HAMMING**

Son códigos correctores de errores cuya distancia mínima es **3**. Permiten detectar errores de **2 bits** y corregir errores de **1 bit**.

Se forman añadiendo a1 código a proteger una serie de bits para detectar varias paridades. El conjunto de bits que se añaden forman un número en binario puro que indica la posición del bit erróneo. Si no hay error, el número será 0.

Siendo **n** el número de bits del código del que partimos y **p** los bits que añadiremos para la detección y corrección de errores, se debe cumplir que:

$$\mathbf{2^p \geq n + p + 1}$$

Ej. **Código Hamming para BCD Natural**:

En BCD natural el número de bis es 4 (**n**=4) por tanto **p** deberá valer 3.

$$2^3 \geq 4 + 3 + 1$$

numerando los bits (de izquierda a derecha) de 1 a 7, los bits 7, 6, 5 y 3 serán los correspondientes a la representación BCD. Los bits 4, 2 y 1 serán los bits de control cuyo valor calcularemos de la siguiente manera:

b4 será un bit de paridad par sobre los bits b7, b6 y b5

b2 será un bit de paridad par sobre los bits b7, b6 y b3

b1 será un bit de paridad par sobre los bits b7, b5 y b3

para comprobar la posible incorrección de una representación formaremos un número en binario de tres bits de longitud (c3 c2 c1), fijando el valor de sus bits del siguiente modo:

c3 será 1 si la hay error en la paridad controlada por b4
c2 será 1 si la hay error en la paridad controlada por b2
c1 será 1 si la hay error en la paridad controlada por b1

el número así formado, en caso de ser cero indicará que no hubo error, y en caso contrario indicará el número de bit erróneo.

	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	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0

Vamos a comprobarlo con un ejemplo. Se transmite 0101101 (5) y se recibe 0101001 que no es ninguna de las combinaciones válidas del código.

b7	b6	b5	b4
0	1	0	1

La paridad es par, por tanto C3 = 0

b7	b6	b3	b2
0	1	0	0

La paridad es impar, por tanto C2 = 1

b7	b5	b3	b1
0	0	0	1

La paridad es impar, por tanto C1 = 1

De c3 c2 y c1 obtenemos el número $011_{(2}$ o sea el $3_{(10}$, por tanto el bit erróneo es el número tres, que en vez de un 0 debería contener un 1.