



Representación de la Información en los Sistemas Digitales

© Luis Entrena, Celia López,
Mario García, Enrique San Millán

Universidad Carlos III de Madrid

Introducción a los computadores

- Computador: Máquina que procesa información



Sistemas analógicos y digitales

- **Sistemas analógicos:** aquellos cuyas variables toman valores continuos en el tiempo
 - Las magnitudes físicas son en su mayoría analógicas
- **Sistemas digitales:** aquellos cuyas variables toman valores discretos en el tiempo
 - Se utilizan valores discretos llamados dígitos
 - Precisión limitada
 - Las cantidades digitales son más fáciles de manejar
 - Las magnitudes analógicas se pueden convertir a magnitudes digitales mediante muestreo

Sistemas analógicos y digitales

- Sistema Analógico



- Sistema digital



Sistemas binarios

- Sistemas binarios: sistemas digitales que sólo utilizan dos posibles valores
 - Los dígitos binarios se denominan bits (Binary digiT)
 - Se representan mediante los símbolos 0 y 1, ó L y H
 - Los sistemas binarios son casi los únicos utilizados. Por extensión, se utiliza el término digital como sinónimo de binario
- ¿Por qué binario?
 - Más fiable: mayor inmunidad frente al ruido
 - Más sencillo de construir: sólo hay que distinguir entre dos valores

Índice

- Sistemas de Numeración
- Conversiones entre sistemas de numeración
- Códigos Binarios:
 - Códigos BCD
 - Códigos progresivos y cíclicos
 - Códigos alfanuméricos
 - Códigos detectores y códigos correctores de errores
 - Representación de números enteros y reales

Sistemas de Numeración

- Permiten representar los números mediante dígitos
- El sistema que utilizamos habitualmente es el sistema decimal:
 - $N = a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_1 10 + a_0$
 - Ejemplo: $272_{10} = 2 \cdot 10^2 + 7 \cdot 10 + 2$
- Se puede hacer lo mismo pero utilizando bases diferentes a 10:

$$N = a_n b^n + \underbrace{a_{n-1}}_{\text{Dígito}} \underbrace{b^{n-1}}_{\text{Peso}} + \dots + a_1 b + a_0$$

Base

Sistemas de Numeración

- En un sistema con base b los dígitos posibles son:
 - $0, 1, \dots, b-1$
- Con n dígitos se pueden representar b^n números posibles, desde el 0 hasta el b^n-1
- Esta representación sirve también para números que no sean naturales:
 - Ejemplo: $727,23_{10} = 7 \cdot 10^2 + 2 \cdot 10 + 7 + 2 \cdot 10^{-1} + 2 \cdot 10^{-2}$
- Los sistemas que se utilizan en los sistemas digitales son: binario ($b=2$), octal ($b=8$) y hexadecimal ($b=16$)

Sistema Binario

- En este sistema la base es 2. Permiten representar perfectamente la información en los sistemas digitales.
 - Los dígitos posibles son 0 y 1. Un dígito en sistema binario se denomina “bit”.
 - Con n bits se pueden representar 2^n números
- El bit de mayor peso se denomina **bit más significativo** o **MSB** (“Most Significant Bit”), y el bit de menor peso se denomina **bit menos significativo** o **LSB** (“Least Significant Bit”)

← Habitualmente el MSB se escribe a la izquierda y el LSB a la derecha

- Ejemplo: $\overset{\text{MSB}}{\textcircled{1}}00101\underset{\text{LSB}}{\textcircled{0}}_2 = 1 \cdot 2^6 + 1 \cdot 2^3 + 1 \cdot 2^1 = 74_{10}$

Sistema Octal

- En este sistema la base es 8.
 - Los dígitos son 0,1,2,3,4,5,6,7
 - Con n dígitos se pueden representar 8^n números
- Está muy relacionado con el sistema binario (8 es una potencia de 2, en concreto $2^3=8$)
 - Esto permite convertir fácilmente de octal a binario y de binario a octal

- Ejemplo:

$$137_8 = 1 \cdot 8^2 + 3 \cdot 8^1 + 7 \cdot 8^0 = 95_{10}$$

Sistema Hexadecimal

- En este sistema la base es 16.
 - Los dígitos son 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
 - Está relacionado con el sistema binario ($2^4=16$)
 - Un dígito hexadecimal permite representar lo mismo que 4 bits (ya que $2^4=16$). Un dígito hexadecimal se denomina también “**nibble**”.
 - Dos dígitos hexadecimales equivalen por tanto a 8 bits. El conjunto de 8 bits o dos dígitos hexadecimales, se denomina “**byte**”.
- Notaciones: $23AF_{16} = 23AF_{\text{hex}} = 23AFh = 0x23AF = 0x23\ 0xAF$.
- Ejemplo: $23AFh = 2 \cdot 16^3 + 3 \cdot 16^2 + 10 \cdot 16 + 15 = 9135_{10}$

Conversiones entre sistemas de numeración

- Pasar de cualquier sistema a sistema decimal:

- $N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0$
- Ejemplos:
 - $1001010_2 = 1 \cdot 2^6 + 1 \cdot 2^3 + 1 \cdot 2^1 = 74_{10}$
 - $137_8 = 1 \cdot 8^2 + 3 \cdot 8^1 + 7 \cdot 8^0 = 95_{10}$
 - $23AF_{16} = 2 \cdot 16^3 + 3 \cdot 16^2 + 10 \cdot 16 + 15 = 9135_{10}$

- Para pasar de decimal a otro sistema:

- Método de descomposición en pesos
- Método de divisiones sucesivas por la base

Método de descomposición en pesos

- Consiste en descomponer el número en potencias de la base.
 - Se busca la potencia de la base (menor) más cercana al número.
 - Se van buscando potencias sucesivamente para que la suma de todas ellas sea el número decimal que se quiere convertir.
 - Finalmente los pesos de las potencias utilizadas se utilizan para representar el número en la base buscada.
- Éste método es útil sólo para sistemas donde las potencias de la base son conocidas. Por ejemplo para sistema binario: 1, 2, 8, 16, 32, 64, 128, 256, ...
- Ejemplo:
 - $25_{10} = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 = 11001_2$

Método de divisiones sucesivas por la base

- Consiste en dividir el número decimal a convertir sucesivamente por la base y los cocientes obtenidos en las divisiones anteriores
 - El último cociente obtenido es el MSB del resultado
 - Los restos obtenidos son el resto de dígitos, siendo el primero de los restos obtenidos el LSB

- Ejemplo:

$$\begin{array}{r}
 25 \overline{) 2} \\
 \underline{1} \\
 12 \overline{) 2} \\
 \underline{0} \\
 6 \overline{) 2} \\
 \underline{0} \\
 3 \overline{) 2} \\
 \underline{1} \\
 1
 \end{array}
 \rightarrow 25_{10} = 11001_2$$

The diagram shows the successive division of 25 by 2. The remainders are circled and labeled from bottom to top: 1, 1, 0, 0, 1. A red arrow points from the bottom remainder (1) to the label 'LSB' (Least Significant Bit). Another red arrow points from the top remainder (1) to the label 'MSB' (Most Significant Bit).

- Este método es más general que el anterior. Sirve para convertir de decimal a cualquier otra base.

Conversión de números reales

- La conversión de binario a decimal se hace igual que para números enteros (utilizando pesos negativos para la parte decimal):

$$\begin{aligned} 101,011_2 &= 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ &= 4 + 1 + 0,25 + 0,125 = 5,375_{10} \end{aligned}$$

- La conversión de decimal a binario, se hace en dos partes:
 - Se convierte primero la parte entera por el método de divisiones sucesivas por la base (o por descomposición de pesos)
 - Luego se convierte la parte decimal por un método análogo, multiplicaciones sucesivas por la base.

Método de multiplicaciones sucesivas por la base (parte decimal)

- Consiste en multiplicar la parte decimal del número por la base sucesivamente.
 - Se multiplica la parte decimal del número por 2. La parte entera del resultado es el primer dígito (MSB de la parte decimal) de la conversión
 - Se vuelve a tomar la parte decimal, y se multiplica por 2 otra vez, y nuevamente la parte entera es el siguiente dígito.
 - Se itera tantas veces como se quiera, según la precisión que se quiera obtener en la conversión.
- Ejemplos:

$$0,3125_{10} = 0,0101_2$$

$$0,3125 \times 2 = 0,625 \Rightarrow 0$$

$$0,625 \times 2 = 1,25 \Rightarrow 1$$

$$0,25 \times 2 = 0,5 \Rightarrow 0$$

$$0,5 \times 2 = 1 \Rightarrow 1$$

$$0,1_{10} = 0,000110011 \dots_2$$

$$0,1 \times 2 = 0,2 \Rightarrow 0$$

$$0,2 \times 2 = 0,4 \Rightarrow 0$$

$$0,4 \times 2 = 0,8 \Rightarrow 0$$

$$0,8 \times 2 = 1,6 \Rightarrow 1$$

$$0,6 \times 2 = 1,2 \Rightarrow 1$$

$$0,2 \times 2 = 0,4 \Rightarrow 0 \text{ <- se repiten las cuatro cifras, periódicamente}$$

$$0,4 \times 2 = 0,8 \Rightarrow 0$$

$$0,8 \times 2 = 1,6 \Rightarrow 1$$

...

Otros métodos de conversión

- Los sistemas octal y hexadecimal están relacionados con el binario, ya que sus bases son potencias exactas de 2 (la base binaria). Esto permite convertir entre estos sistemas de forma muy sencilla:
 - **OCTAL a BINARIO:** Convertir cada dígito en 3 bits
 - Ejemplo: $735_8 = \underbrace{111}_7 \underbrace{011}_3 \underbrace{101}_5_2$
 - **BINARIO a OCTAL:** Agrupar en grupos de 3 bits y convertirlos de forma independiente a octal.
 - Ejemplo: $\underbrace{1}_1 \underbrace{011}_3 \underbrace{100}_4 \underbrace{011}_3_2 = 1343_8$
 - **HEXADECIMAL a BINARIO:** Convertir cada dígito en 4 bits
 - Ejemplo: $3B2h = \underbrace{0011}_3 \underbrace{1011}_B \underbrace{0010}_2_2$
 - **BINARIO a HEXADECIMAL:** Agrupar en grupos de 4 bits y convertirlos de forma independiente a octal
 - Ejemplo: $\underbrace{1010}_{Ah} \underbrace{1110}_{Eh} \underbrace{0011}_{3h}_2 = 2E3h$


Códigos Binarios

- Los códigos binarios son códigos que utilizan únicamente 0s y 1s para representar la información
- La información que se puede representar con códigos binarios puede ser de múltiples tipos:
 - Números naturales
 - Números enteros
 - Números reales
 - Caracteres alfabéticos y otros símbolos
- Una misma información (por ejemplo un número natural) se puede representar utilizando diferentes códigos.
 - Es importante especificar siempre qué código que se está utilizando cuando se representa una información en un código binario.

Código Binario Natural

- Es un código binario en el que se representa un número natural mediante su representación en sistema binario
 - Es el código binario más simple
 - Aprovecha que la representación en sistema binario de un número natural utiliza únicamente 0s y 1s
- Notación: Utilizaremos el indicador “BIN” para indicar que un código binario es el código binario natural:
 - $1001_{\text{BIN}} = 1001_2$

Códigos BCD (“Binary-Coded Decimal”)

- Permiten representar números naturales de una forma alternativa al binario natural.
- Se asigna un código de 4 bits a cada dígito decimal. Un número decimal se codifica en BCD dígito a dígito.
- El código BCD más habitual es el BCD natural  (existen otros códigos BCD).
- Ejemplo:
 - $78_{10} = 0111\ 1000_{\text{BCD}}$
- La codificación BCD de un número no tiene por qué coincidir con el código binario natural:
 - $78_{10} = 1001110_{\text{BIN}}$
- **INCONVENIENTE:** No todas las combinaciones corresponden a un código BCD. Por ejemplo, 1110_{BCD} no existe.
- **VENTAJA:** Facilidad de conversión decimal-binario.

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

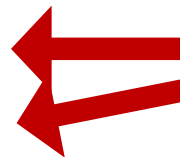
Códigos progresivos y cíclicos

- Dos codificaciones binarias se dice que son **adyacentes** si sólo hay bit diferente entre ambas.
 - **0000** y **0001** son adyacentes, ya que sólo difieren en el último bit
 - **0001** y **0010** no son adyacentes, ya que los dos últimos bits son diferentes
- Un código se dice que es **progresivo** si todas las codificaciones consecutivas son adyacentes.
 - El código binario natural no es progresivo, ya que 0001 y 0010 no son adyacentes.
- Un código se dice que es **cíclico** si además la primera y la última codificación son adyacentes.
- Los códigos progresivos y cíclicos más utilizados son:
 - Código Gray
 - Código Johnson

Código Gray

- El Código Gray es un código progresivo y cíclico
- Ejemplo de Código Gray de 3 bits:

Decimal	Código Gray
0	0 0 0
1	0 0 1
2	0 1 1
3	0 1 0
4	1 1 0
5	1 1 1
6	1 0 1
7	1 0 0



Todas las
codificaciones
consecutivas son
adyacentes

Código Gray

- Construcción del código Gray de n bits:
 - Primero se copian los códigos de n-1 bits y se añaden otros n-1 copiando los anteriores en orden inverso
 - Luego se añade un cero a la izquierda en los de arriba y un uno en los de abajo

- Código de 1 bit:

0
1

- Código de 2 bits:

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



Código Gray

- Código de 3 bits:

0 0		0 0 0
0 1		0 0 1
1 1		0 1 1
1 0		0 1 0
<hr/>		<hr/>
1 0	→	1 1 0
1 1		1 1 1
0 1		1 0 1
0 0		1 0 0

- Iterando se pueden construir los códigos Gray de n bits

Conversión entre los códigos Gray y Binario Natural

Se puede convertir directamente de Gray a Binario y de Binario a Gray, sin necesidad de construir toda la tabla:

BINARIO A GRAY:

$$(A_0 A_1 A_2 \dots A_n)_{\text{BIN}} \rightarrow (B_0 B_1 B_2 \dots B_n)_{\text{GRAY}}$$

- $B_0 = A_0$
- $B_1 = A_0 + A_1$
- $B_2 = A_1 + A_2$
- ...
- $B_n = A_{n-1} + A_n$



Ejemplo:

$$1011_{\text{BIN}} \rightarrow 1110_{\text{GRAY}}$$

GRAY A BINARIO:

$$(A_0 A_1 A_2 \dots A_n)_{\text{GRAY}} \rightarrow (B_0 B_1 B_2 \dots B_n)_{\text{BIN}}$$

- $B_0 = A_0$
- $B_1 = A_1 + B_0$
- $B_2 = A_2 + B_1$
- ...
- $B_n = A_n + B_{n-1}$



Ejemplo:

$$1011_{\text{GRAY}} \rightarrow 1101_{\text{BIN}}$$

BIN	GRAY
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

Código Johnson

- Es otro código progresivo y cíclico
- En cada codificación aparecen agrupados los ceros a la izquierda y los unos a la derecha, o viceversa.
- Ejemplo código Johnson de 3 bits:

Decimal	Johnson		
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0

Códigos alfanuméricos

- Representan símbolos, que pueden ser:
 - Dígitos
 - Letras mayúsculas y minúsculas
 - Signos de puntuación
 - Caracteres de control (espacio, salto de línea, retorno de carro, etc.)
 - Otros símbolos gráficos (operadores matemáticos, etc.)
- Un código alfanumérico mínimo que contenga los 10 dígitos, las 26 letras del alfabeto inglés, mayúsculas y minúsculas (52), necesita al menos 6 bits.
- Los códigos más utilizados en la actualidad son:
 - Código ASCII (7 bits)
 - Códigos ASCII extendidos (8 bits)
 - Códigos unicode (8-32 bits)

Códigos ASCII y ASCII extendidos

- El código ASCII (“American Standard Code for Information Interchange”) fue publicado por primera vez en 1963.
- Es un código de 7 bits (128 códigos) estándar que contiene:
 - Dígitos
 - Letras mayúsculas y minúsculas del alfabeto inglés internacional
 - Signos de puntuación
 - Caracteres básicos de control
- Los códigos ASCII extendidos se utilizan para añadir caracteres adicionales:
 - No son estándar, difieren de una región a otra
 - Los 128 primeros códigos coinciden con el ASCII estándar por compatibilidad

Código ASCII Estándar

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	~	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	DS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Códigos ASCII Extendidos

EJEMPLO:

ACII extendido LATIN-1
(ISO 8859-1)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-																
1-																
2-		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8-																
9-																
A-		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	®	¯	
B-	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C-	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D-	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E-	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F-	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Códigos ASCII Extendidos

EJEMPLO:
ASCII extendido Cirílico
ISO 8859-5

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-																
1-																
2-		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8-																
9-																
A-		Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	-	Ў	Ц
B-	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
C-	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
D-	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
E-	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F-	№	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	џ	џ	џ

Códigos Unicode

- Los códigos Unicode (“Universal Code”) fueron creados en 1991 para tener códigos alfanuméricos estándar, comunes en todas las regiones
 - Se utiliza el mismo código unicode para idiomas Chino, Árabe, etc.
- Como máximo necesitan 32 bits
 - Los primeros 7 bits permiten la compatibilidad con ASCII
 - Con 1 byte se puede representar el código US-ASCII
 - Con 2 bytes: caracteres latinos y alfabetos árabes, griego, cirílico, armenio, hebreo, sirio y thaana.
 - Con 3 bytes: resto de caracteres utilizados en todos los lenguajes
 - Con 4 bytes: caracteres gráficos y poco comunes
- Diferentes versiones de representación. Las más comunes:
 - **UTF-8**: Códigos de 1 byte, pero son de longitud variable (se pueden utilizar 4 grupos de 1 byte para representar un símbolo)
 - **UCS-2**: Códigos de 2 bytes de longitud fija
 - **UTF-16**: Códigos de 2 bytes, de longitud variable (se pueden utilizar 2 grupos de 2 bytes para representar un símbolo)
 - **UTF-32**: Códigos de 4 bytes

Códigos Unicode

EJEMPLO:

- Parte del Unicode correspondiente al alfabeto cirílico

Se necesita el segundo byte para la representación

- Las codificaciones completas se pueden encontrar en:

<http://www.unicode.org/charts>

	Cyrillic															
	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	040A	040B	040C	040D	040E	040F
0	Ё	А	Р	а	р	ё	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
1	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
2	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
3	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
4	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
5	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
6	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
7	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
8	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
9	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
A	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
B	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
C	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
D	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ
E	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ
F	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ

Códigos detectores y correctores de errores

- En los sistemas digitales pueden aparecer errores
 - Errores físicos de los circuitos
 - Interferencias electromagnéticas (EMI)
 - Fallos de alimentación eléctrica
 - Etc.
- Códigos detectores de error:
 - Pueden permitir detectar un error en la codificación
- Códigos correctores de error:
 - Permiten detectar un error y además corregirlo
- Los códigos detectores de error y los códigos correctores de error no utilizan las 2^n posibles codificaciones con n bits

Códigos detectores de error

- Códigos de paridad:
 - Añaden un bit adicional (paridad del número) que permite detectar errores simples en la codificación (error en 1 bit)
 - La paridad que se considera es la de la **suma** de los n bits de la codificación
 - **NOTA:** la paridad no tiene nada que ver con si la codificación binaria es par o impar (un número binario es par si acaba en 0 e impar si acaba en 1).
 - Dos posibles convenios:
 - Añadir un 0 cuando la paridad sea par y 1 cuando sea impar: Se denomina código de paridad par (ya que considerando la suma de los n bits + el bit de paridad la paridad siempre es par)
 - Añadir un 1 cuando la paridad sea par y 0 cuando sea impar: Se denomina código de paridad impar (ya que la suma de los n bits + bit paridad es siempre impar)

Códigos detectores de error

- Ejemplo de paridad:

Código detector de errores (código de paridad impar) a partir del código binario natural de dos bits:



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

- Ejemplo de utilidad del código detector:

Si utilizamos este código en una comunicación entre dos sistemas binarios, el sistema receptor podría detectar si hay un error comprobando la paridad.

Ejemplo: Se transmite la codificación 001, y el receptor recibe la codificación 000 (error en el último bit).

Paridad de 001: impar
Paridad de 000: par



**No coinciden:
Error detectado**

Códigos detectores de error

- Existen más códigos detectores de error:
 - Número de unos:
 - Se añade a la codificación la suma de unos de la codificación (no sólo la paridad de la suma, sino la suma completa)
 - Número de transiciones:
 - Se añade a la codificación el número de transiciones de 0 a 1 y de 1 a 0 en la codificación
 - Códigos CRC (Cyclic Redundancy Checking):
 - Buscan añadir el menor número posible de bits que permitan detectar el mayor número posible de fallos
 - Estos códigos también permiten corregir algunos errores
- Los códigos más utilizados son los de **paridad** (por su sencillez) y **CRC** (por su eficacia)

Códigos correctores de error

- Los códigos correctores permiten no sólo detectar sino también pueden corregir un error.
- Para que un código permita corregir errores, la distancia mínima (número mínimo de bits diferentes entre dos codificaciones) debe ser mayor de 2.
 - Se puede corregir la codificación buscando la codificación más cercana perteneciente al código
- Hamming describió un método general para construir códigos con distancia mínima de 3, conocidos como **códigos de Hamming**
- Estos códigos son importantes, a partir de ellos se obtienen muchos de los utilizados en sistemas de comunicaciones (por ejemplo los códigos de bloque Reed-Solomon)

Codificación de números enteros y reales

- Además de los códigos binarios vistos hasta ahora, hay otros códigos importantes que se utilizan para representar números enteros y números reales:
 - Números enteros: Códigos de signo y magnitud, Complemento a Uno, Complemento a Dos
 - Números reales: Códigos de Punto fijo y Coma Flotante
- Estos códigos se estudiarán en detalle en el Tema 4: Aritmética Binaria

Referencias

- Fundamentos de Sistemas Digitales. Thomas L. Floyd. Pearson Prentice Hall
- Introducción al Diseño Lógico Digital. John P. Hayes. Addison-Wesley
- Diseño Digital. John F. Wakerly. Pearson Prentice Hall