

Arquitectura de Computadoras

Unidad 0
Códigos

Martín Ferreyra Biron-Edgardo Gho
Carlos Rodriguez



UM 1082 LAZ/3

82225

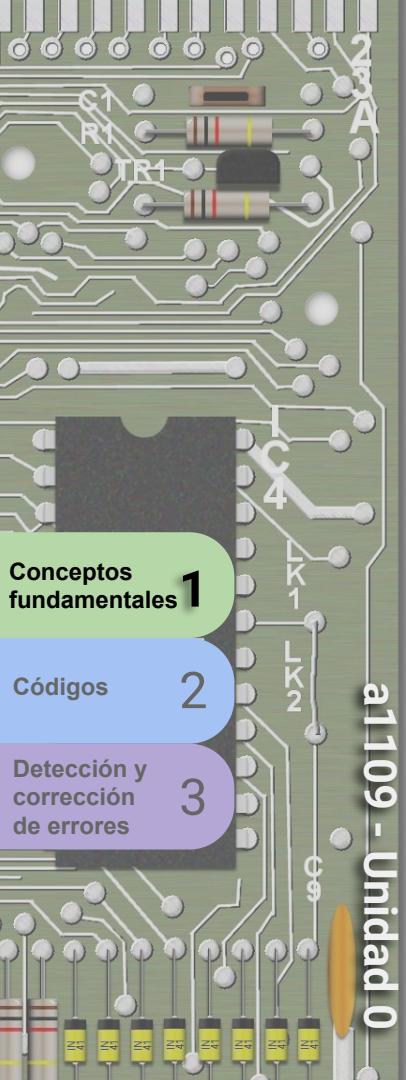
FERRANTI
ULA 2C184E
8214

ZILOG
Z8400A PS
Z80A CPU
8220

SINCLAIR
RESEARCH
D2364C 649

Toshiba
TMM2016P
2-EE4

Códigos: conceptos fundamentales



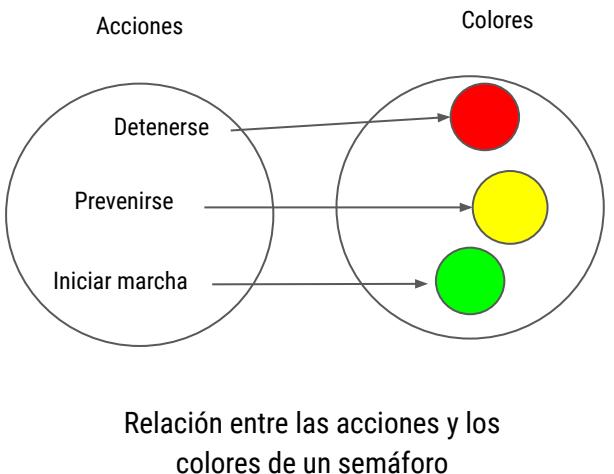
Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

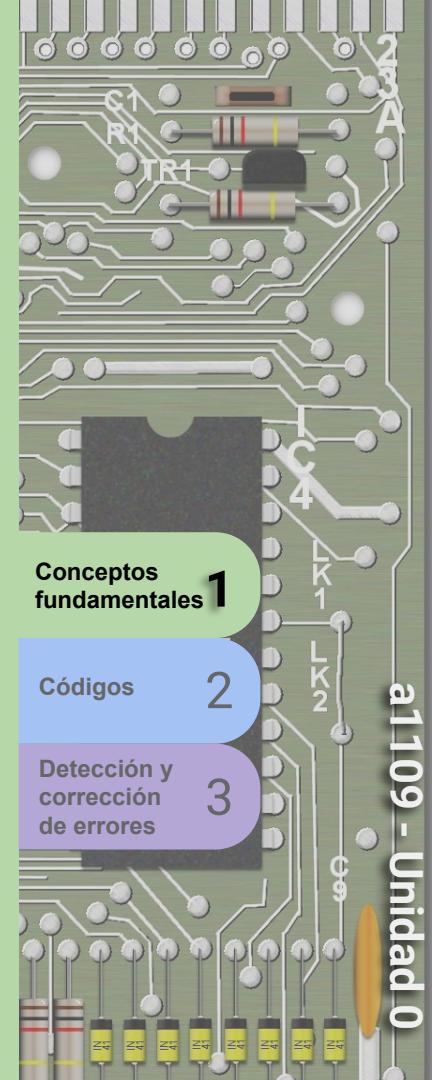
El concepto de código se podría definir como la relación biunívoca entre los elementos de dos conjuntos en donde en uno de esos conjuntos se tienen los elementos que se desean codificar y en el otro conjunto los elementos que se desean utilizar como código.

Que la relación sea biunívoca, significa que para cada elemento de un conjunto se le corresponde uno y solo un elemento del otro conjunto. La vida cotidiana está plagada de este concepto.



John Von Neumann

Relación entre una imagen (QR Code) y el texto “John Von Neumann” (en este caso utilizamos una imagen sola por simplicidad, pero dicha imagen solamente puede representar el texto “John Von Neumann”



Otros ejemplos de código pueden ser la relación biunívoca entre el DNI y las personas, las patentes y los autos, el número de teléfono y la tarjeta SIM, etc.

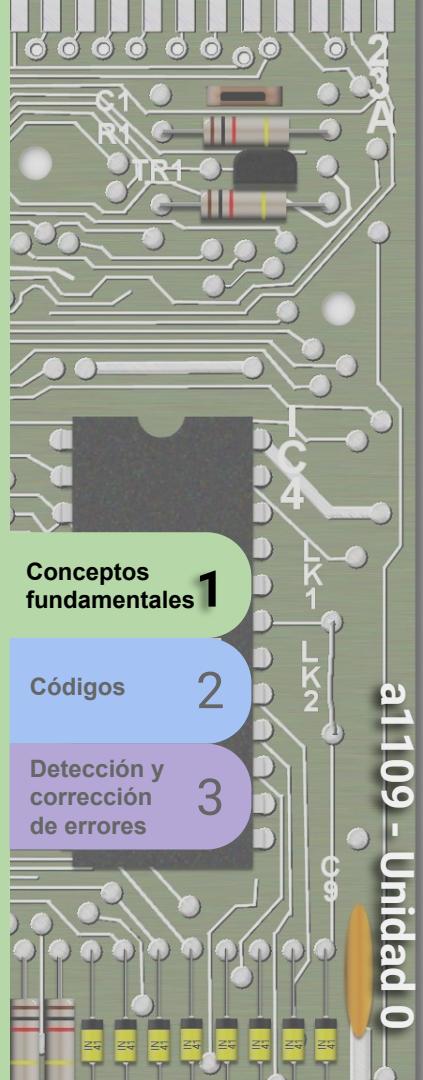
Código binarios

Se definen como códigos binarios a aquellos códigos que están formados solo por unos y ceros. Es decir se denominan códigos binarios a aquellas relaciones entre un conjunto de elementos y combinaciones de unos y ceros de forma biunívoca. Un código binario arbitrario es el que muestra a continuación

Elementos a codificar

ξ	1010
η	0011
γ	1011
θ	0101
ϱ	1111

Elementos codificados con un código binario arbitrario



Módulo de un código

Se define como módulo de un código a la cantidad de elementos que dicho código permite representar. El código binario arbitrario mostrado en la diapositiva anterior posee un módulo de 5

ξ	1010
η	0011
γ	1011
θ	0101

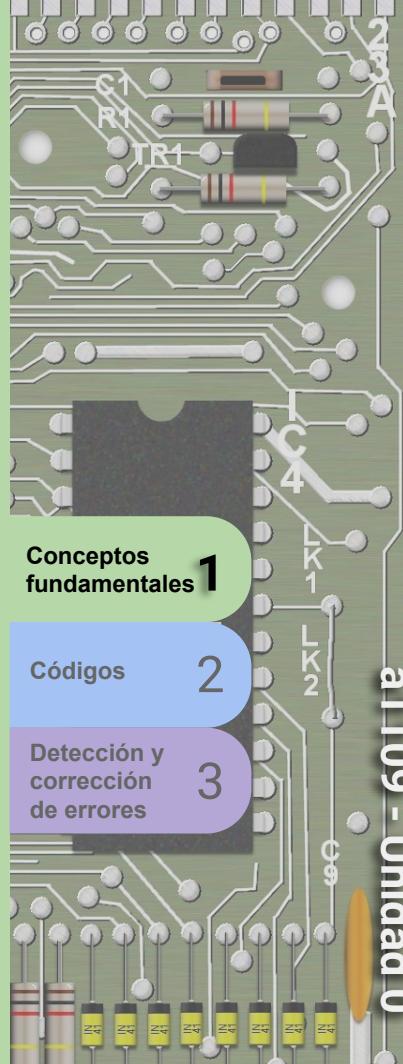
Módulo 4

ξ	1010
η	0011
γ	1011
θ	0101
ϱ	1111

Módulo 5

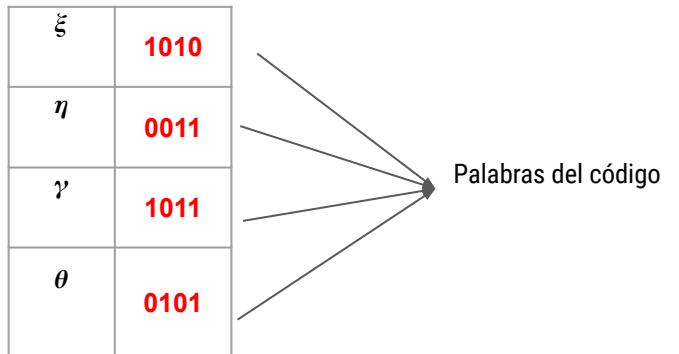
ξ	1010
η	0011
γ	1011
θ	0101
λ	0010
ϱ	1111

Módulo 6



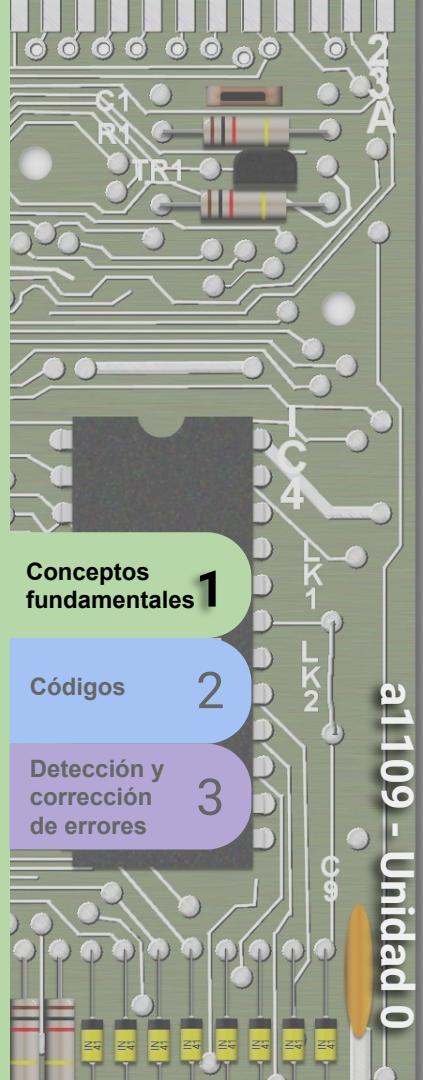
Palabra de un código

Se define como palabra de un código a cualquiera de las combinaciones del código



Código de palabra de longitud fija y código de palabra de longitud variable

Hasta ahora presentamos ejemplos de códigos cuya longitud siempre fue fija, es decir que el código siempre tuvo la misma cantidad de elementos en la palabra, pero esto no es estrictamente necesario para definir un código, sino que también pueden existir o crearse códigos de longitudes variables, es decir códigos donde cada palabra puede tener una cantidad de elementos distinta.

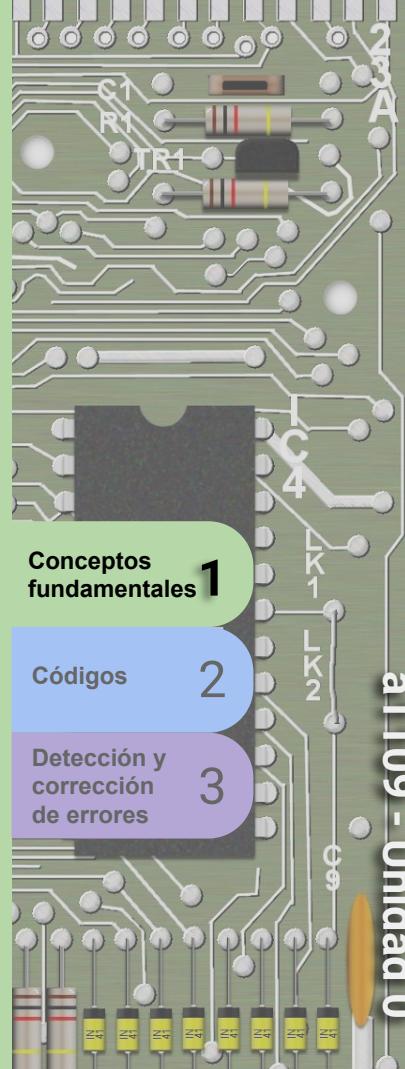


A	-	S	...
B	-...	T	-
C	-.-.	U	..-
D	-..	V	...-
E	.	W	---
F	-.-.	X	----
G	--.	Y	---.
H	Z	---
I	..	1	----
J	---	2	---
K	-.-	3	...
L	-..	4
M	--	5
N	-.	6
O	---	7
P	-..-	8
Q	-.-.	9	---
R	-..	0	----

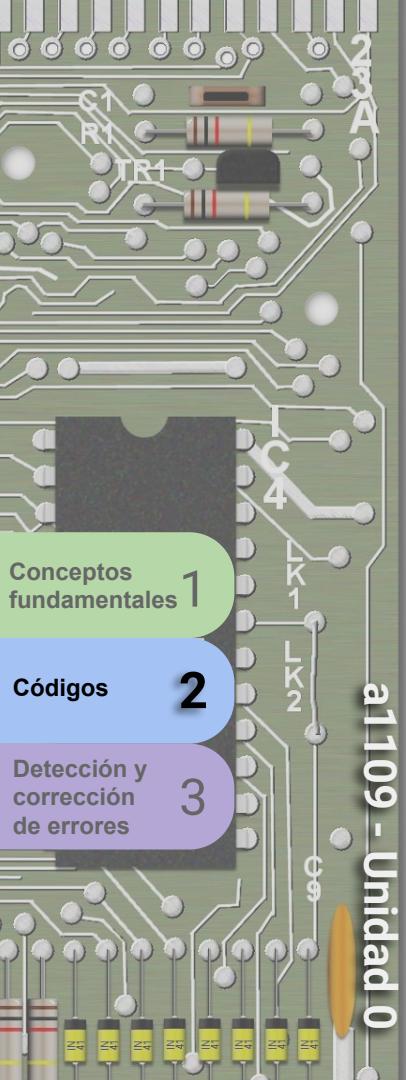
El código Morse, un código de palabra de longitud variable

ξ	1010
η	0011
γ	1011
θ	0101
λ	0010
ϱ	1111

Un código arbitrario de palabra de longitud fija



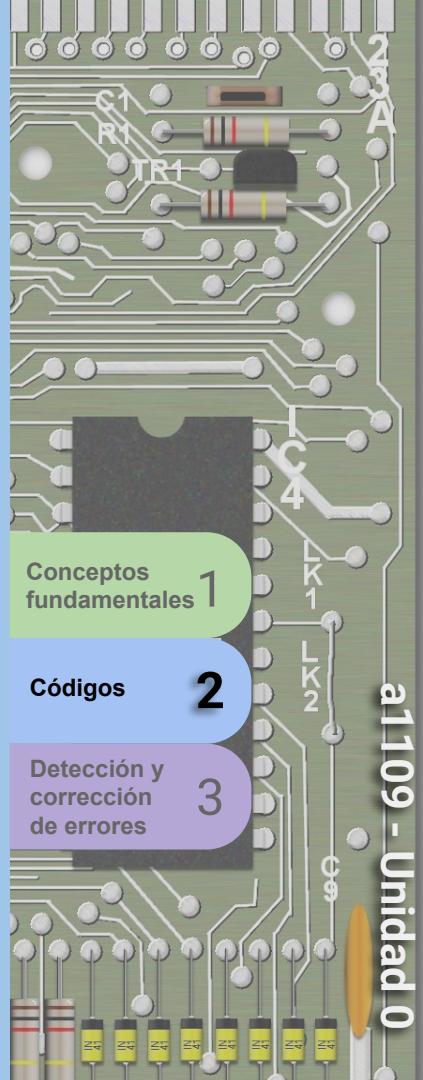
Códigos



Códigos decimales

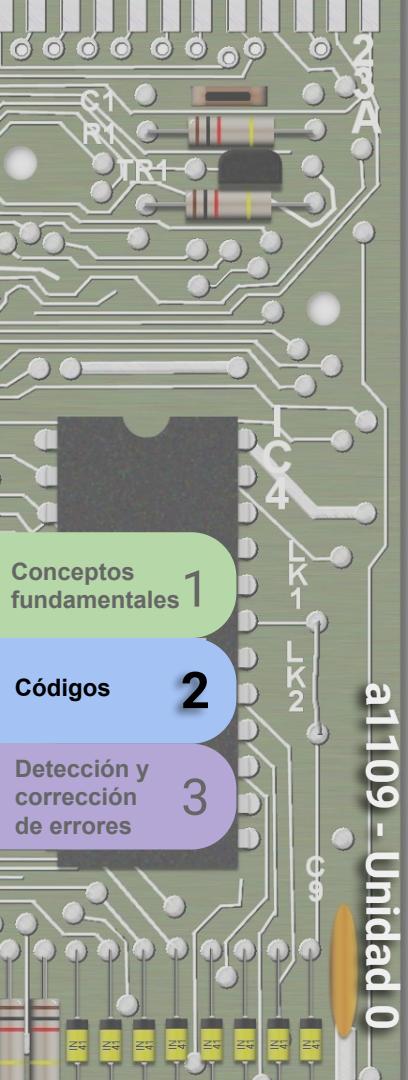
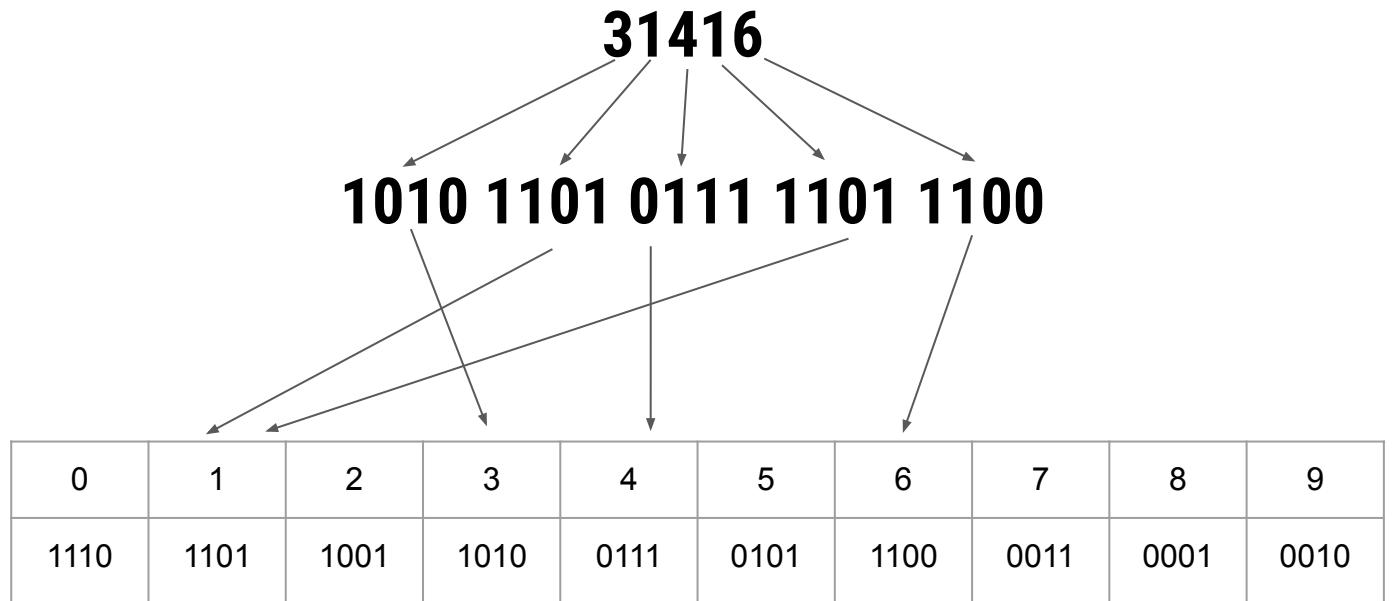
Los códigos numéricos o códigos decimales son códigos utilizados para representar cantidades numéricas y que solo se limitan a representar los diez dígitos decimales. Por lo tanto son todos ellos códigos de módulo 10. A estos códigos se los conoce como códigos BCD (Binary Coded Decimal). Un ejemplo de un código decimal **arbitrario** podría ser el que se muestra a la derecha.

0	1110
1	1101
2	1001
3	1010
4	0111
5	0101
6	1100
7	0011
8	0001
9	0010



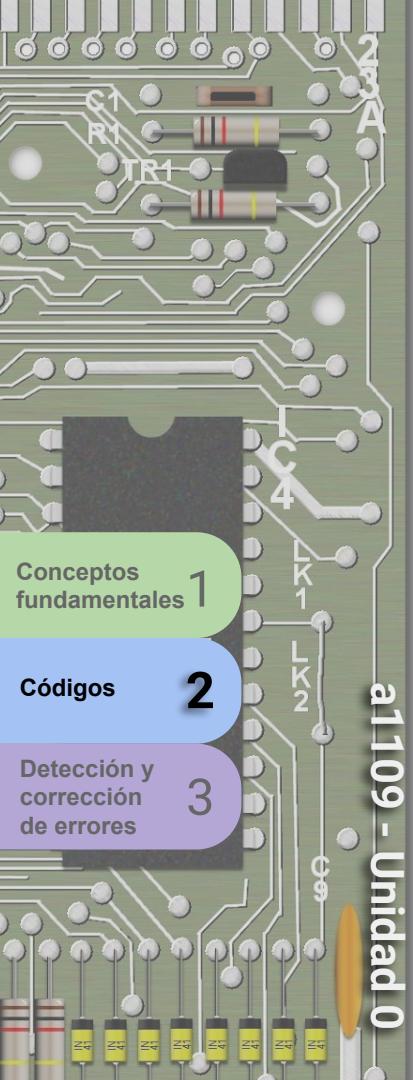
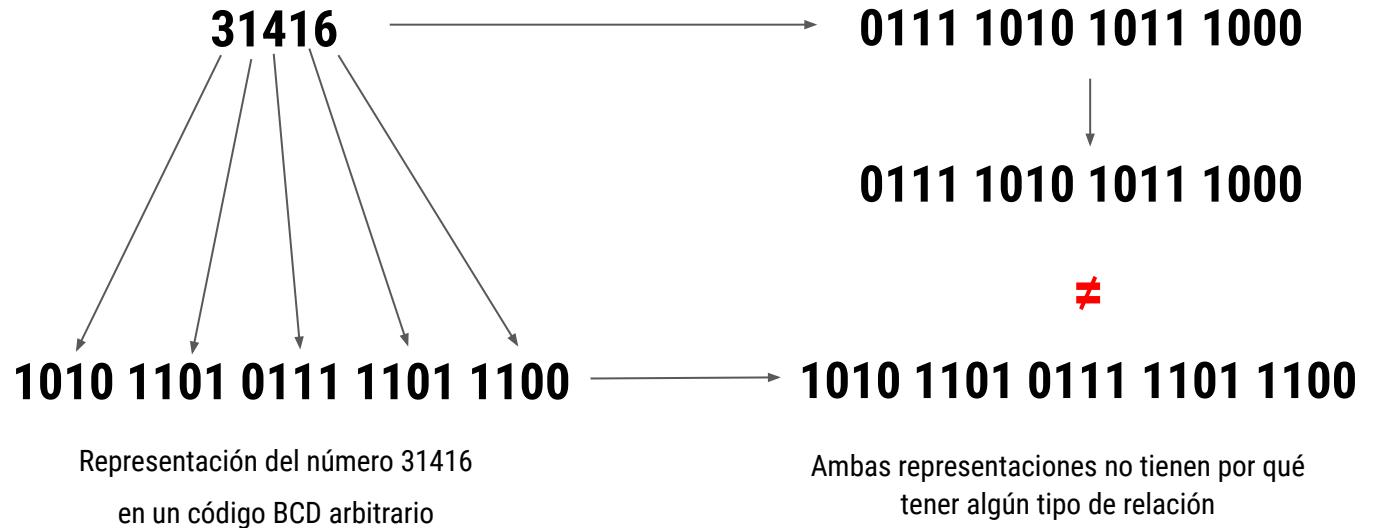
Códigos decimales

Los códigos decimales pueden ser utilizados para representar cifras. Para lograrlo solamente se debe escribir por cada dígito decimal su codificación como se muestra en el siguiente ejemplo



Códigos decimales

Es importante comprender que la codificación en un código BCD no tiene que tener relación alguna con la representación binaria del número codificado



Códigos pesados

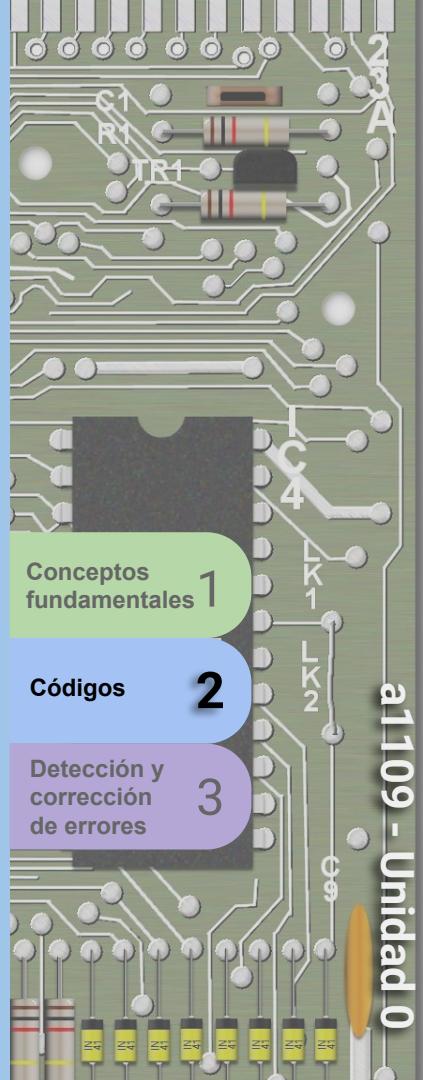
Hasta ahora se ha hablado de códigos arbitrarios sin ningún tipo de limitación, patrón o criterio para definirlos (más allá de la longitud de sus palabras) aunque nada impide que se tomen ciertos criterios para definir un código de acuerdo a las necesidades. Uno de esos criterios que pueden utilizarse, es el de definir **pesos**. Estos pesos darán un valor a cada una de las posiciones que conforman la palabra. Cuando se desee obtener el valor numérico que representa una palabra del código lo único que se deberá hacer es sumar los pesos de las columnas donde exista un uno

DECIMAL	8	4	2	1
	A	B	C	D
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

Pesos (8,4,2 y 1)

$$4+2=6$$

8	4	2	1
0	1	1	0



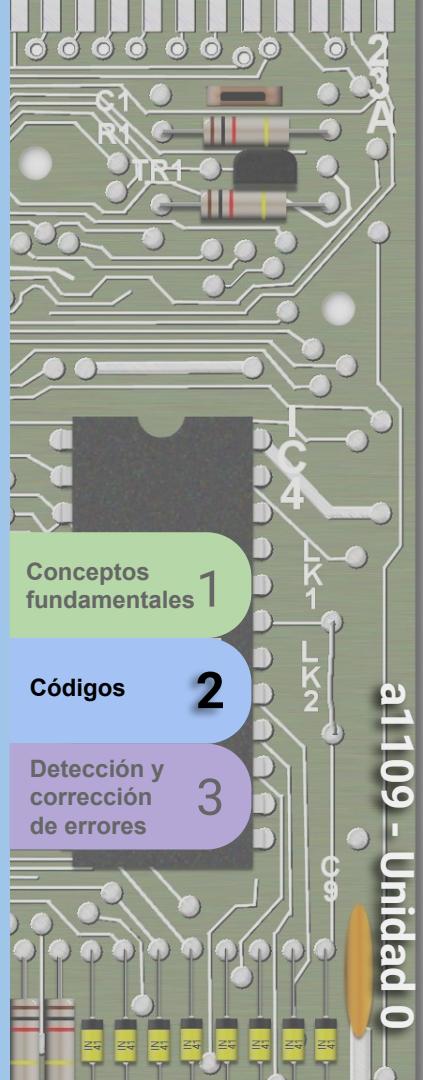
Códigos BCD 8421

Convertamos el número 2917 a BCD 8421

0010 1001 0001 0111

La representación en nibbles es poco práctica para trabajar en una computadora por lo tanto cada uno de los dígitos en BCD debería representarse en un byte.

00000010 00001001 00000001 00000111



Representar 2917 en BCD de a un dígito por byte no es eficiente.

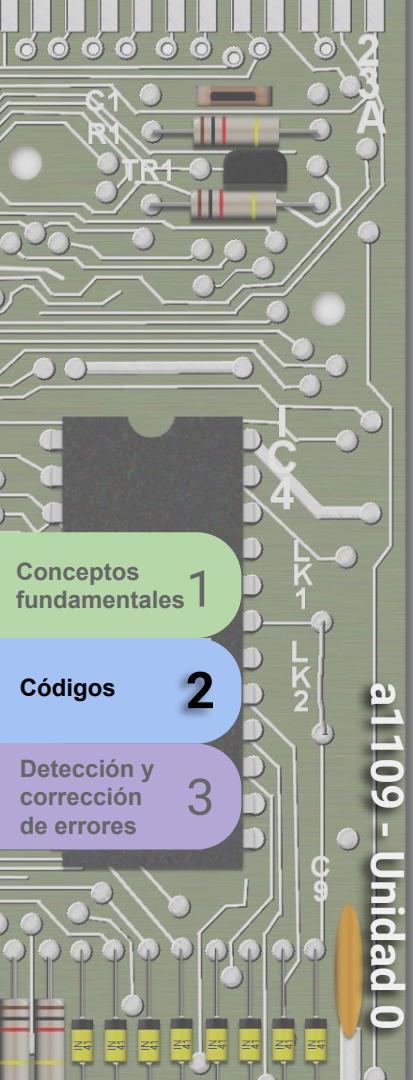
00000010 00001001 00000001 00000111

Una manera práctica de solucionar esta situación es representar dos dígitos en un solo byte

00000010 00001001 00000001 00000111
00101001 00010111
 | | |
 2 9 1 7

Esta forma particular de representar números en código BCD se denomina

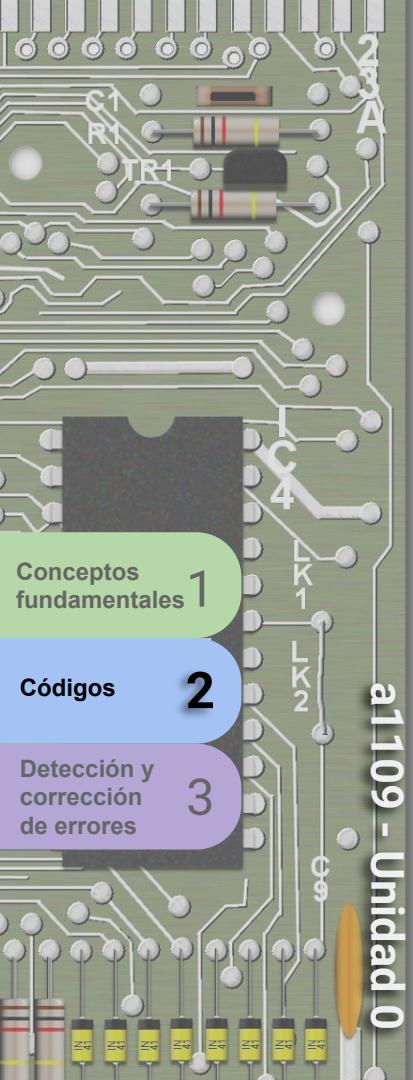
BCD EMPAQUETADO



Códigos sin peso o Códigos no pesados

Ya habiendo explicado los códigos pesados, también es importante mencionar y explicar los códigos no pesados. Estos códigos , como es deducible , no están formados a partir de pesos definidos para cada una de las posiciones de cada uno de los dígitos que conforman las palabras del código. En cambio se utilizan otros criterios o ninguno. Un ejemplo de código no pesado es el que se muestra a continuación.

0	0000
1	1111
2	0001
3	1110
4	0010
5	1101
6	0100
7	1011
8	1000
9	0111



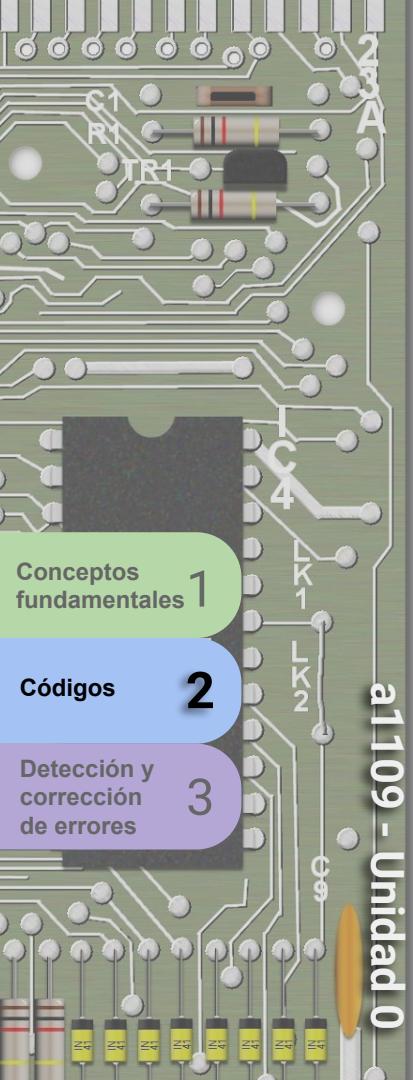
Código progresivo y código cerrado

0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

Se define como código progresivo a aquel código en el cual cada palabra difiere de la siguiente en un solo bit.

Se define como código cerrado a aquel código en el cual su primera y última palabra difieren en un solo bit

El código que se muestra en este diapositiva posee ambas características

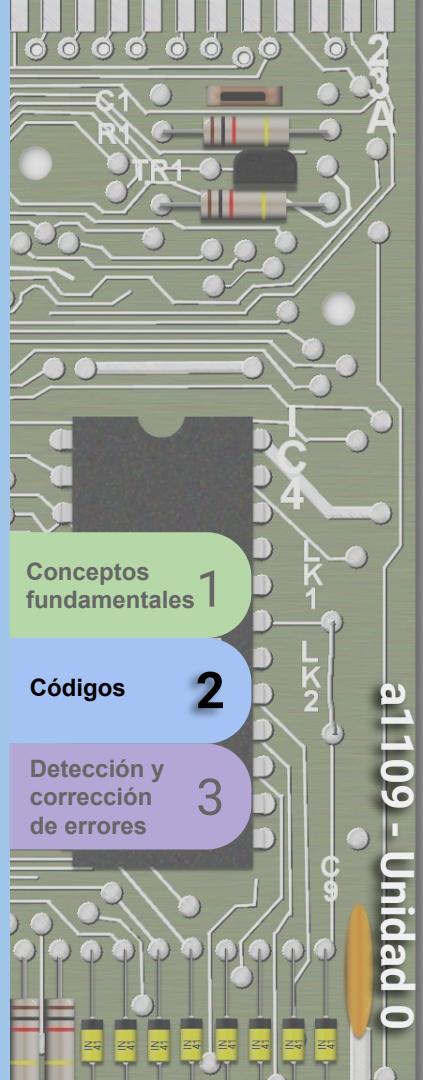


Código Johnson

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Otro ejemplo de código progresivo es el código Johnson , que además de ser progresivo es no pesado y cerrado.

El código Johnson permite representar con n bits 2^n combinaciones. Por lo tanto si deseamos codificar los dígitos del 0-9, necesitaremos 5 bits ($2^5=10$)



Conceptos
fundamentales 1

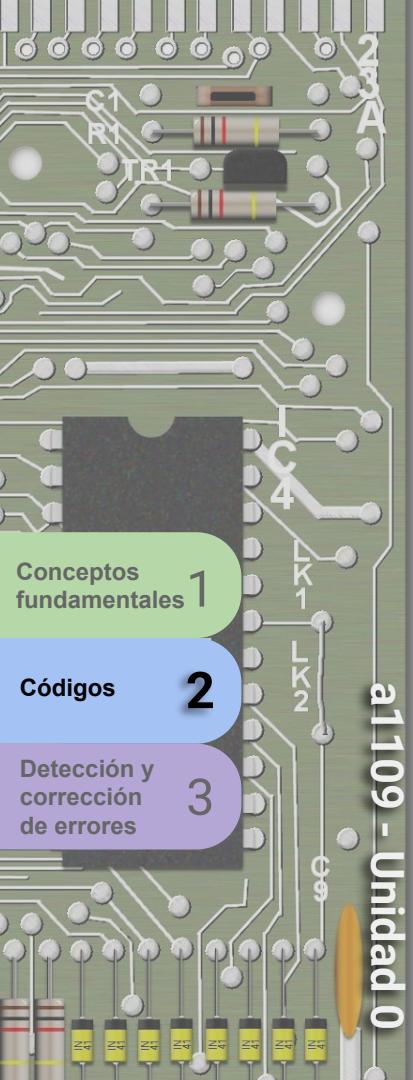
Códigos 2

Detección y
corrección
de errores 3

Código Johnson

0	00000
1	00001
2	
3	
4	
5	
6	
7	
8	
9	

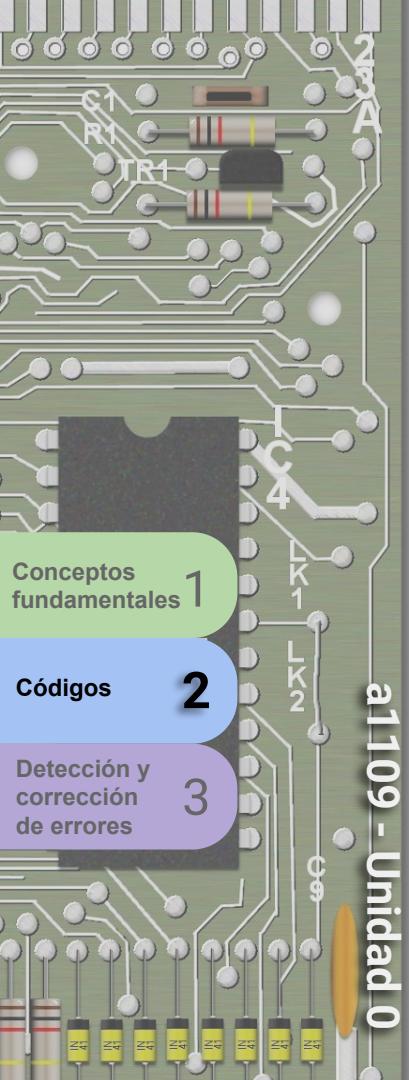
Para crear el código Johnson de 5 bits , comenzemos entonces escribiendo las dos primeras palabras correspondientes al código. Note que no existe hasta ahora ninguna diferencia con el binario puro



Código Johnson

0	00000
1	00001
2	00011
3	00111
4	
5	
6	
7	
8	
9	

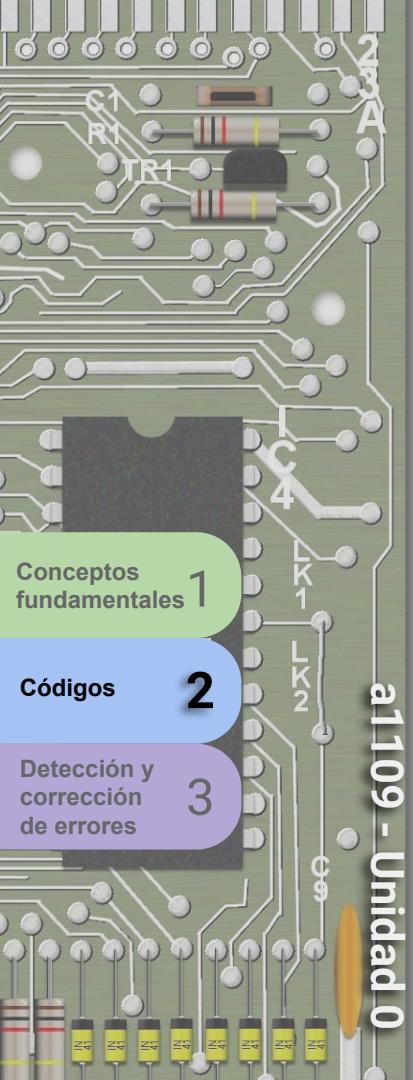
Para crear la siguiente palabra en el código Johnson la regla de generación es *inyectar* un uno desde la derecha a la palabra anterior. Por lo tanto para generar la palabra correspondiente al 2 se le debe *inyectar* un uno desde la derecha a la palabra 00001 quedando 00011 , y con el mismo procedimiento obtenemos la palabra correspondiente al 3 (00111)



Código Johnson

0	00000
1	00001
2	00011
3	00111
4	01111
5	11111
6	
7	
8	
9	

Si continuamos aplicando esta regla, podremos generar las palabras correspondientes al 4 y al 5, pero esta ya no es aplicable cuando se desea generar la palabra correspondiente al 6 debido a que los 5 bits de la palabra anterior ya se encuentran en 1.



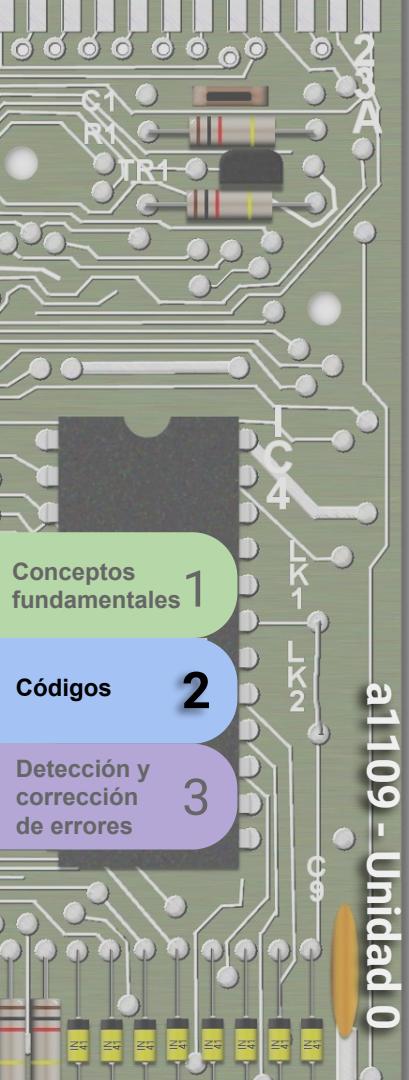
Código Johnson

0	00000
1	00001
2	00011
3	00111
4	01111
5	11111
6	11110
7	11100
8	11000
9	10000

Código BCD utilizando un código Johnson

Por lo tanto al llegar a esta situación, donde todos los bits se encuentran en 1, para crear la palabra siguiente, se tomará la palabra anterior y se le *inyectará* un 0 desde la derecha, descartando el bit más significativo. En consecuencia para crear la palabra correspondiente al 6 , se tomará la palabra correspondiente al 5 , 11111, y se le *inyectará* un cero desde la derecha (descartando el bit más significativo) quedando la nueva palabra 11110

Aplicando esta regla ,el código Johnson de 5 bits, resulta como se muestra en la tabla.

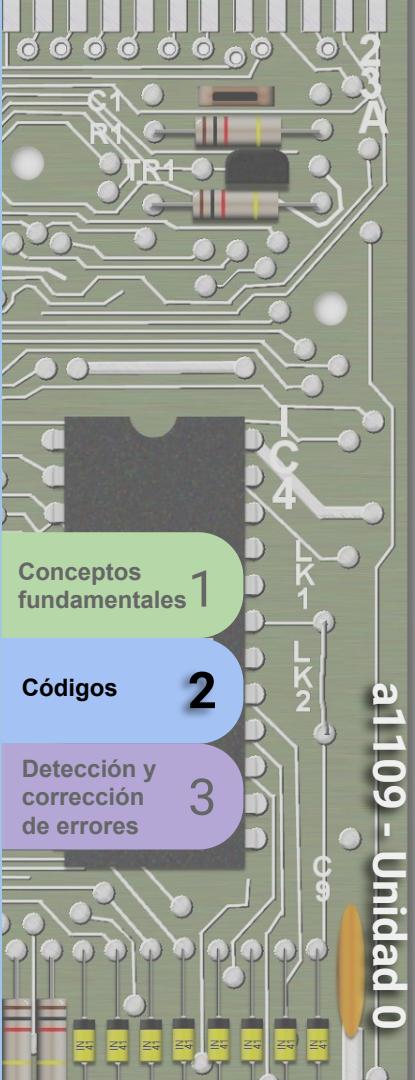


Código Gray

Otro ejemplo de código progresivo es el código Gray. Este código además de ser progresivo es no pesado, cerrado y posee una característica propia: es un código reflejado. El módulo del código Gray es 2^n , siendo n la cantidad de bits que conformaran las palabras del código . Por otra parte, un código Gray de n bits puede ser generado a partir de un código Gray de n-1 bits.

El código Gray más sencillo que se puede formar es aquel en el cual n es 1 , por lo tanto se conseguirá un código de módulo 2 ($2^1=2$) entonces solamente se podrán codificar dos elementos.

Elemento	Código Gray
0	0
1	1



Código Gray

El ejemplo de código Gray anterior es trivial, entonces ¿Cómo generamos un código Gray con $n=2$? Tomando el código Gray con $n=1$, copiamos el mismo y agregaremos las mismas combinaciones del código Gray, debajo de la tabla, pero reflejadas, entendiéndose *reflejar* a escribir las combinaciones rebatidas como si se vieran en un espejo respecto al eje de simetría (línea roja). Finalmente para obtener el código Gray con $n=2$ agregaremos de un lado del eje de simetría un bit más a cada una de las palabras , en 0, y del otro lado del eje de simetría se realizará el mismo procedimiento pero con el nuevo bit en 1.

Código de Gray $n=1$

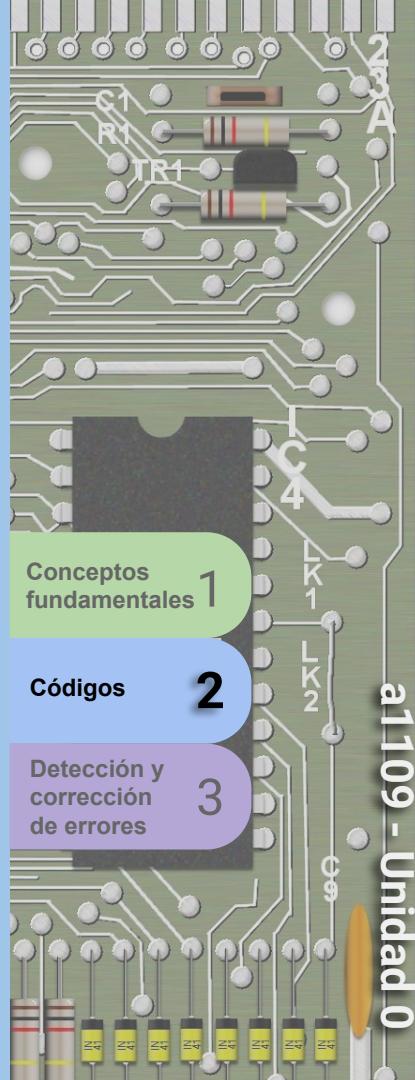
0	0
1	1

Reflejamos

0	0
1	1
2	1
3	0

Código de Gray $n=2$

0	00
1	01
2	11
3	10



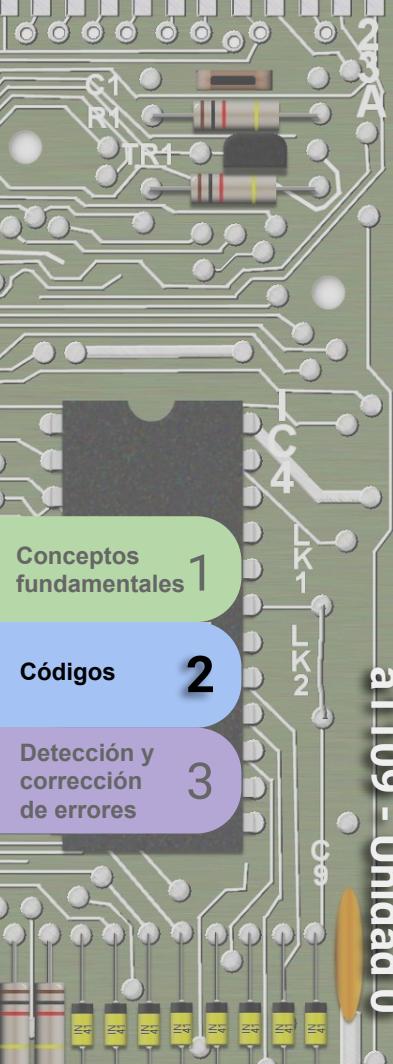
Código Gray

De la misma forma podemos obtener un código Gray con n=3 y módulo 8

0	00
1	01
2	11
3	10

0	00
1	01
2	11
3	10
4	10
5	11
6	01
7	00

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



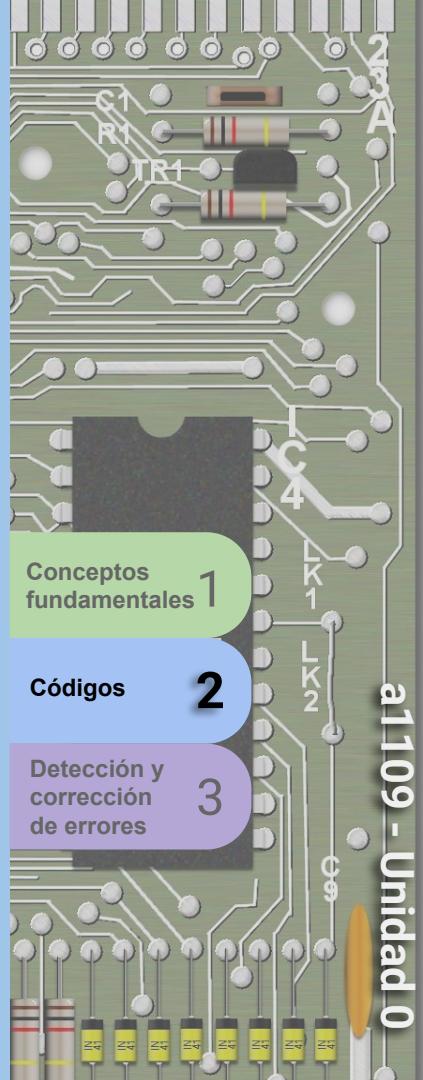
Código Gray

Con el mismo procedimiento podemos obtener un código Gray con $n = 4$ y módulo 16

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

0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100
8	100
9	101
10	111
11	110
12	010
13	011
14	001
15	000

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



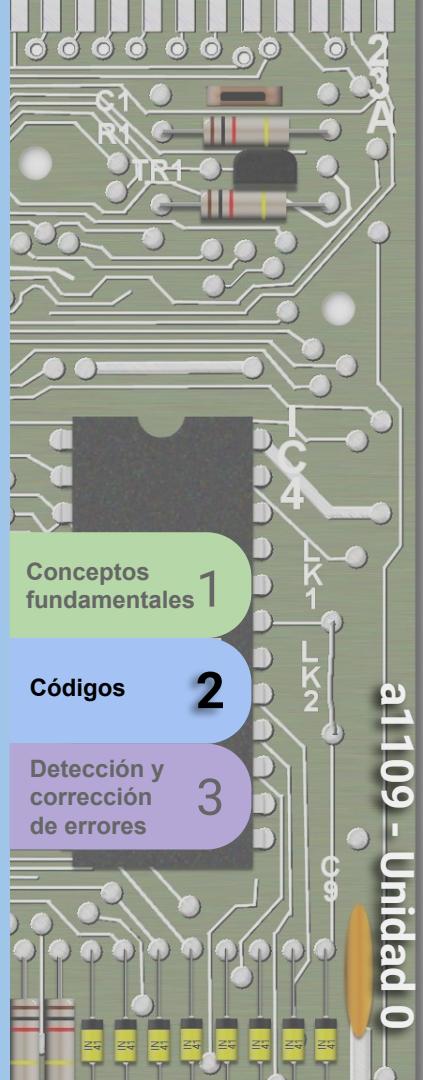
Código Gray

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

Que el código Gray sea reflejado significa que para cada palabra del código existirá otra palabra simétrica respecto al eje de simetría que solamente variará en un bit.

Note que para la palabra correspondiente al número 7 (0100) la palabra simétrica es la correspondiente al número 8 (1100). Ambas palabras difieren en un bit.

Lo mismo sucede con las palabras correspondientes al 5 (0111) y 10 (1111), solo difieren en un bit.



Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

Código Gray

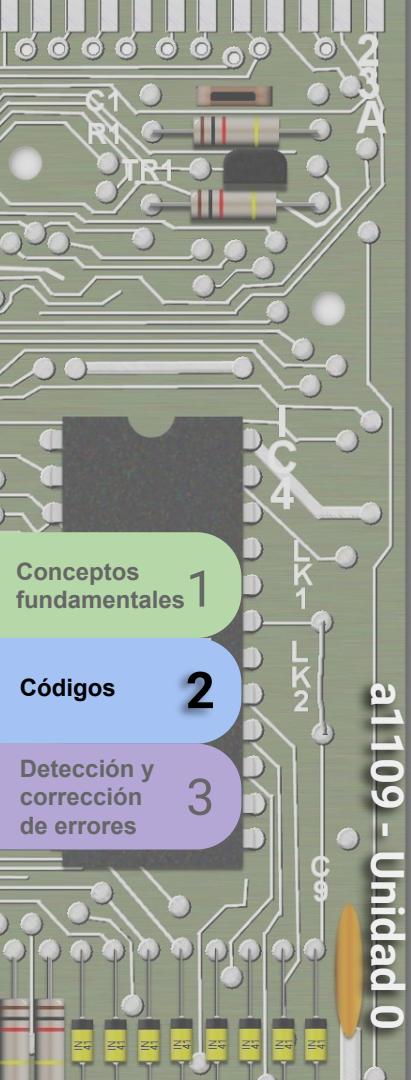
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

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

Si deseamos crear un código BCD a partir del código Gray con módulo 16, bastará con eliminar 6 combinaciones. Así obtendríamos un código BCD que cumpliría con todas las características de los códigos ya vistos. Pero si son eliminadas sin ningún tipo de criterio perderíamos las propiedades que posee el código Gray.

Para crear códigos que posean la estructura del código de Gray y tengan módulos menores a 2^n se deben eliminar combinaciones y sus simétricas, desde el eje de simetría o desde los extremos. Por ejemplo, para obtener un código con módulo 14 se podrán eliminar las combinaciones que se muestran en las tablas.

Si se eliminan combinaciones simétricas pero no se lo hace desde el eje de simetría o desde los extremos se obtendrá un código reflejado, pero no progresivo



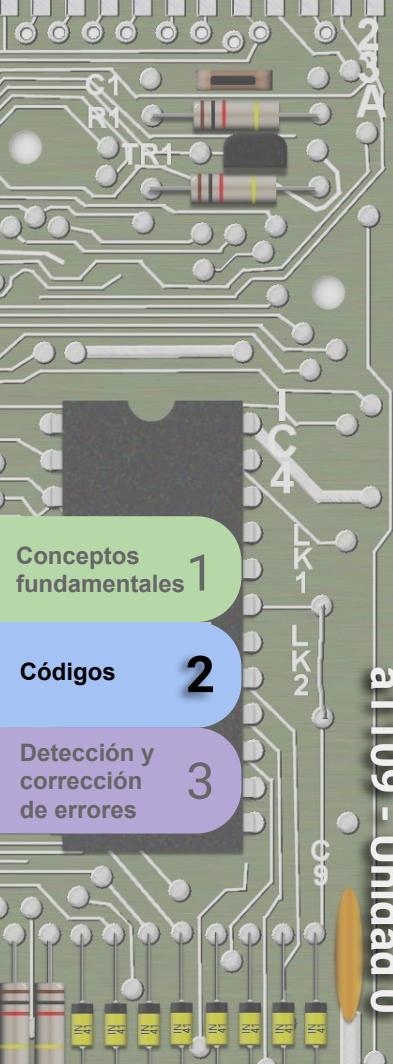
Código Gray

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

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

Finalmente si deseamos obtener un código BCD a partir de un código Gray con módulo 16 , manteniendo las propiedades del mismo, debemos eliminar las 6 combinaciones centrales o las tres primeras y las tres últimas.

Código BCD utilizando un código Gray

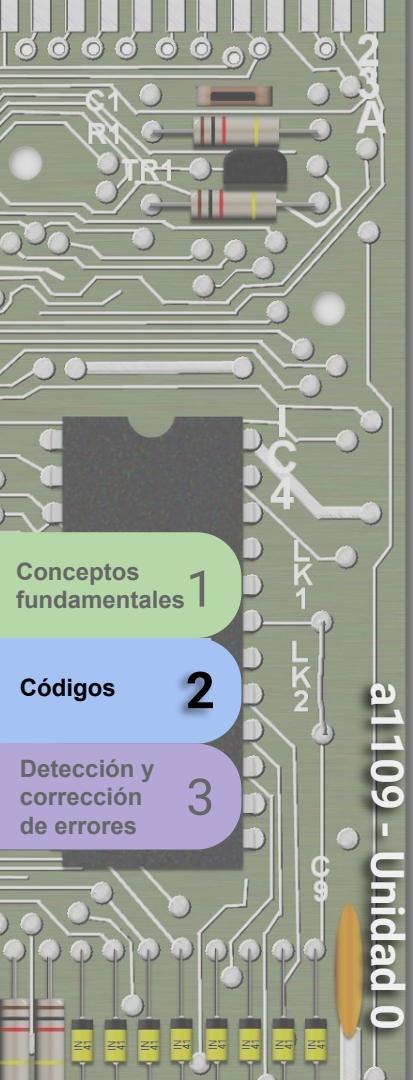


Códigos autocomplementados- Código Aiken

DECIMAL	2	4	2	1
	A	B	C	D
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	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Código BCD Aiken

El código que se muestra en la diapositiva se denomina código Aiken. El mismo es un código pesado cuyos pesos son 2, 4, 2 y 1 y que posee una característica particular.



Códigos autocomplementados- Código Aiken

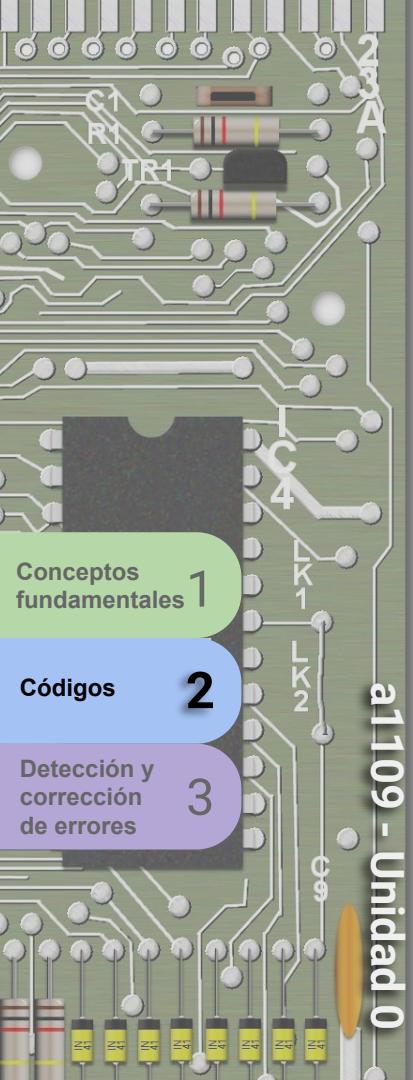
DECIMAL	2	4	2	1
	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Código BCD Aiken

$$\begin{aligned}C_{B-1}(0) &= 9 \\&\text{ &} \\C_{B-1}(0000) &= 1111\end{aligned}$$

El código que se muestra en la diapositiva se denomina código Aiken. El mismo es un código pesado cuyos pesos son 2, 4, 2 y 1 y que posee una característica particular.

Al tomar la palabra correspondiente al número 0, si se complementa a la base menos uno, se obtendrá 1111 , cuyo valor está asociado al número 9 en el código Aiken. Note además que si se complementa a la base menos uno a 0 , se obtiene 9.



Códigos autocomplementados- Código Aiken

DECIMAL	2	4	2	1
	A	B	C	D
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	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

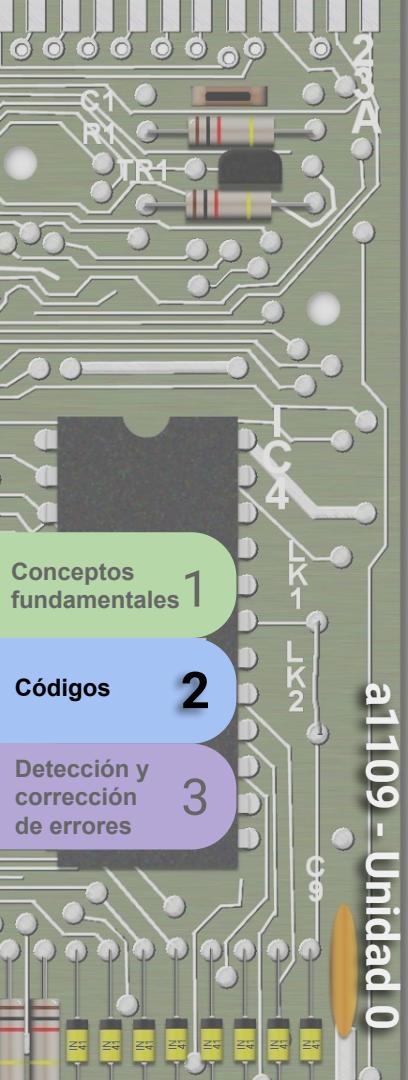
Código BCD Aiken

$$\begin{aligned}C_{B-1}(1) &= 8 \\&\text{ &} \\C_{B-1}(0001) &= 1110\end{aligned}$$

El código que se muestra en la diapositiva se denomina código Aiken. El mismo es un código pesado cuyos pesos son 2, 4, 2 y 1 y que posee una característica particular.

Al tomar la palabra correspondiente al número 0, si se complementa a la base menos uno, se obtendrá 1111 , cuyo valor está asociado al número 9 en el código Aiken. Note además que si se complementa a la base menos uno a 0 , se obtiene 9.

Si se realiza el mismo procedimiento con la palabra asociada al número 1, se obtendrá 1110 que corresponde a la palabra asociada al número 8 en el código Aiken. Nuevamente note que si se complementa a la base menos uno a 1 se obtiene 8. Si realizamos el mismo procedimiento para todas las palabras del código Aiken observaremos la misma propiedad.



Códigos autocomplementados- Código Aiken

DECIMAL	2	4	2	1
	A	B	C	D
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	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

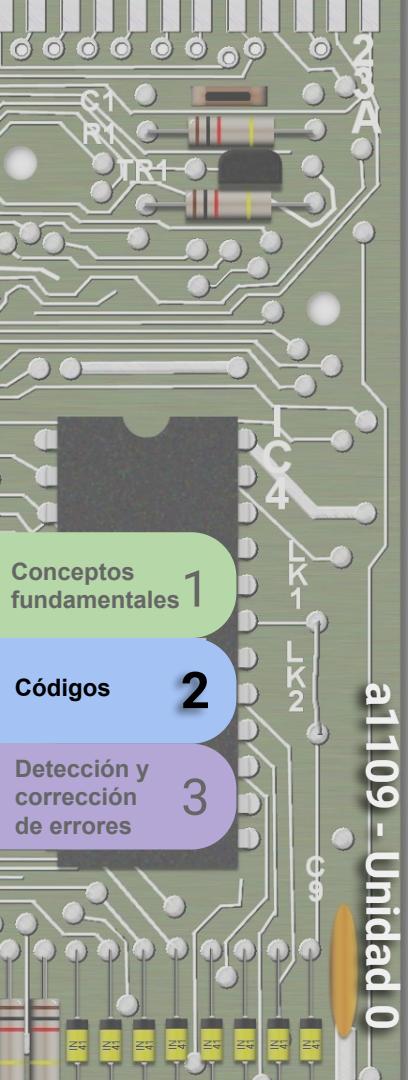
Código BCD Aiken

El código que se muestra en la diapositiva se denomina código Aiken. El mismo es un código pesado cuyos pesos son 2, 4, 2 y 1 y que posee una característica particular.

Al tomar la palabra correspondiente al número 0, si se complementa a la base menos uno, se obtendrá 1111 , cuyo valor está asociado al número 9 en el código Aiken. Note además que si se complementa a la base menos uno a 0 , se obtiene 9.

Si se realiza el mismo procedimiento con la palabra asociada al número 1, se obtendrá 1110 que corresponde a la palabra asociada al número 8 en el código Aiken. Nuevamente note que si se complementa a la base menos uno a 1 se obtiene 8. Si realizamos el mismo procedimiento para todas las palabras del código Aiken observaremos la misma propiedad.

Cuando un código posee esta propiedad en todas sus palabras , se puede decir que el código es **autocomplementado**.

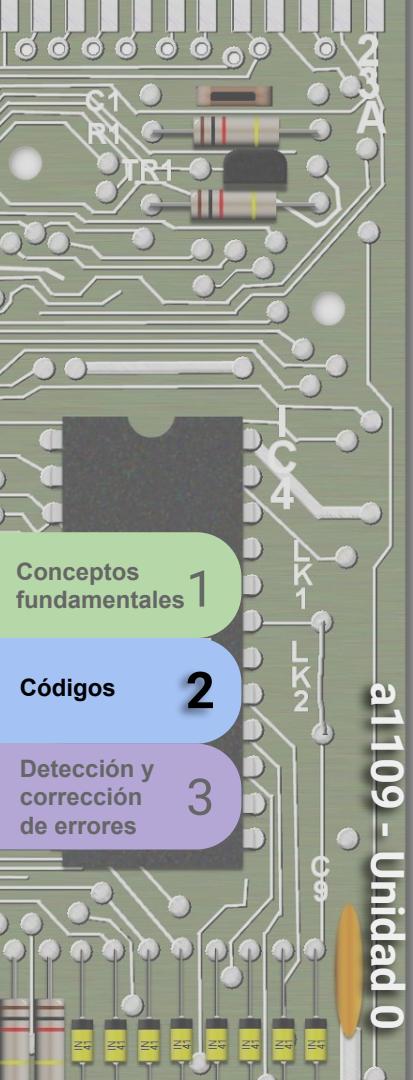


Códigos autocomplementados- Código BCD Exceso 3 (XS 3)

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1						
1	0	1	0	0						
2	0	1	0	1						
3	0	1	1	0						
4	0	1	1	1						
5	1	0	0	0						
6	1	0	0	1						
7	1	0	1	0						
8	1	0	1	1						
9	1	1	0	0						

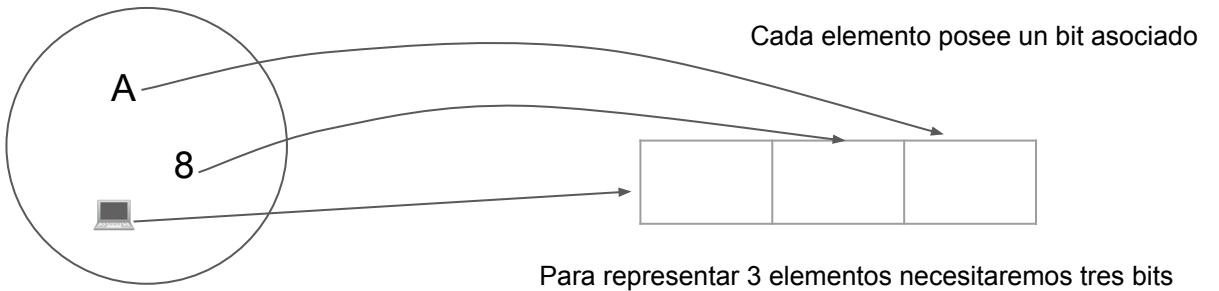
Código BCD XS 3

Otro código que posee la característica de autocomplementado es el código BCD exceso 3 (XS 3). Este código BCD no pesado contiene las diez primeras combinaciones del binario puro a partir de 3, por lo tanto el 0 en exceso 3 se representa como 0011, el 1 como 0100, el 2 como 0101, etc.

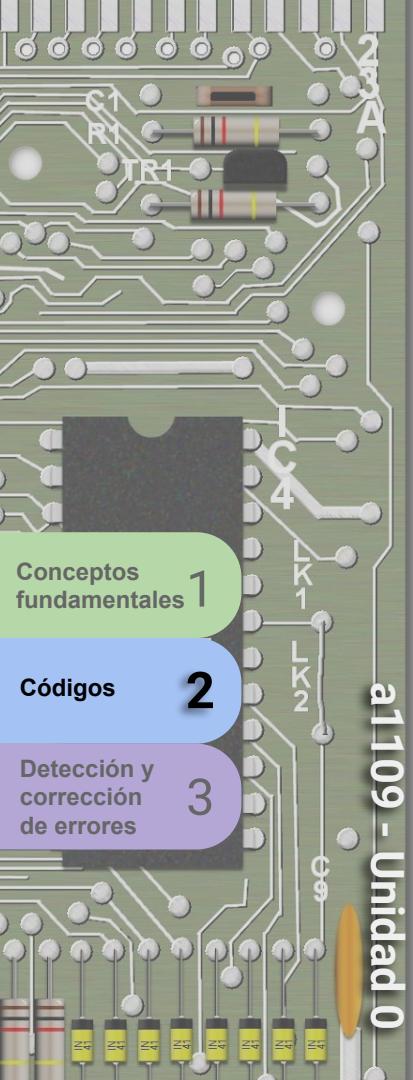


One hot & One cold

El código one hot , es un sencillo código , sin peso y con longitud de palabra fija. En el código one hot , cada palabra poseerá un bit por cada uno de los elementos que se desea representar y cada uno de estos bits estará asociado a un elemento a representar.



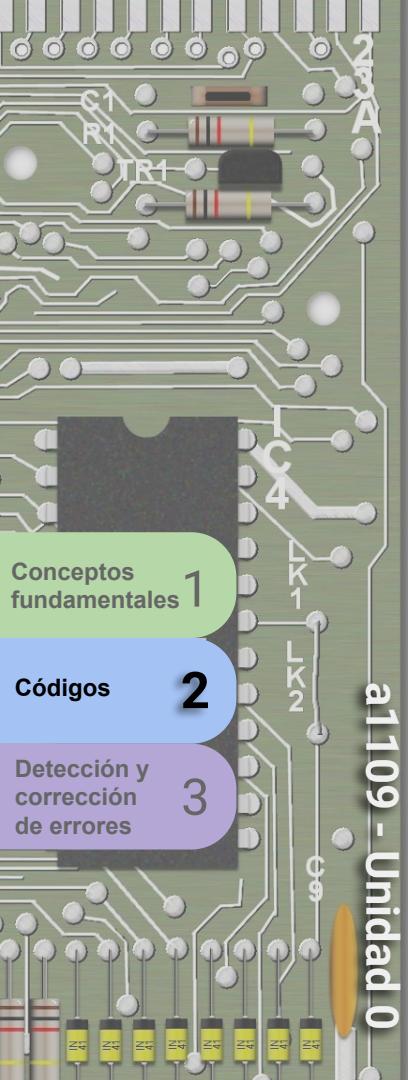
Cada una de las palabras de este código contendrá todos los bits en cero y solamente un bit en uno , el bit que está asociado al elemento que se desea representar con esa palabra. Como es evidente no se permite más de un uno en cada una de las palabras. Este código se puede implementar de manera inversa , cambiando los unos por ceros y los ceros por uno, en este caso al código se lo denomina one-cold



One hot & One cold

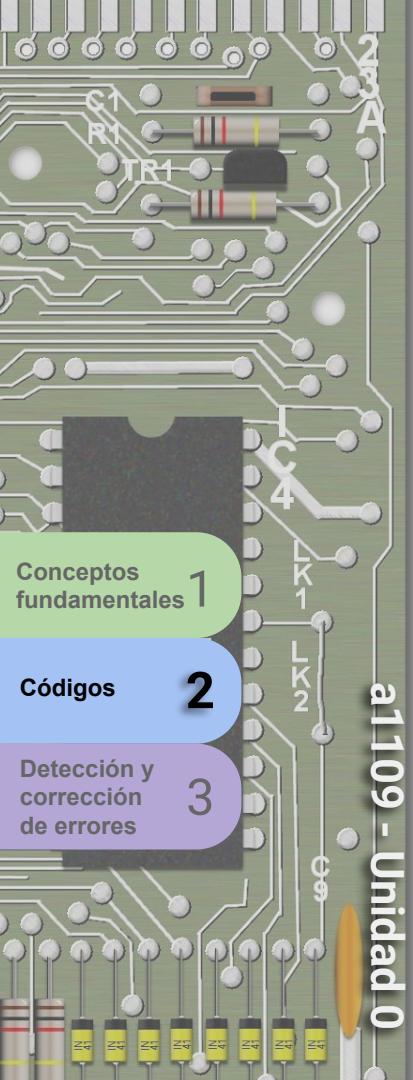
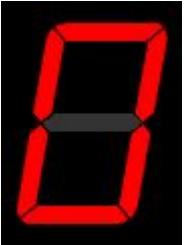
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	1	0

Código BCD utilizando One-hot



¿Por qué no usar solamente el sistema binario de numeración para representar cantidades?

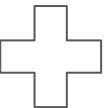
Existen situaciones en las cuales es más útil utilizar otros códigos de representación que no sea el binario puro. Uno de esos motivos es por simplicidad. Imaginen una calculadora. Los dígitos en una calculadora son cargados uno a uno, y si se desea trabajar con binario puro, por cada ingreso de un dígito se debería reconvertir el número de BCD 8421 , por ejemplo, a binario puro, creando trabajo innecesario.



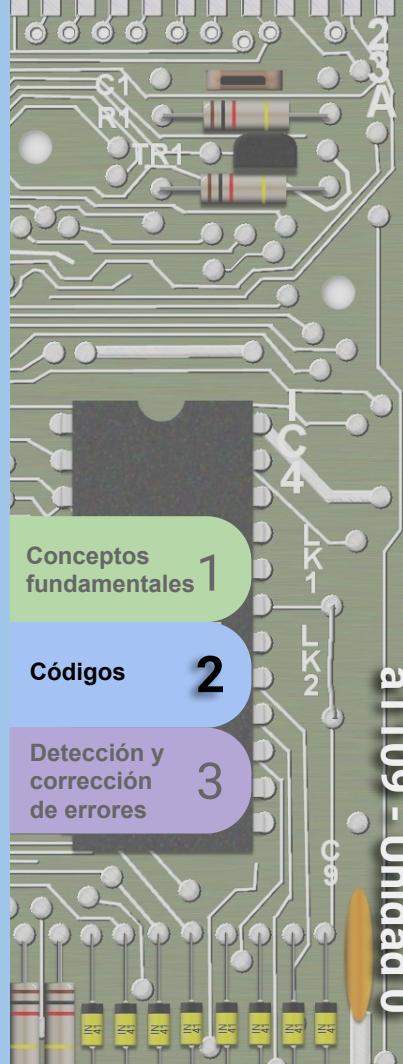
Suma en BCD 8421

Realice la siguiente suma en BCD 8421 1230+2534

1230 = 0001 0010 0011 0000



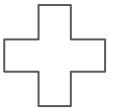
2534 = 0010 0101 0011 0100



Suma en BCD 8421

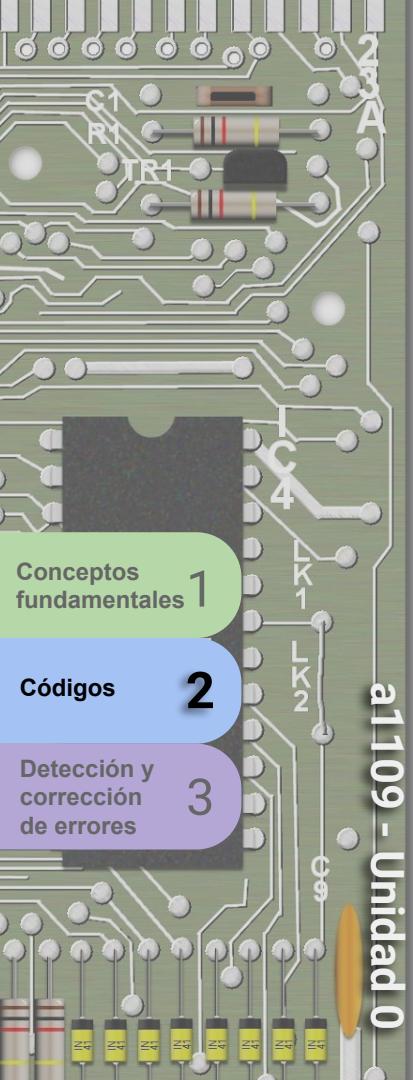
Realice la siguiente suma en BCD 8421 1230+2534

1230 = 0001 0010 0011 0000



2534 = 0010 0101 0011 0100

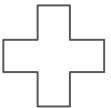
3764 = 0011 0111 0110 0100



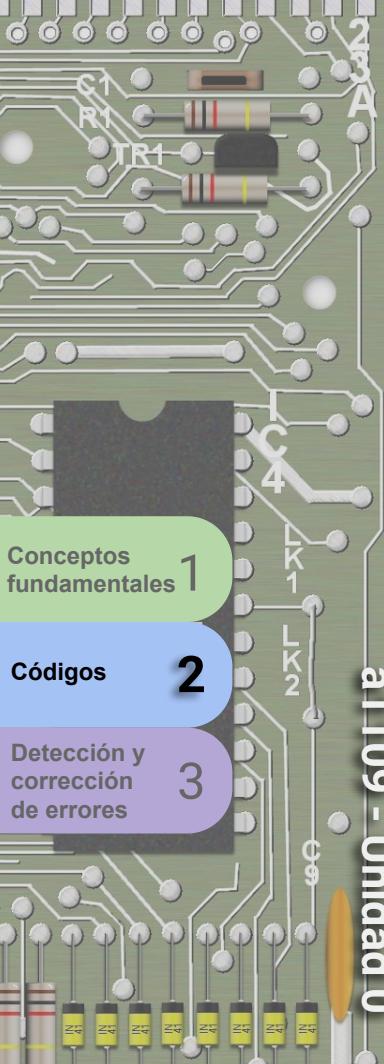
Suma en BCD 8421

Realice la siguiente suma en BCD 8421 $2961+9989$

2961 = 0010 1001 0110 0001



9989 = 1001 1001 1000 1001



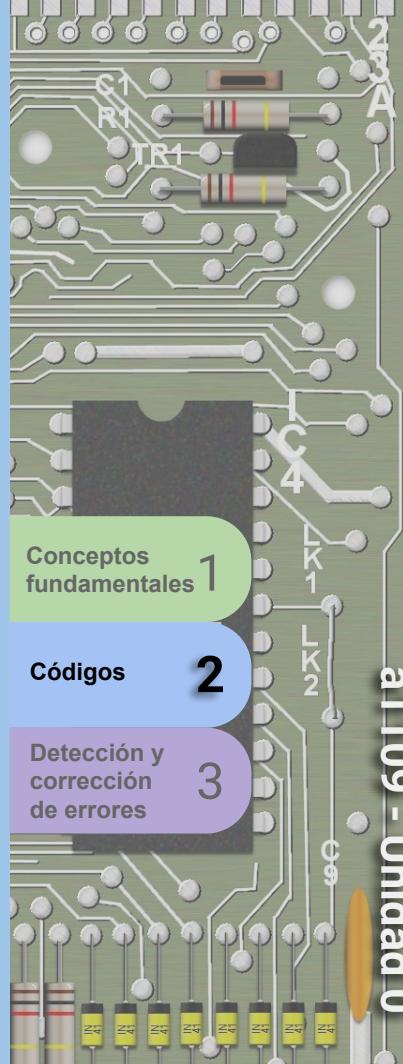
Suma en BCD 8421

Realice la siguiente suma en BCD 8421 $2961+9989$

$2961 = \begin{array}{r} 1 \\ 0010 \\ 1001 \\ 0110 \\ 0001 \end{array}$

$9989 = \begin{array}{r} 1001 \\ 1001 \\ 1000 \\ 1001 \end{array}$

$$\begin{array}{r} 1100 \\ 0010 \\ 1110 \\ 1010 \\ \hline 12 \quad 18 \quad 14 \quad 10 \end{array}$$

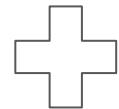


Suma en BCD 8421

Realice la siguiente suma en BCD 8421 $2961+9989$

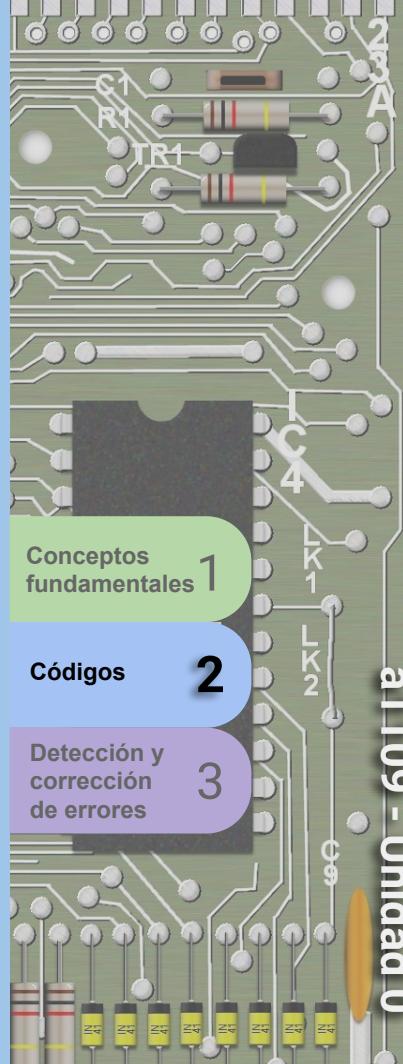
2961 = $\begin{array}{r} 1 \\ 0010 \end{array}$ 1001 0110 0001

9989 = 1001 1001 1000 1001



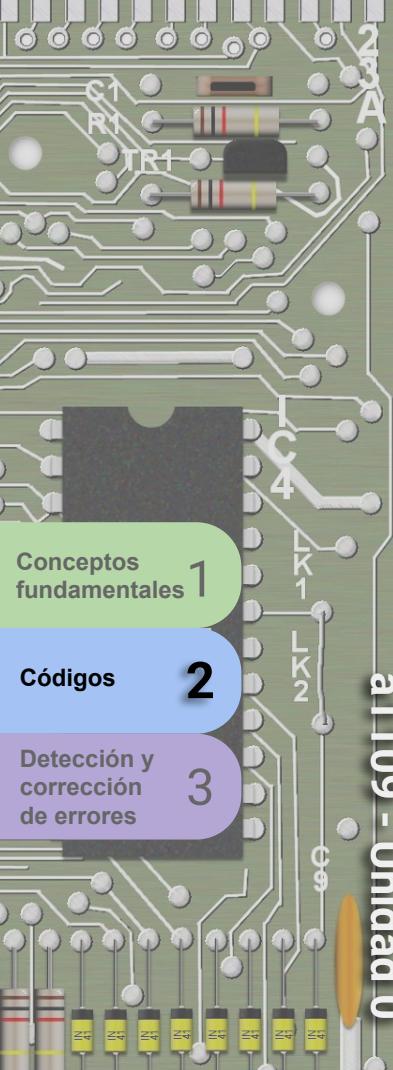
Fuera de código **1100** **0010** **1110** **1010**

12 **18** **14** **10**



Códigos BCD 8421

A	B	C	D	DECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15



Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

Suma en BCD 8421

Realice la siguiente suma en BCD 8421 $2961+9989$

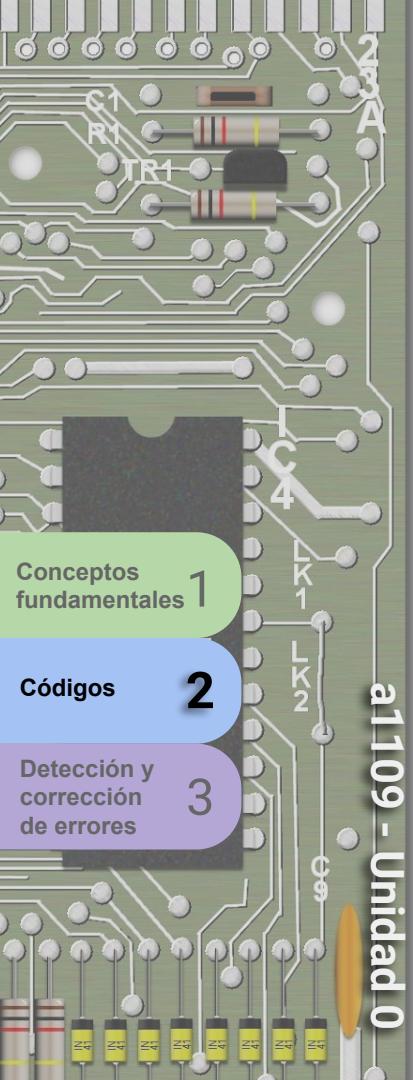
2961 = $\begin{array}{c} 1 \\ 1 \\ 1 \end{array}$ 0010 1001 0110 0001

9989 = 1001 1001 1000 1001

1 **1100** **0010** **1110** **1010**

0000 110 110 110 110

12950 = 0001 0010 1001 0101 0000



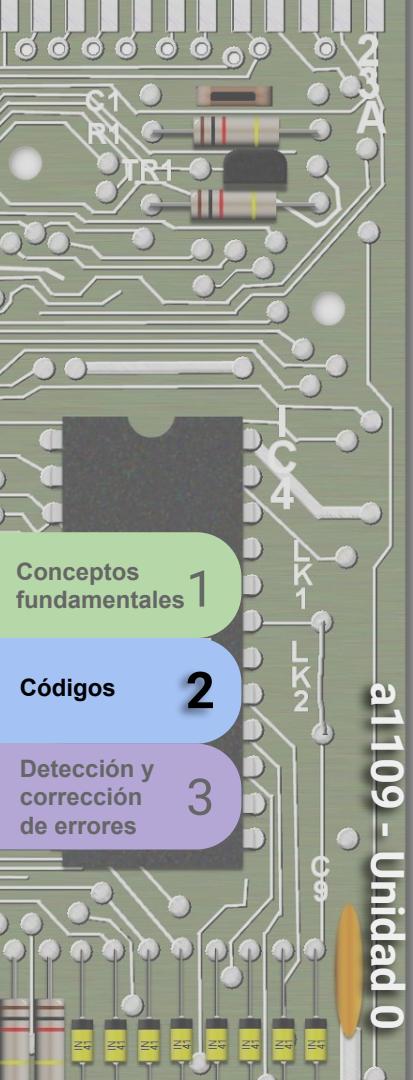
Suma en BCD XS 3

La suma en BCD XS 3 es también posible como lo es en BCD 8421, aunque al igual que en este último caso se necesitan realizar correcciones para llegar al resultado correcto. Comencemos sumando 0+0

DECIMAL	0	0	0	1	1
0	0	0	1	1	
1	0	1	0	0	
2	0	1	0	1	
3	0	1	1	0	
4	0	1	1	1	
5	1	0	0	0	
6	1	0	0	1	
7	1	0	1	0	
8	1	0	1	1	
9	1	1	0	0	

$$\begin{array}{r} 0011 \\ + 0011 \\ \hline \end{array} \quad \begin{array}{l} \rightarrow 0 \text{ (En XS 3)} \\ \text{+} \\ \rightarrow 0 \text{ (En XS 3)} \end{array}$$

Código BCD XS 3



Suma en BCD XS 3

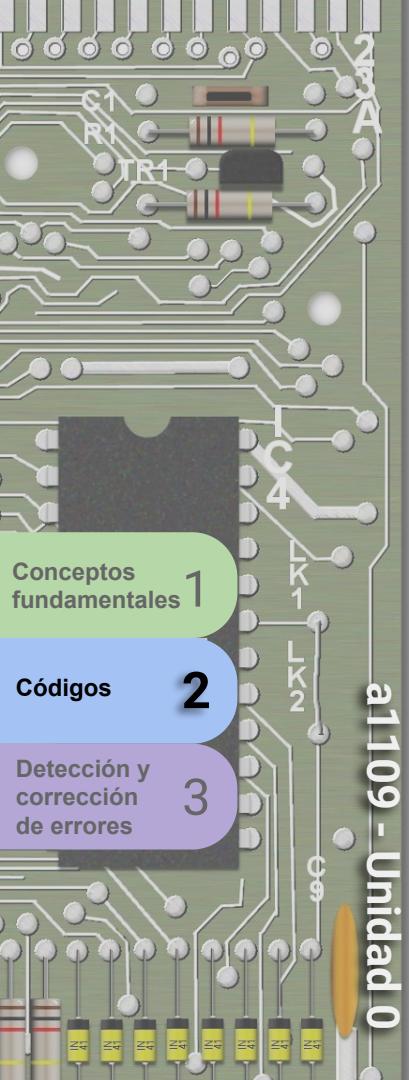
La suma en BCD XS 3 es también posible como lo es en BCD 8421, aunque al igual que en este último caso se necesitan realizar correcciones para llegar al resultado correcto. Comencemos sumando 0+0

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1						
1	0	1	0	0						
2	0	1	0	1						
3	0	1	1	0						
4	0	1	1	1						
5	1	0	0	0						
6	1	0	0	1						
7	1	0	1	0						
8	1	0	1	1						
9	1	1	0	0						

$$\begin{array}{r} \overset{1}{\overset{1}{0}} 011 \\ + 0011 \\ \hline 0110 \end{array} \rightarrow \begin{array}{l} 0 \text{ (En XS 3)} \\ 0 \text{ (En XS 3)} \\ \hline 3 \text{ (En XS 3)} \end{array}$$

El resultado obtenido no es correcto. El resultado debería ser 0011

Código BCD XS 3



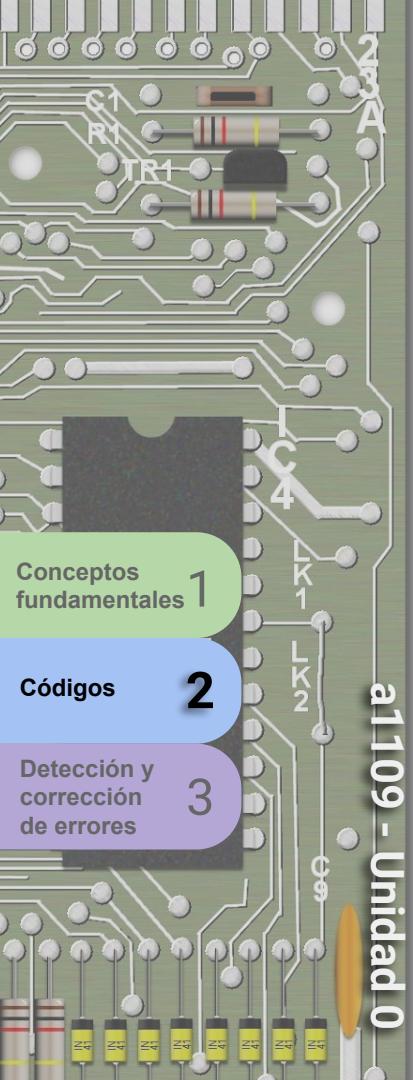
Suma en BCD XS 3

Al sumar dos números en BCD XS 3 obtenemos un número que se encontrará en binario puro más seis , no más tres. Para corregir el resultado y convertirlo nuevamente en exceso tres debemos restar 0011.

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1						
1	0	1	0	0						
2	0	1	0	1						
3	0	1	1	0						
4	0	1	1	1						
5	1	0	0	0						
6	1	0	0	1						
7	1	0	1	0						
8	1	0	1	1						
9	1	1	0	0						

Código BCD XS 3

$$\begin{array}{r} \begin{matrix} & 1 \\ 0011 & + \\ 0011 & \hline \end{matrix} \\[10pt] \begin{matrix} 0110 \\ 0011 \\ \hline \end{matrix} \end{array} \quad \begin{array}{l} \rightarrow 0 \text{ (En XS 3)} \\ \rightarrow 0 \text{ (En XS 3)} \\ \rightarrow 3 \text{ (En XS 3)} \\ \rightarrow 0 \text{ (En XS 3)} \end{array}$$



Suma en BCD XS 3

Realicemos ahora la suma en BCD XS 3 de los números 1 y 9

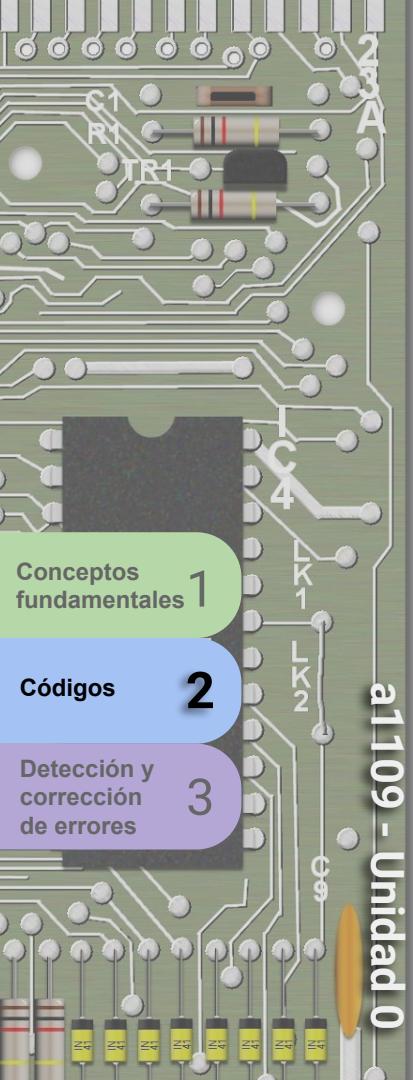
DECIMAL	0	0	1	1
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

$$\begin{array}{r} 1 \\ 0100 \\ + 1100 \\ \hline 0001\ 0000 \end{array}$$

1 (En XS 3)

9 (En XS 3)

Código BCD XS 3



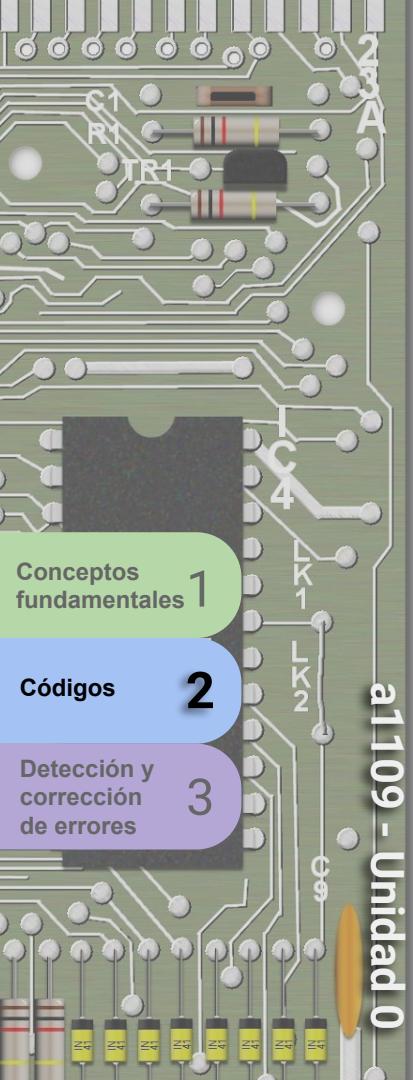
Suma en BCD XS 3

Realicemos ahora la suma en BCD XS 3 de los números 1 y 9

DECIMAL	0	0	0	1	1
0	0	0	1	1	
1	0	1	0	0	
2	0	1	0	1	
3	0	1	1	0	
4	0	1	1	1	
5	1	0	0	0	
6	1	0	0	1	
7	1	0	1	0	
8	1	0	1	1	
9	1	1	0	0	

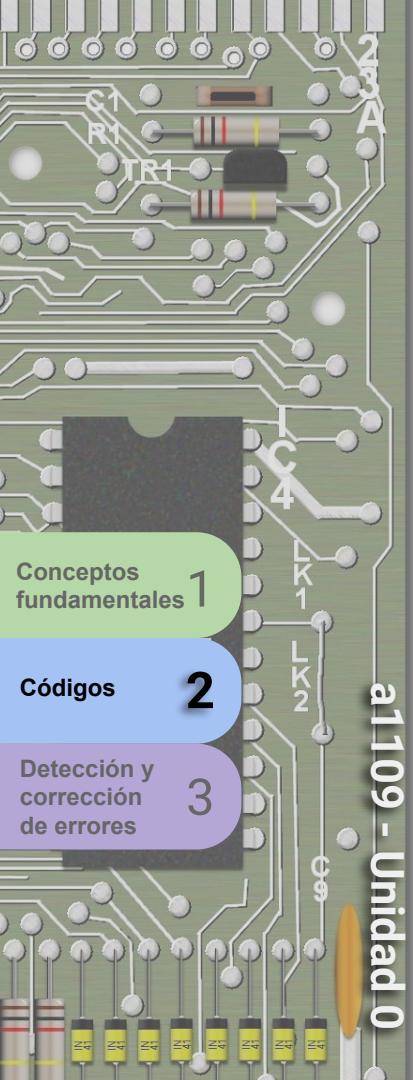
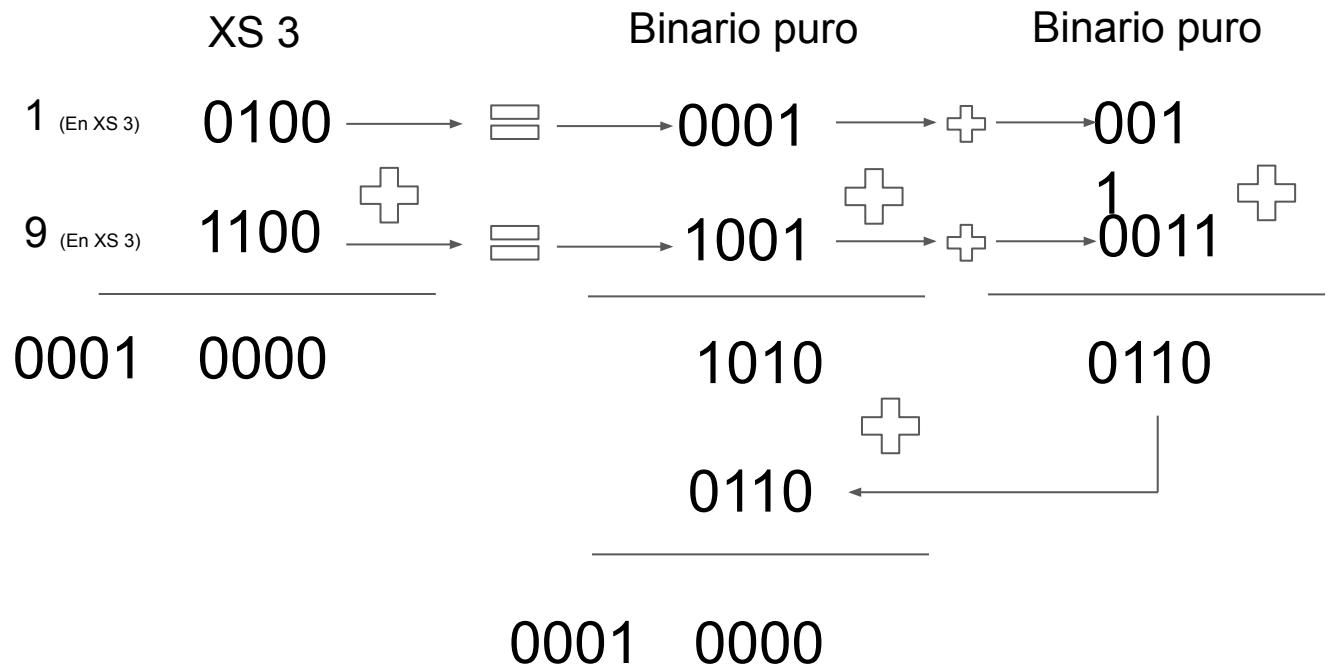
$$\begin{array}{r} 1 \\ 0100 \\ + 1100 \\ \hline 0001\ 0000 \end{array} \rightarrow \begin{array}{l} 1 \text{ (En XS 3)} \\ 9 \text{ (En XS 3)} \\ \text{NO CORRESPONDE} \\ \text{A XS 3} \end{array}$$

Código BCD XS 3



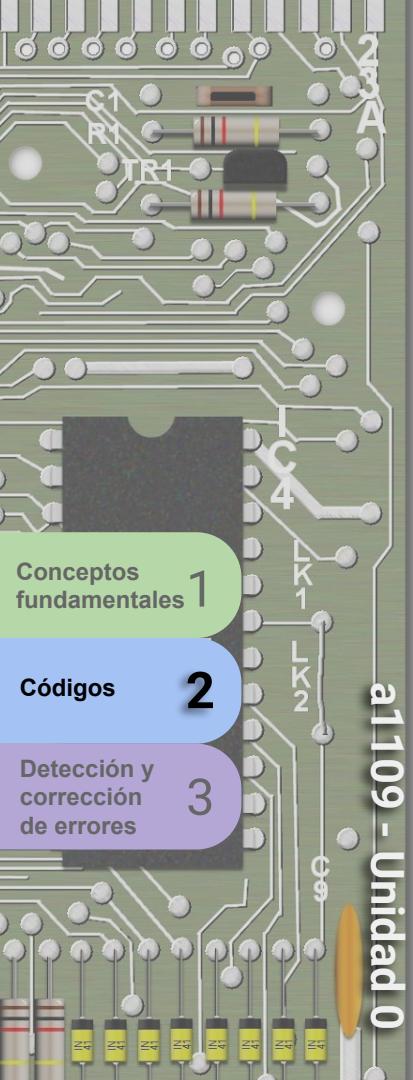
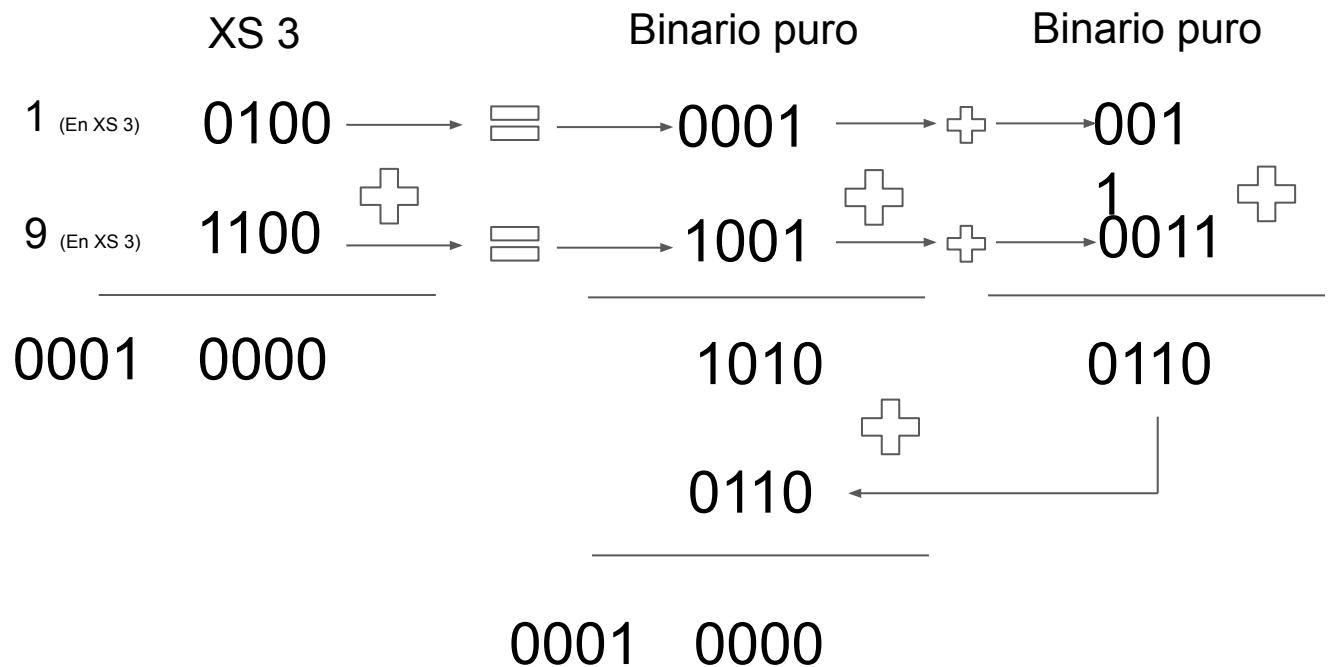
Suma en BCD XS 3

Analicemos con detenimiento la suma de 1 + 9 en XS 3. Para esto realizamos la suma , pero separando los excesos 3 de los números , como se muestra a continuación:



Suma en BCD XS 3

Observemos que al sumar 1+9 en binario puro obtenemos 1010 (10 en decimal). Al sumarle 0110 (la suma de los excesos de 1 y 9 en XS 3) al resultado de 1+9, obtenemos 1 y 0 en binario puro como si estuviéramos trabajando en BCD 8421



Suma en BCD XS 3

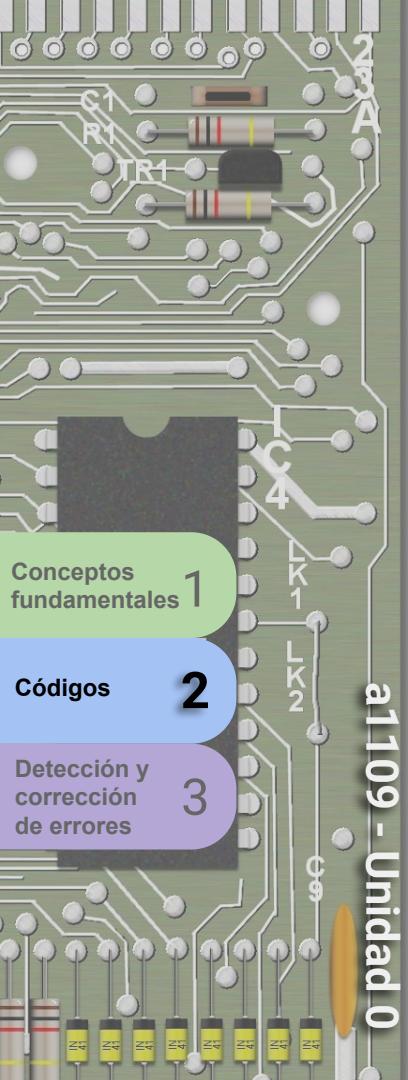
Y en efecto es la misma suma, como se observa en la diapositiva. Cuando en XS 3 la suma de dos números es mayor a 9 el resultado estará expresado como si fuera BCD 8421 ya corregido, debido a que al sumar dos veces el exceso de 3 unidades, estamos sumando seis.

Corrección de la suma 1+9 en
BCD 8421

$$\begin{array}{r} 0 \quad 0001 \\ + \quad 1001 \\ \hline .0 \quad 1010 \\ + \quad 110 \\ \hline .1 \quad 0000 \end{array}$$

Suma de 1+9 en XS 3 descompuesta

XS 3	Binario puro	Binario puro
0100	$\equiv 0001$	$\rightarrow + \rightarrow 001$
1100	$\equiv 1001$	$\rightarrow + \rightarrow 10011$
$1010 \qquad \qquad \qquad 0110$		
0	0110	$\rightarrow + \rightarrow$
$0001 \quad 0000$		



Conceptos fundamentales 1

Códigos 2

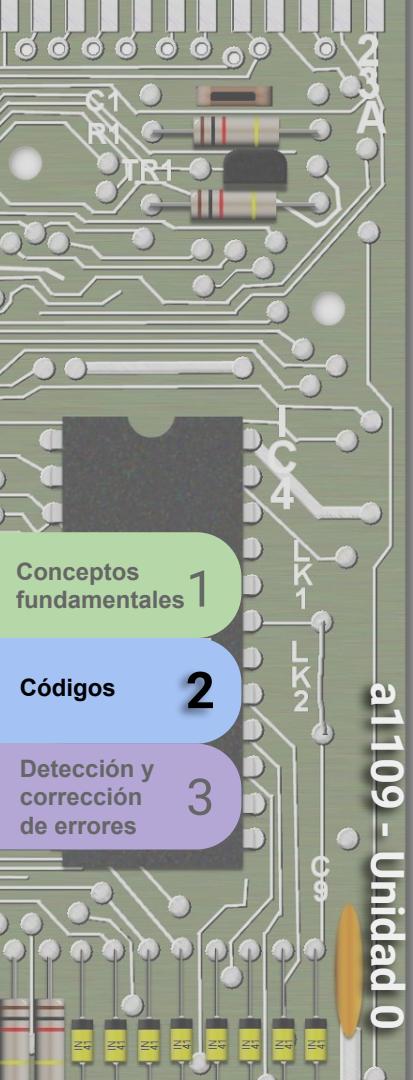
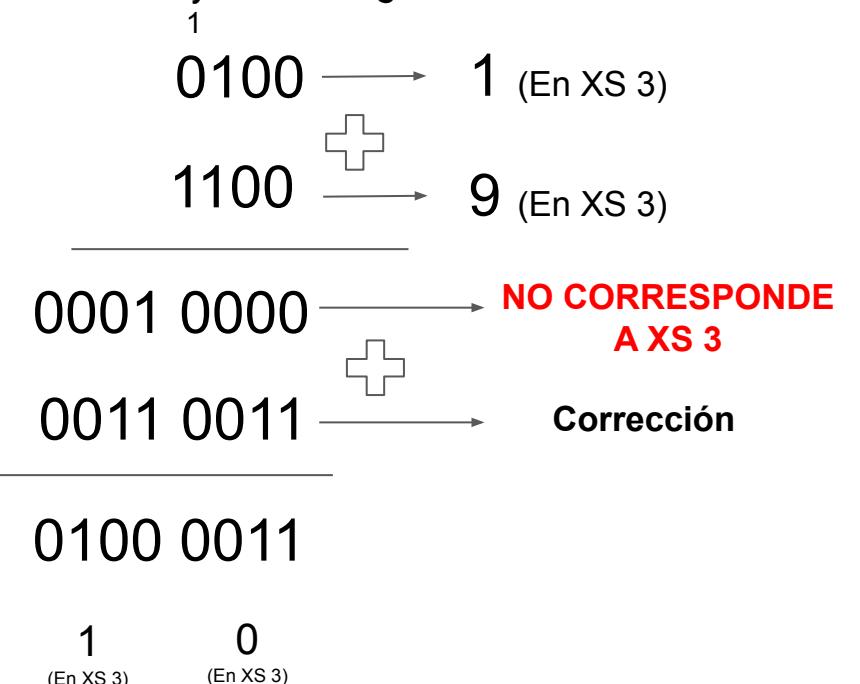
Detección y corrección de errores 3

Suma en BCD XS 3

Es evidente que en estos casos no se debe restar tres al resultado, sino que al ahora encontrarse los números en binario puro se deberá sumar 3 para transformar esos numero a XS 3 y así corregir el resultado.

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1						
1	0	1	0	0						
2	0	1	0	1						
3	0	1	1	0						
4	0	1	1	1						
5	1	0	0	0						
6	1	0	0	1						
7	1	0	1	0						
8	1	0	1	1						
9	1	1	0	0						

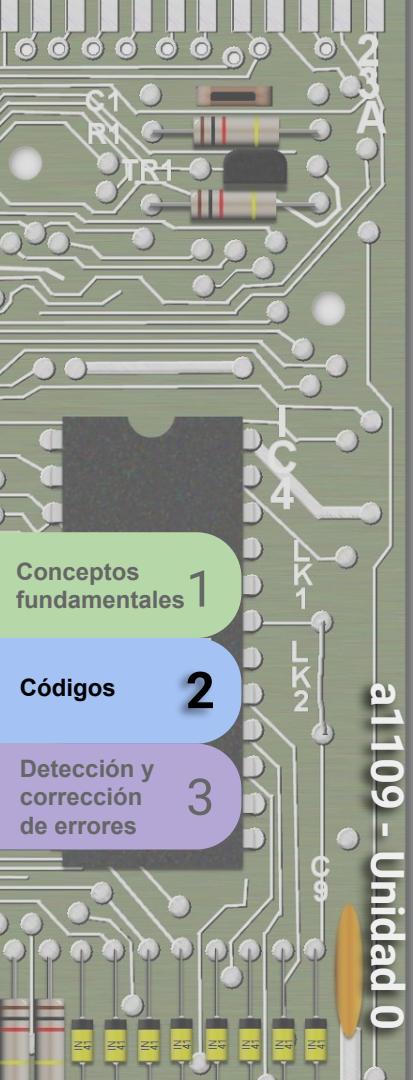
Código BCD XS 3



Suma en BCD XS 3

En resumen , en todos los casos se deben realizar correcciones cuando se realizan sumas en BCD XS 3 y las mismas son:

- Si el resultado de la suma de dos dígitos en BCD XS 3 NO produjo carry, entonces se le debe restar 3 al resultado obtenido
- Si el resultado de la suma de dos dígitos en BCD XS 3 PRODUJO carry, entonces se le debe sumar 3 al resultado obtenido

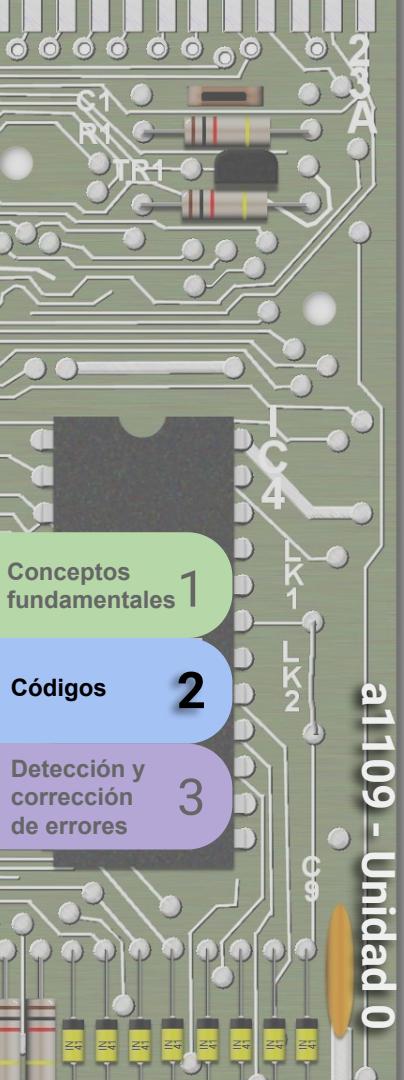


Suma en BCD XS 3

Veamos ahora un ejemplo de una suma. Sumemos 52389+61720 cuyo resultado debería ser 114109

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0
2	0	1	0	1	0	0	1	0	0	0
3	0	1	1	0	0	0	0	1	0	0
4	0	1	1	1	0	0	0	0	1	0
5	1	0	0	0	0	1	0	0	0	0
6	1	0	0	1	0	0	1	0	0	0
7	1	0	1	0	0	0	0	1	0	0
8	1	0	1	1	0	0	0	0	1	0
9	1	1	0	0	0	0	0	0	0	1

$$\begin{array}{r} 52389 = 1000 \quad 0101 \quad 0110 \quad 1011 \quad 1100 \\ 61720 = 1001 \quad 0100 \quad 1010 \quad 0101 \quad 0011 \\ \hline & 0001 \quad 0001 \quad 1010 \quad 0001 \quad 0000 \quad 1111 \end{array}$$



Código BCD XS 3

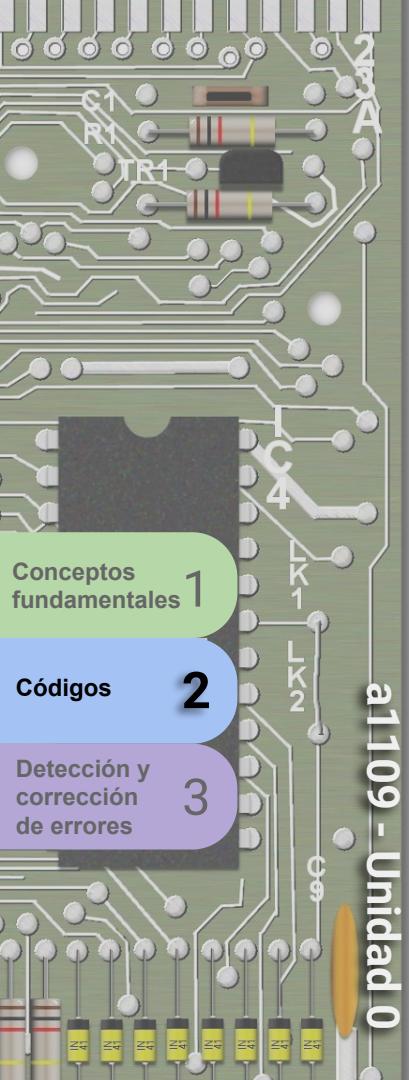
Suma en BCD XS 3

Ya habiendo sumado todos los números procederemos a corregir los resultados sumando o restando según hemos explicado.

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0
2	0	1	0	1	0	0	1	0	0	0
3	0	1	1	0	0	0	1	1	0	0
4	0	1	1	1	0	0	1	1	1	0
5	1	0	0	0	0	1	0	0	0	0
6	1	0	0	1	0	0	0	1	0	1
7	1	0	1	0	0	0	1	0	1	0
8	1	0	1	1	0	0	1	0	1	1
9	1	1	0	0	0	0	0	0	0	0

$$\begin{array}{r} 52389 = \quad \overset{1}{\text{1}} \quad 1000 \quad 0101 \quad \overset{1}{\text{1}} \quad \overset{1}{\text{1}} \quad \overset{1}{\text{1}} \quad 1100 \\ 61720 = \quad \overset{1}{\text{1}} \quad 1001 \quad 0100 \quad 1010 \quad 0101 \quad 0011 \\ \hline & \text{0001} & \text{0001} & \text{1010} & \text{0001} & \text{0000} & \text{1111} \\ & +0011 & +0011 & -0011 & +0011 & +0011 & -0011 \end{array}$$

Código BCD XS 3



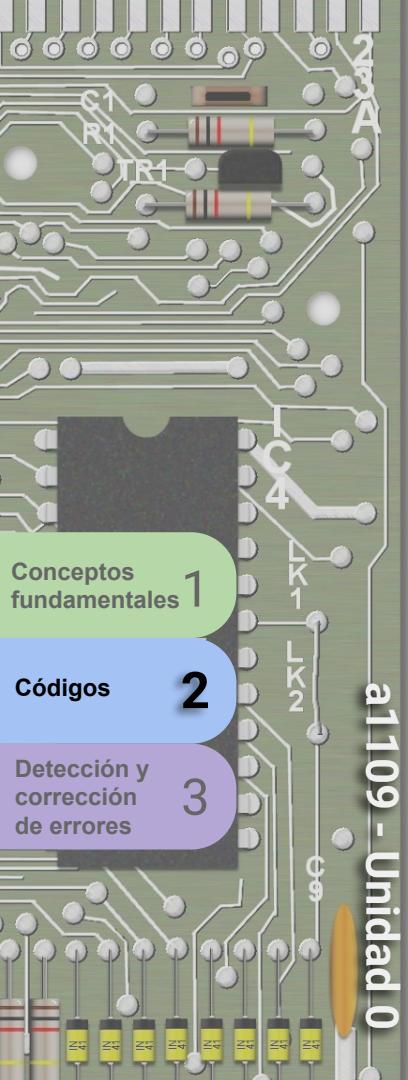
Suma en BCD XS 3

Sumamos y restamos según sea el caso y obtenemos 114109 en XS 3 que es el resultado esperado

DECIMAL	0	1	2	3	4	5	6	7	8	9
0	0	0	1	1	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	1
2	0	1	0	1	0	0	1	0	1	0
3	0	1	1	0	0	0	1	1	0	1
4	0	1	1	1	0	0	1	1	1	0
5	1	0	0	0	0	1	0	0	0	1
6	1	0	0	1	0	0	0	1	0	1
7	1	0	1	0	0	1	0	1	0	0
8	1	0	1	1	0	0	1	0	1	1
9	1	1	0	0	0	0	0	0	0	0

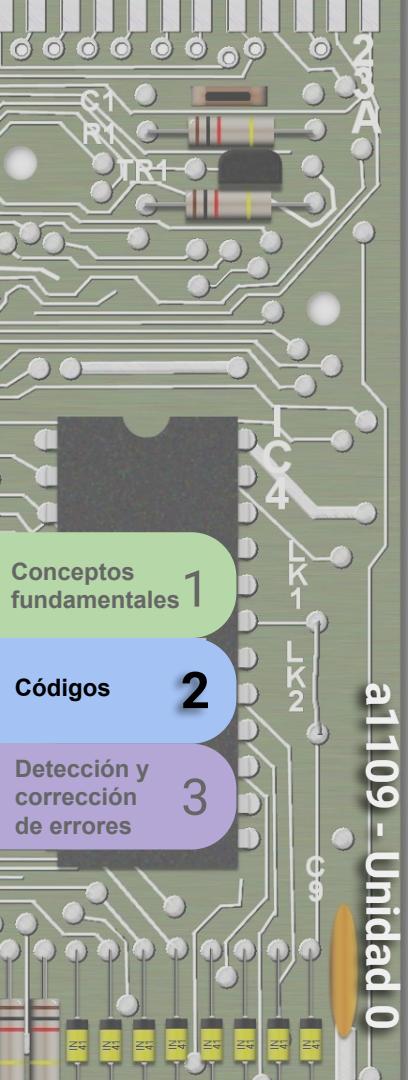
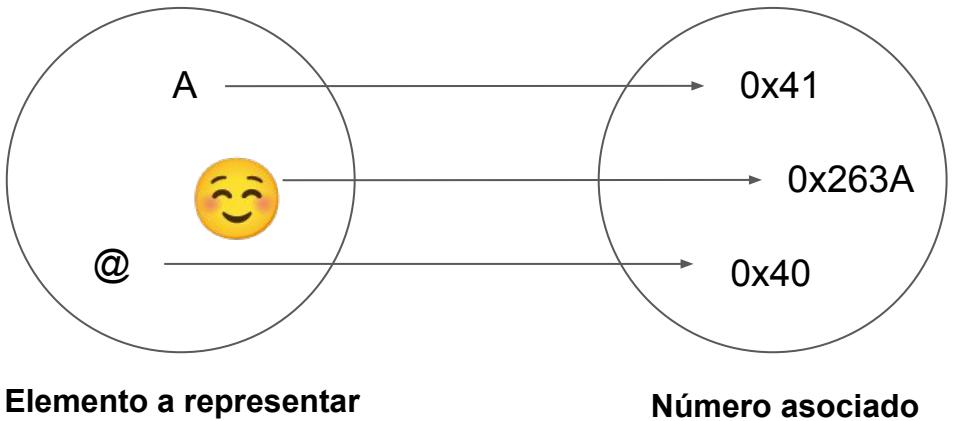
$$\begin{array}{r} 52389 = 1000 \quad 0101 \quad 0110 \quad 1011 \quad 1100 \\ 61720 = 1001 \quad 0100 \quad 1010 \quad 0101 \quad 0011 \\ \hline & 0001 \quad 0001 \quad 1010 \quad 0001 \quad 0000 \quad 1111 \\ & +0011 \quad +0011 \quad -0011 \quad +0011 \quad +0011 \quad -0011 \\ \hline & 0100 \quad 0100 \quad 0111 \quad 0100 \quad 0011 \quad 1100 \\ & 1 \quad 1 \quad 4 \quad 1 \quad 0 \quad 9 \end{array}$$

Código BCD XS 3



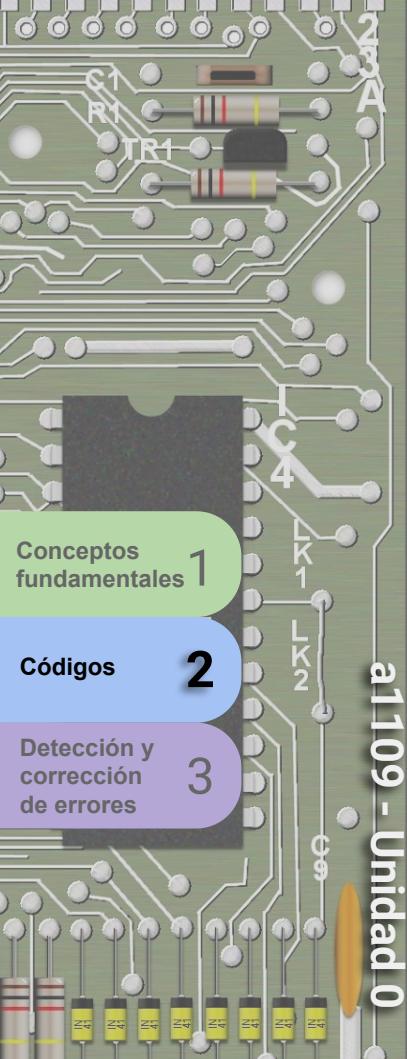
Códigos alfanuméricicos

En una computadora necesitamos representar más que cifras, también se necesitan representar letras, caracteres de control , símbolos y números. Por lo tanto necesitaremos códigos que permitan representar a estos elementos. Los códigos que cumplen esta función se denominan códigos alfanuméricos.



Códigos alfanuméricicos- Código ASCII

Dec	Hex	Oct	Car.	Dec	Hex	Oct	Car.	Dec	Hex	Oct	Car.	Dec	Hex	Oct	Car.
0	00	000	NUL (null)	32	20	040	SPACE	64	40	100	Ø	96	60	140	`
1	01	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	02	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	03	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	04	004	EOT (end of transmission)	36	24	044	\$	68	44	104	D	100	64	144	d
5	05	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	06	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	07	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	08	010	BS (backspace)	40	28	050	(72	48	110	H	104	68	150	h
9	09	011	TAB (horizontal tab)	41	29	051)	73	49	111	I	105	69	151	i
10	0A	012	LF (NL line feed)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF (NP form feed)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	0E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL



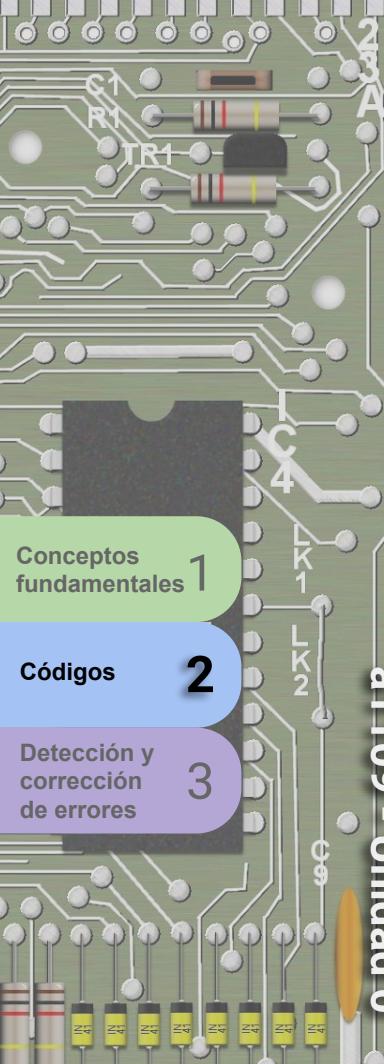
Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

Códigos alfanuméricos- ASCII extendido

Dec	Hex	Oct	Car												
128	80	200	ç	160	A0	240	á	192	C0	300	+	224	E0	340	ó
129	81	201	ü	161	A1	241	í	193	C1	301	-	225	E1	341	ß
130	82	202	é	162	A2	242	ó	194	C2	302	-	226	E2	342	ô
131	83	203	ã	163	A3	243	ú	195	C3	303	+	227	E3	343	ö
132	84	204	ä	164	A4	244	ñ	196	C4	304	-	228	E4	344	ë
133	85	205	à	165	A5	245	ñ	197	C5	305	+	229	E5	345	đ
134	86	206	â	166	A6	246	*	198	C6	306	ã	230	E6	346	µ
135	87	207	ç	167	A7	247	°	199	C7	307	À	231	E7	347	þ
136	88	210	è	168	A8	250	ż	200	C8	310	+	232	E8	350	ń
137	89	211	ë	169	A9	251	ı	201	C9	311	+	233	E9	351	ú
138	8A	212	è	170	AA	252	ń	202	CA	312	-	234	EA	352	ò
139	8B	213	ï	171	AB	253	ȝ	203	CB	313	-	235	EB	353	û
140	8C	214	í	172	AC	254	ȝ	204	CC	314	ı	236	EC	354	ý
141	8D	215	í	173	AD	255	ı	205	CD	315	-	237	ED	355	ÿ
142	8E	216	Ã	174	AE	256	«	206	CE	316	+	238	EE	356	—
143	8F	217	Â	175	AF	257	»	207	CF	317	»	239	EF	357	‘
144	90	220	í	176	B0	260	—	208	DO	320	ð	240	F0	360	-
145	91	221	æ	177	B1	261	—	209	D1	321	ð	241	F1	361	±
146	92	222	È	178	B2	262	—	210	D2	322	È	242	F2	362	—
147	93	223	ð	179	B3	263	ı	211	D3	323	È	243	F3	363	ȝ
148	94	224	ö	180	B4	264	ı	212	D4	324	È	244	F4	364	ı
149	95	225	ð	181	B5	265	Ã	213	D5	325	ı	245	F5	365	ȝ
150	96	226	ð	182	B6	266	Ã	214	D6	326	ı	246	F6	366	—
151	97	227	ú	183	B7	267	Ã	215	D7	327	ı	247	F7	367	—
152	98	230	ÿ	184	B8	270	ø	216	D8	330	ı	248	F8	370	—
153	99	231	ö	185	B9	271	ı	217	D9	331	+	249	F9	371	—
154	9A	232	Ü	186	BA	272	ı	218	DA	332	+	250	FA	372	—
155	9B	233	ø	187	BB	273	+	219	DB	333	-	251	FB	373	ı
156	9C	234	£	188	BC	274	+	220	DC	334	-	252	FC	374	—
157	9D	235	Ø	189	BD	275	¢	221	DD	335	ı	253	FD	375	—
158	9E	236	×	190	BE	276	¥	222	DE	336	ı	254	FE	376	—
159	9F	237	ƒ	191	BF	277	+	223	DF	337	-	255	FF	377	(Espacio)



Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

Códigos alfanuméricicos

El código ASCII, de amplio uso , no soluciono por completo la necesidad de representar una gran cantidad de caracteres de distintos idiomas

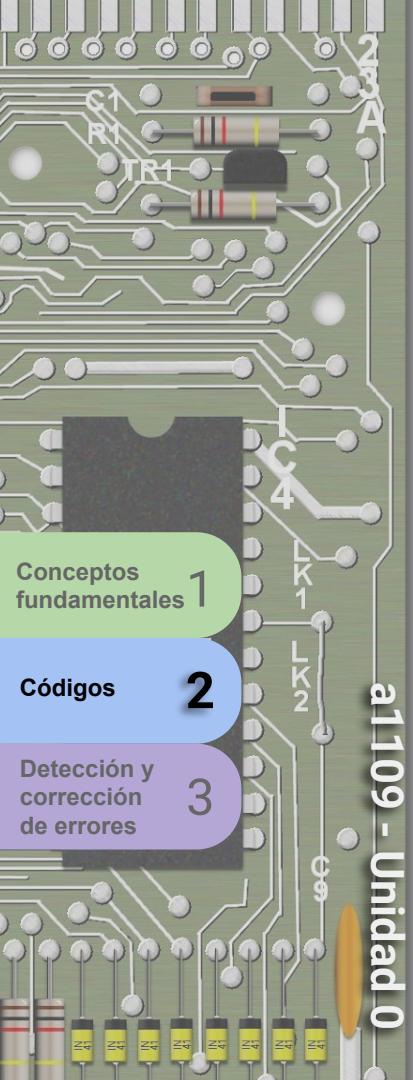
Otros códigos , como UNICODE, tratan de alcanzar este fin incluyendo además de los caracteres látinos , ideogramas e incluso emojis. Para este fin se utilizan 32 bits.



0x0001F4BB

El emoji de laptop corresponde a 0X0001F4BB en UNICODE

UNICODE además respeta el código ASCII de 7 bits , manteniendo así compatibilidad



Códigos alfanuméricos-Unicode

Codifiquemos la palabra HAL en Unicode letra a letra

H Unicode:

00000000 00000000 00000000 01001000

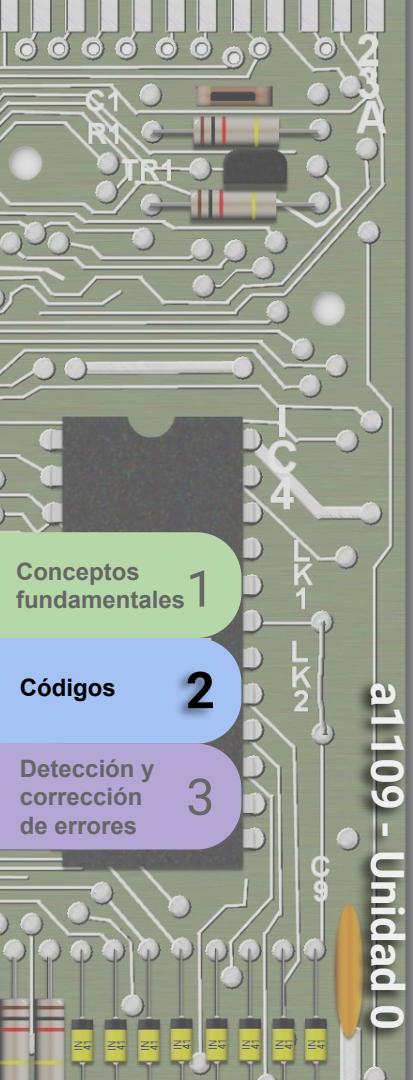
A Unicode:

00000000 00000000 00000000 01000001

L Unicode:

00000000 00000000 00000000 01001100

Como se mencionó, Unicode utiliza 32 bits para los diversos caracteres que representa , sin embargo podemos ver que la cantidad de bits en 0 , sin información , es voluminosa (en el caso de la palabra HAL se utilizaron 75 bits en 0). Sería ideal utilizar la cantidad mínima posible de bits para representar cada carácter.



Conceptos
fundamentales 1

Códigos 2

Detección y
corrección
de errores 3

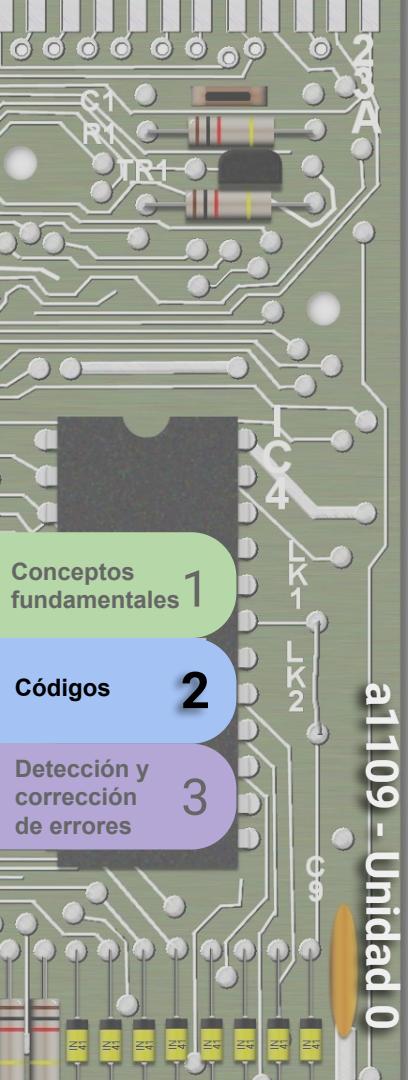
Códigos alfanuméricos-Unicode/ UTF-8

Para solucionar este problema fue creada la codificación UTF (Unicode Transformation Format). En UTF cada carácter , según el caso, puede ocupar una **cantidad variable de bits**, tratando de evitar los bits en 0 que no aportan demasiada información. Existen tres variables de UTF: UTF-8 , UTF-16 y UTF-32.

Comencemos comprendiendo cómo se representa en UTF-8 la letra 'A'

A UTF-8
01000001

En UTF-8 la letra A se representa de la misma forma que en Unicode pero sin utilizar los ceros a la izquierda, por lo tanto solamente se utilizara un byte. Cualquier carácter que se represente con 7 bits en Unicode ocupará un byte en UTF-8 y su bit más significativo será siempre 0.

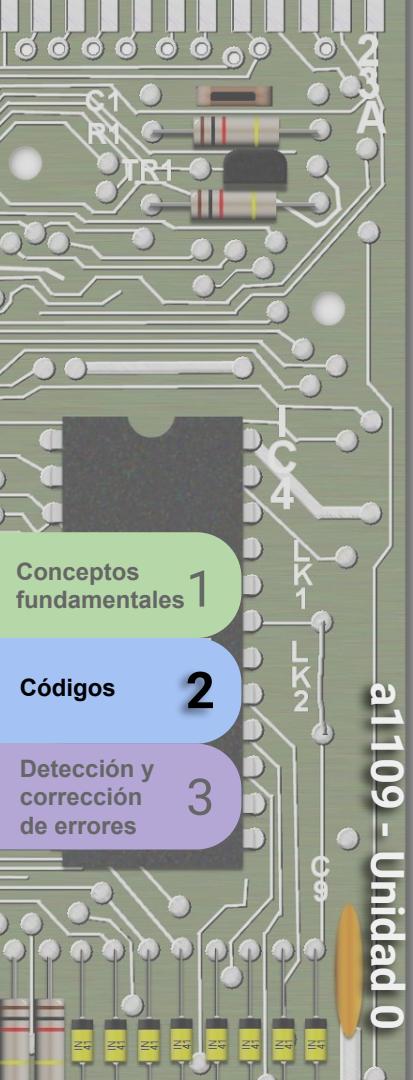


Códigos alfanuméricos-Unicode/ UTF-8

¿Pero qué sucede en el caso de que se desee representar el carácter Ñ? La Ñ en Unicode corresponde al valor 0...11010001, siendo solamente 8 la cantidad de bits útiles. Por lo tanto , para sortear esta situación UTF-8 propone lo siguiente:

En primer lugar no bastará con un byte sino que se necesitaran 2 bytes para representar el carácter Ñ utilizando UTF-8

00000000 00000000



Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

a1109 - Unidad 0

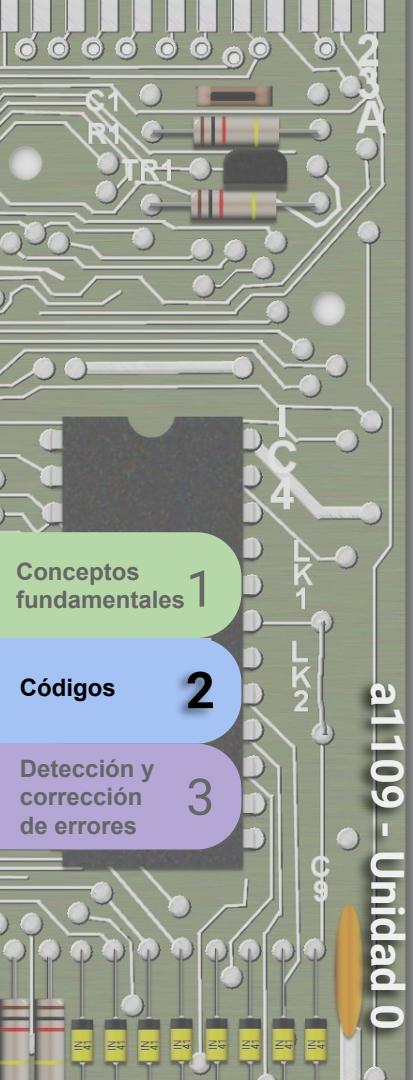
Códigos alfanuméricos-Unicode/ UTF-8

¿Pero qué sucede en el caso de que se desee representar el carácter Ñ? La Ñ en Unicode corresponde al valor 0...11010001, siendo solamente 8 la cantidad de bits útiles. Por lo tanto , para sortear esta situación UTF-8 propone lo siguiente:

En primer lugar no bastará con un byte sino que se necesitaran 2 bytes para representar el carácter Ñ utilizando UTF-8

11000000 00000000

El byte más significativo comenzará con una marca que contendrá tantos bits en 1 como bytes se necesiten para representar al carácter, terminando esta marca con un cero. En este caso necesitaremos 2 bytes por lo tanto se colocará **110**



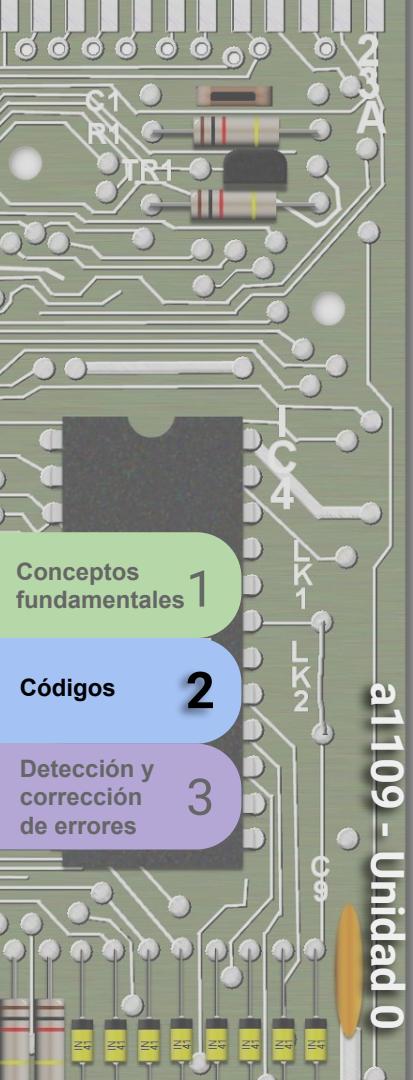
Códigos alfanuméricos-Unicode/ UTF-8

¿Pero qué sucede en el caso de que se desee representar el carácter Ñ? La Ñ en Unicode corresponde al valor 0...11010001, siendo solamente 8 la cantidad de bits útiles. Por lo tanto , para sortear esta situación UTF-8 propone lo siguiente:

En primer lugar no bastará con un byte sino que se necesitaran 2 bytes para representar el carácter Ñ utilizando UTF-8

11000000 10000000

Los bytes de continuación comenzarán con 10 como se puede observar en la diapositiva



Códigos alfanuméricos-Unicode/ UTF-8

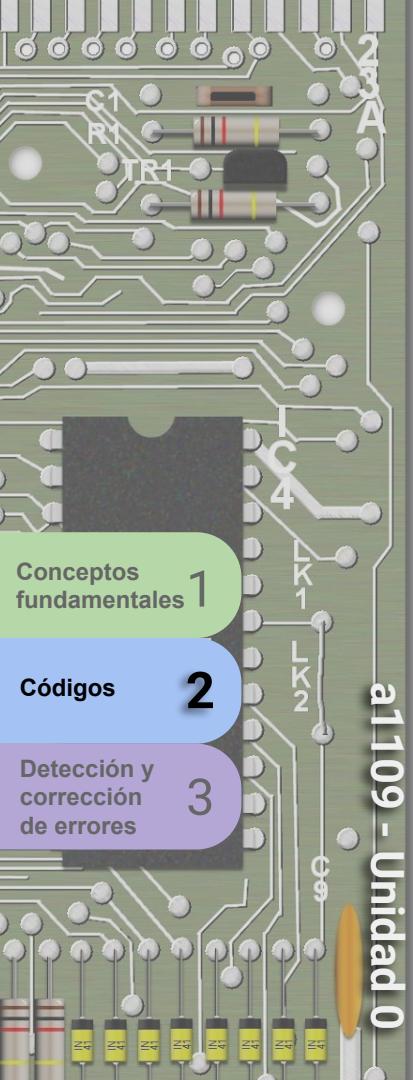
¿Pero qué sucede en el caso de que se desee representar el carácter Ñ? La Ñ en Unicode corresponde al valor 0...11010001, siendo solamente 8 la cantidad de bits útiles. Por lo tanto , para sortear esta situación UTF-8 propone lo siguiente:

En primer lugar no bastará con un byte sino que se necesitaran 2 bytes para representar el carácter Ñ utilizando UTF-8

11000011 10010001

Bits con información útil de Ñ en Unicode=11010001

En los bits que quedan libres se colocaran los bits con información útil que corresponda a la Ñ. En este caso se pudo ahorrar 2 bytes que estarían en 0. Con 2 bytes en UTF-8 podemos representar caracteres de hasta 11 bits (a partir del 0x80 hasta el 0x7FF, los caracteres menores a 0x80 solo necesitan 1 byte)



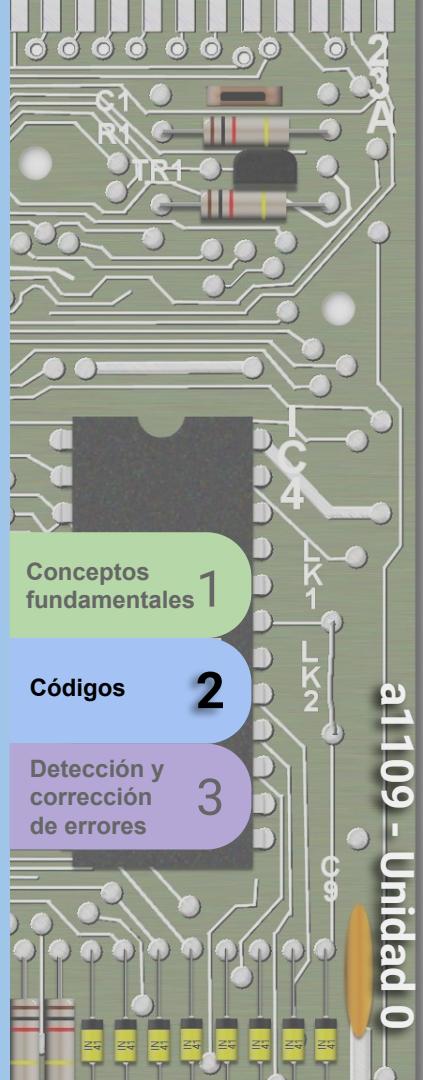
Códigos alfanuméricos-Unicode/ UTF-8

Intentemos representar ahora el emoji  , en UTF-8 . De este emoji ya habíamos dado su código Unicode 0x0001F4BB. Representemos primero esta palabra en binario:

0000 0000 0000 0001 1111 0100 1011 1011

Los datos útiles son los que están marcados en rojo, y es evidente que necesitaremos más de 7 bits para poder representar este emoji. Incluso necesitaremos más de dos bytes para poder representarlo en UTF-8. Como mencionamos anteriormente con dos bytes, en este sistema, se pueden representar caracteres de hasta 11 bits, y en este caso, este carácter posee 17 bits de información útil. Veamos con 3 bytes:

11100000 10000000 10000000



Códigos alfanuméricos-Unicode/ UTF-8

11100000 10000000 10000000

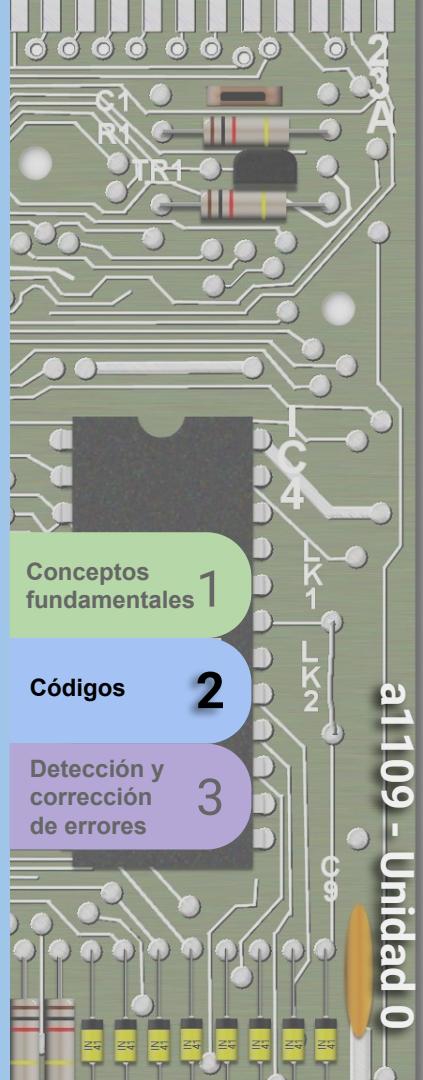
Cumpliendo con las reglas mencionadas anteriormente para representar un carácter con más de 7 bits en UTF-8, vemos que utilizando 3 bytes no es suficiente ya que solamente podemos usar 16 bits , por lo tanto necesitaremos 4 bytes.

11110000 10000000 10000000 10000000

Ahora podemos representar caracteres que posean hasta 21 bits. El paso siguiente para representar el emoji  es trivial

11110000 10011111 10010010 10111011

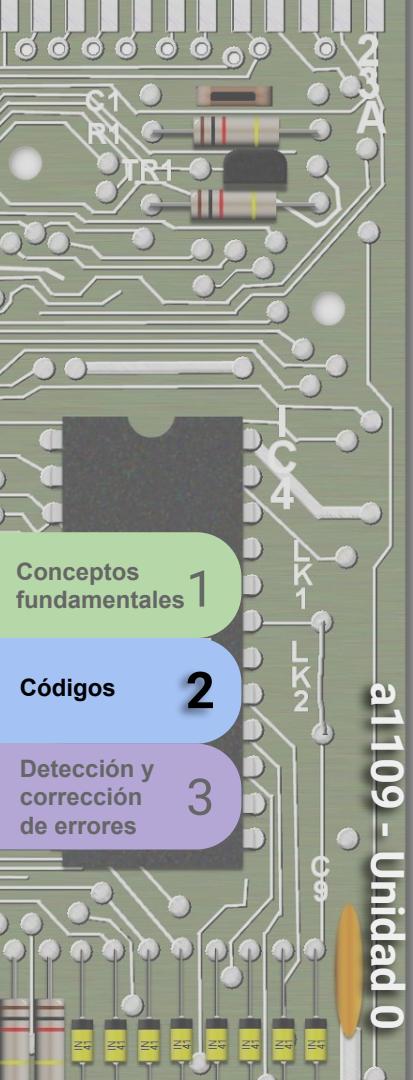
1 11110100
10111011



Códigos alfanuméricos-Unicode/ UTF-8

La siguiente tabla resume la forma de codificar los caracteres unicode en UTF-8

Rango de representación de caracteres	Primer byte	Segundo byte	Tercer byte	Cuarto byte
0x0000-0x007F	0xxxxxx	-	-	-
0x0080-0x07FF	110xxxx	10xxxxxx	-	-
0x0800-0xFFFF	1110xxxx	10xxxxxx	10xxxxxx	-
0x10000-0x10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx



Distancia entre los elementos de un código

Se define como distancia entre dos elementos cualesquiera de un código a la cantidad de bits que deben modificarse para llegar desde uno de esos elementos al otro. Tomando como ejemplo el código BCD 8421 visto anteriormente:

A	B	C	D	DECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Tomamos dos elementos cualesquiera

Observamos la cantidad de elementos que se necesitan modificar para pasar de un elemento al otro

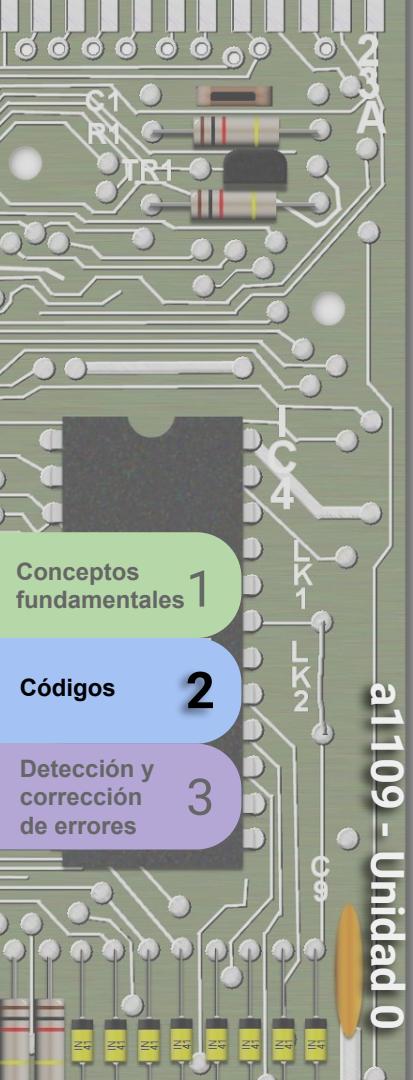
0011

Distancia
2

1001

2

La distancia entre estos dos elementos es 2



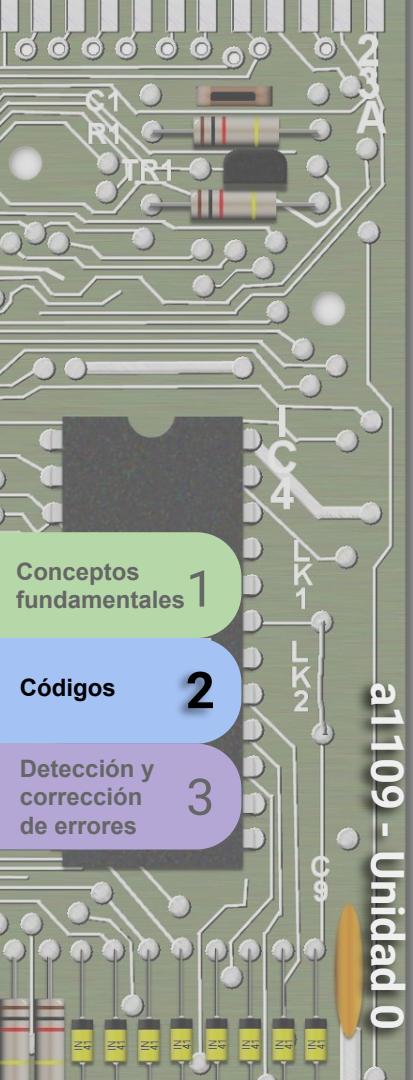
Distancia mínima de un código

Se define como distancia mínima de un código o simplemente distancia de un código a la menor de todas las distancias que pueden encontrarse entre los distintos pares de combinaciones que forman el código en cuestión. Para el caso del código BCD 8421 es simple encontrar su distancia o distancia mínima:

A	B	C	D	DECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

00000
00011 → **Distancia 1**

La distancia del código BCD 8421 o la distancia mínima del BCD 8421 es 1

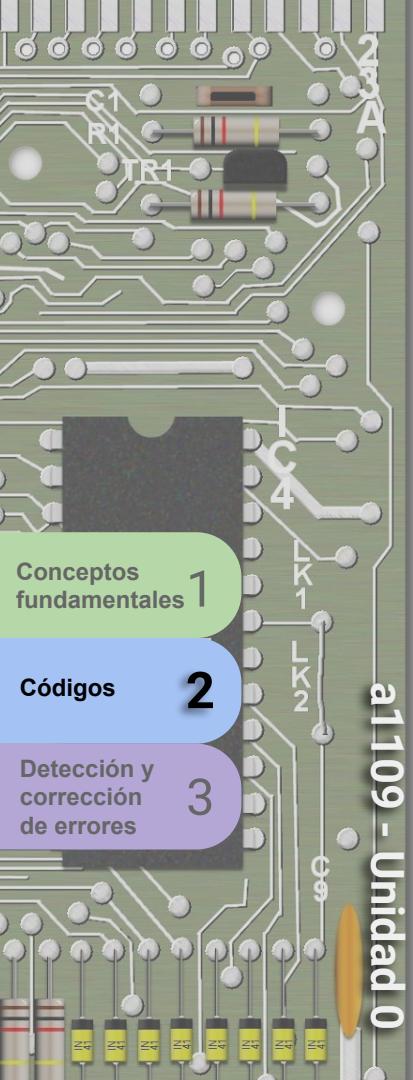


Paridad

Un bit de paridad se define como un bit que se agrega a todas las combinaciones de un código, con el objeto de que cada combinación presente el mismo aspecto en cuanto a la cantidad de unos y ceros que las forman.

Así un código está definido con paridad **par** en los unos si se configuran todas las combinaciones del mismo con un número par de bits con valor en uno. Analicemos el siguiente ejemplo:

Código de módulo 4	Analizamos la cantidad de unos	Bit de paridad	Nuevo código	Paridad de unos
000	Paridad de unos par (0 unos)	0	0000	PAR
001	Paridad de unos impar (1 uno)	1	0011	PAR
011	Paridad de unos par (2 unos)	0	0110	PAR
111	Paridad de unos impar (3 unos)	1	1111	PAR



Conceptos fundamentales 1

Códigos 2

Detección y corrección de errores 3

Paridad

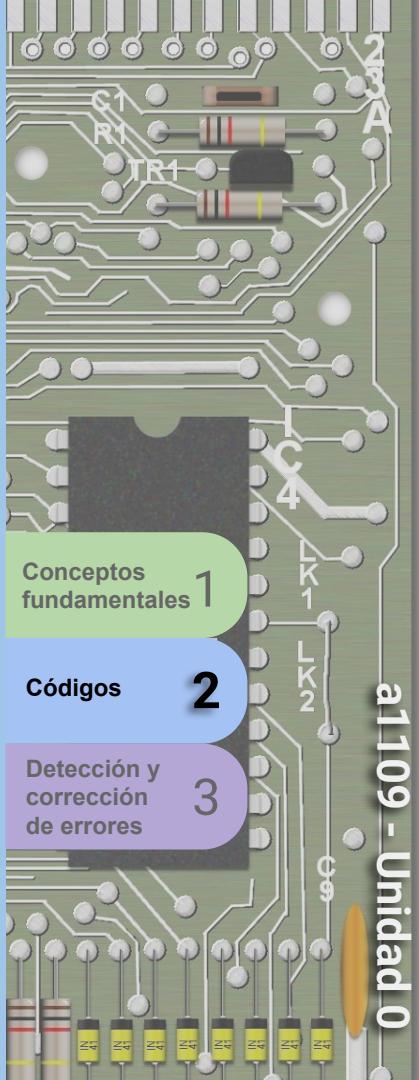
Análogamente un código está definido con paridad **ímpar** en los unos si se configuran todas las combinaciones del mismo con un número ímpar de bits con valor en uno. Analizemos el siguiente ejemplo:

Código de módulo 4	Analizamos la cantidad de unos	Bit de paridad	Nuevo código	Paridad de unos
000	Paridad de unos par (0 unos)	1	0001	IMPAR
001	Paridad de unos ímpar (1 uno)	0	0010	IMPAR
011	Paridad de unos par (2 unos)	1	0111	IMPAR
111	Paridad de unos ímpar (3 unos)	0	1110	IMPAR

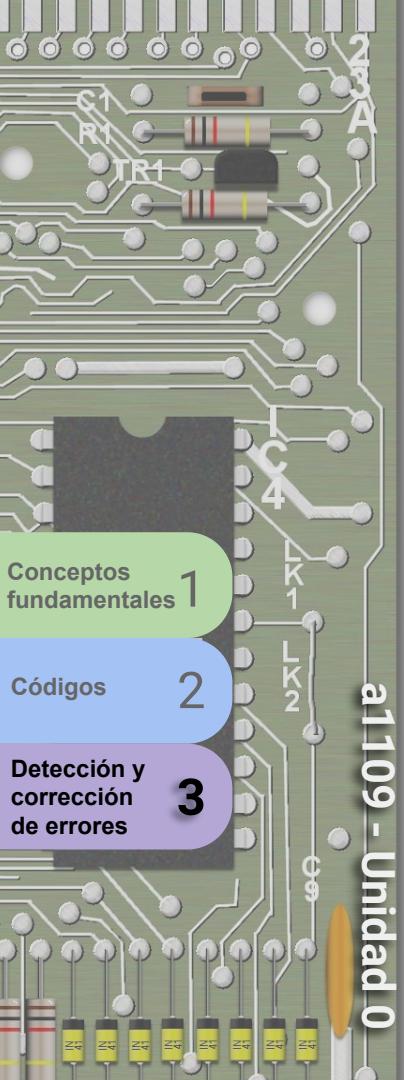
Notar:

*Que al definir la paridad en los unos (en códigos con palabras fijas) también se define la paridad en los ceros

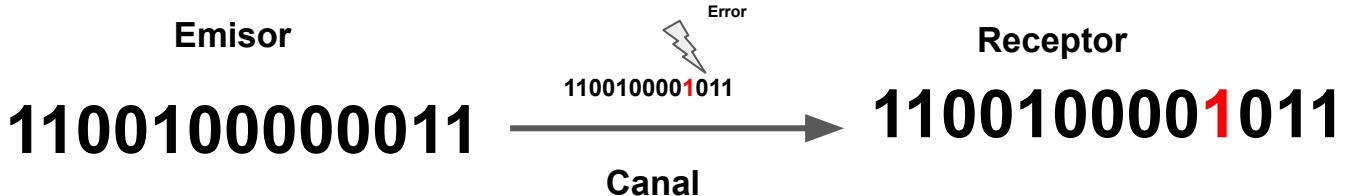
*La distancia del código cambio: al agregar un bit de paridad aumenta en 1 la distancia del código (En este caso de 1 a 2)



Detección y corrección de errores

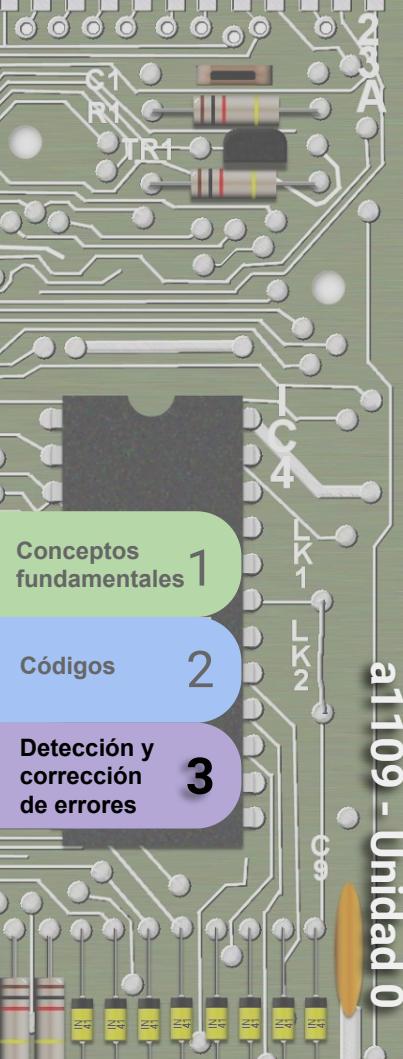


Detección y corrección de errores



Los errores en las comunicaciones suelen ser *comunes*, siendo situaciones que pueden ser accidentales (o no) y por lo tanto difíciles de prevenir, aunque pueden ser detectados e incluso en ciertos casos pueden llegar a corregirse.

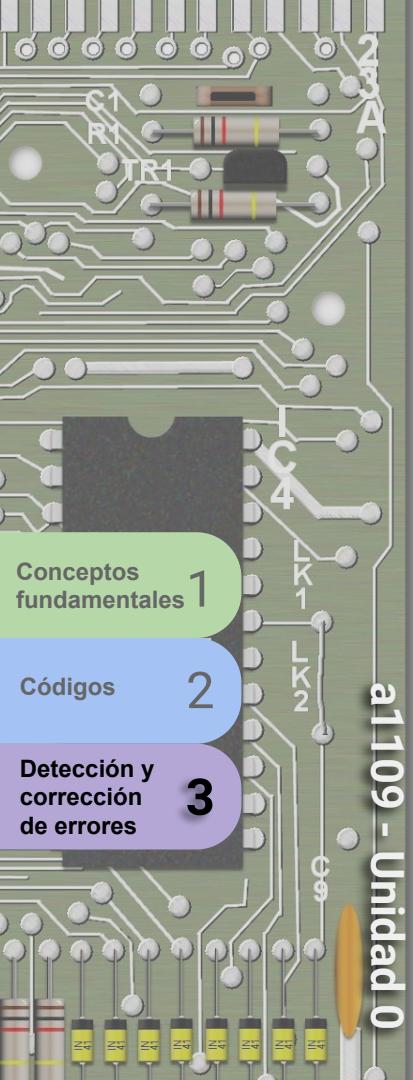
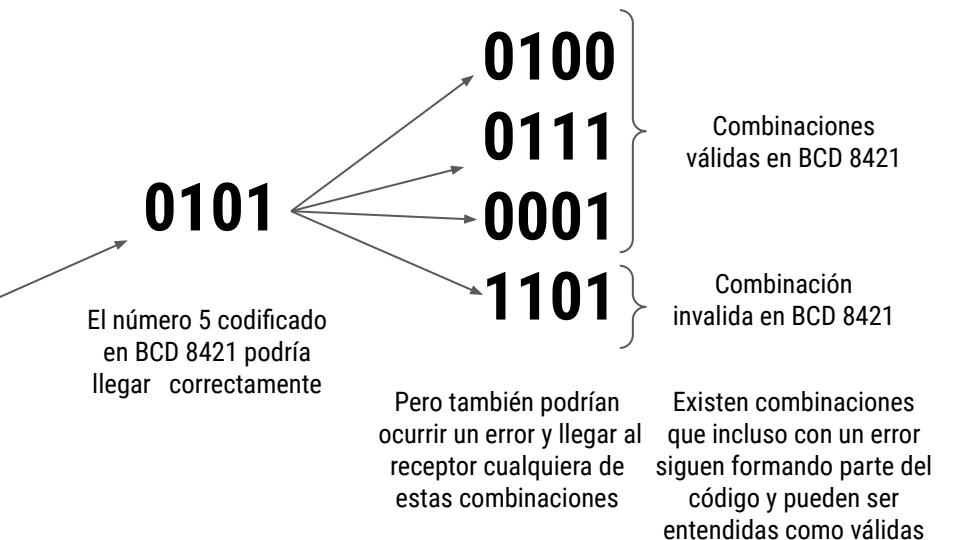
Para ser capaces de corregir un error lo primero que se debe hacer es detectarlo. Detectar un error es comprender, de alguna forma, que la información recibida es inválida.



Detección y corrección de errores

Analicemos la siguiente situación: imaginemos que se desea enviar el número 5 codificado en BCD 8421 por un canal en el que solo puede ocurrir un solo error, es decir un solo cambio de bit.

A	B	C	D	DECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

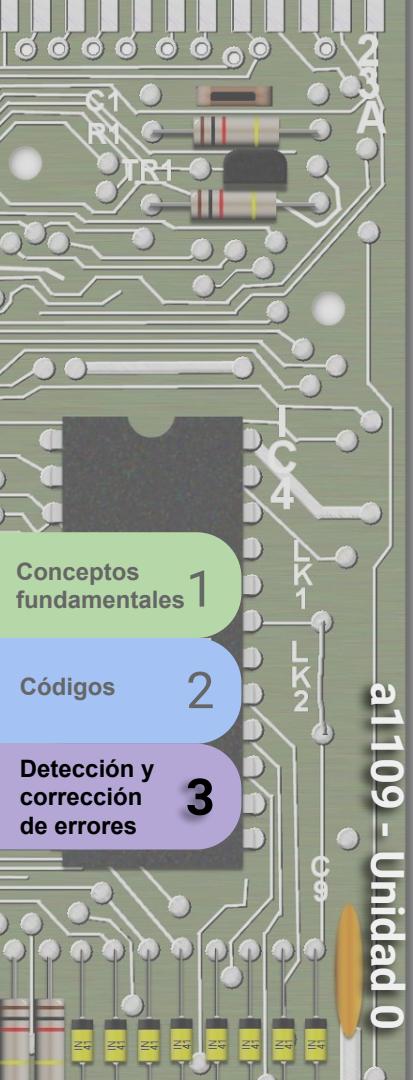


Detección y corrección de errores

Para realizar un mejor análisis de esta situación podemos representar el código BCD 8421 en un mapa. En este mapa AB estará representado por un código Gray de dos bits en el eje Y y CD estará representado igualmente por un código Gray también de dos bits. Por lo tanto si deseamos representar el 5 (0101) lo colocaremos donde AB=01 y donde CD=01 como se puede observar en el gráfico.

		CD				
		AB/CD	00	01	11	10
AB	00	0	1	3		
	01	4	5	7		
	11					
	10	8	9			

■ Combinación válida
■ Combinación invalida



Detección y corrección de errores

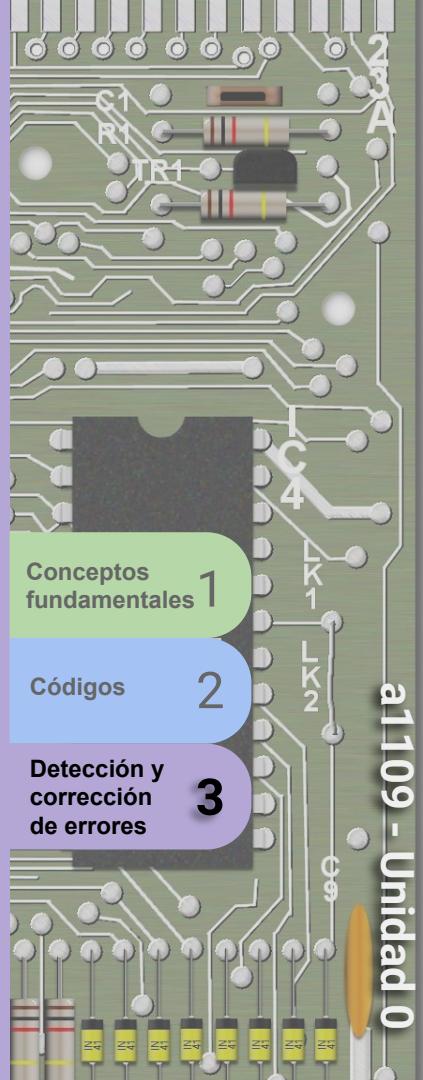
Al utilizar un código gray en ambos ejes , podemos observar con facilidad, cuales pueden ser las posibles combinaciones resultantes al variar un bit de la palabra correspondiente al número 5, como muestran las flechas. Note que solamente las flechas apuntan hacia los costados y hacia arriba y abajo, si nos movieramos diagonalmente, la cantidad de bits que cambiarían sería 2.

CD

AB/CD	00	01	11	10
00	0	1	3	
01	4	5	7	
11				
10	8	9		

AB

Combinación válida
Combinación invalida

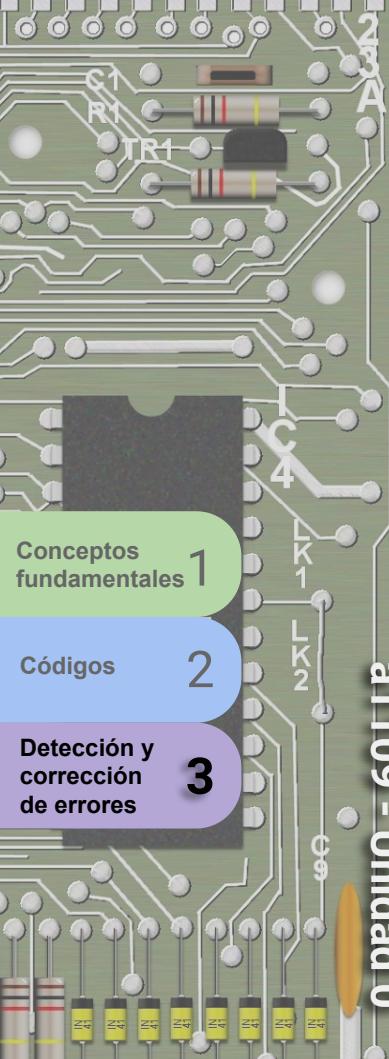


Detección y corrección de errores

Continuando con el ejemplo, que el canal de comunicación pueda producir un error en un bit, significa que la *distancia* entre la palabra del código que se quiso enviar y la palabra que contiene el error es de 1. Pero la distancia del código BCD 8421 también es 1, y es por este motivo que no podemos discernir cuando una palabra recibida contiene un error o no con total certeza (menos en las situaciones donde la palabra esté fuera del código como el caso de 1101)

AB/CD	00	01	11	10
00	0	1	3	
01	4	5	7	
11				
10	8	9		

- Combinación válida
- Combinación invalida



Detección y corrección de errores

Una forma para evitar este inconveniente es aumentar la distancia del código. Hemos visto en diapositivas anteriores que agregar un bit de paridad aumenta en 1 la distancia de cualquier código.

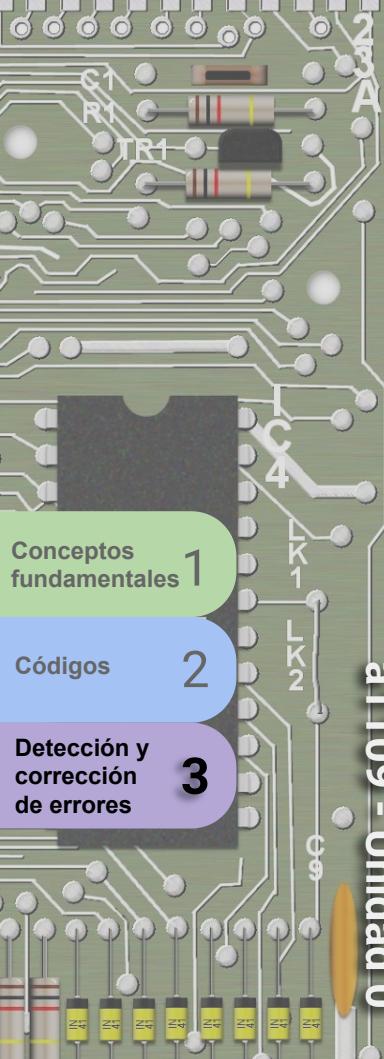
Aumentemos entonces la distancia del código BCD 8421 a dos agregando un bit de paridad. Imaginemos ahora que por el mismo canal de transmisión se desea enviar la palabra 01001 (4) en este código. Si la palabra que se envió posee un error, la distancia entre la palabra recibida y la palabra enviada será de 1 y dicha palabra estará fuera del código. Veamos esto en un ejemplo:

BCD 8421 con paridad				
A	B	C	D	Paridad
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0

01001

El número 4 codificado en
BCD 8421 podría llegar
correctamente...

Aumentamos la distancia en 1.
Ahora la distancia es 2



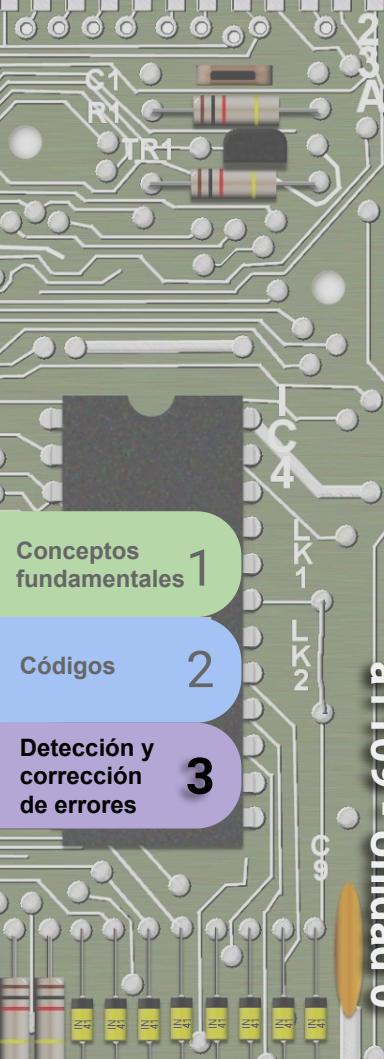
Detección y corrección de errores

...pero también podría ocurrir que al enviar 01001, un bit cambie en el canal de transmisión, debido a un error, aunque ahora **ninguna** de las combinaciones resultantes pertenecerán al código BCD 8421 con paridad. Es decir podemos detectar un error...

BCD 8421 con paridad					Paridad
A	B	C	D		
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0

ABC/D Paridad	00	01	11	10
000	0		1	
001		2		3
011	6		7	
010		4		5
110				
111				
101				
100		8		9

- Combinación válida
- Combinación invalida

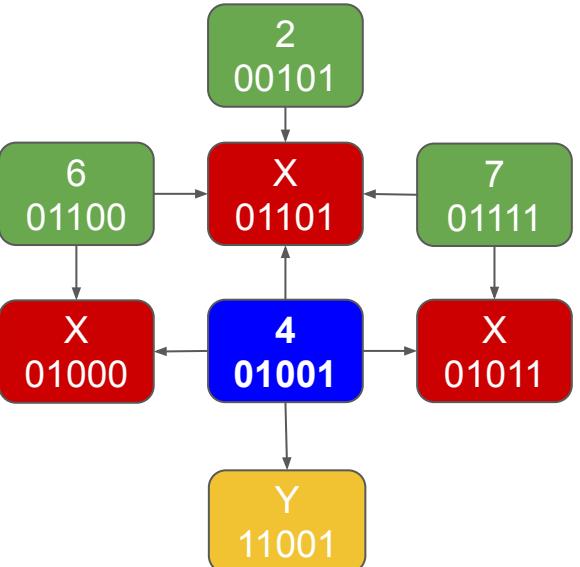


Detección y corrección de errores

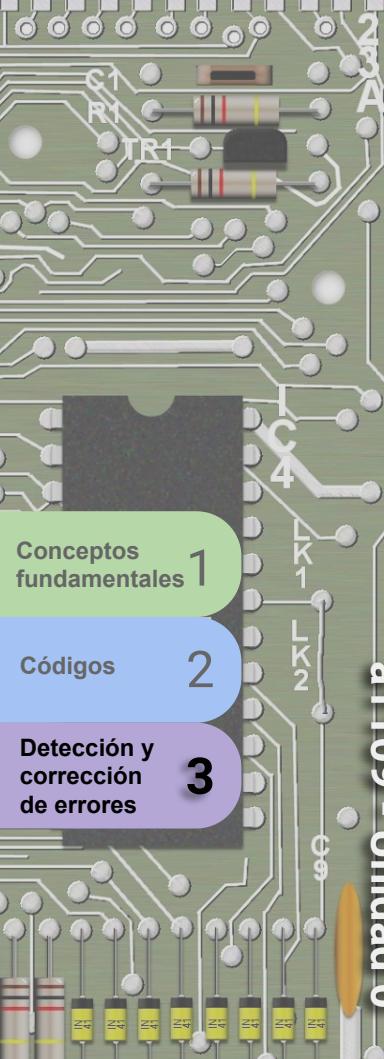
...pero no podemos corregirlo. Tomemos la palabra 01001 (4) y supongamos que en la transmisión un bit cambió. La palabra recibida es 01101, la cual es una combinación inválida. Podríamos suponer que si recibimos la palabra 01101 el valor original fue 01001 (4), pero esto **no es verdad**. Veamos, si tomamos la palabra 01100 (6) y se cambia un bit podemos llegar a 01101, **la misma combinación inválida que se produjo enviando 01001 (4) y cambiando un bit**. Por lo tanto podemos ver que en esta situación y con un código con una distancia de dos **podemos detectar un error** pero no podemos saber cuál fue el valor real enviado. La tabla muestra esta situación. Probablemente necesitemos aumentar aún más la distancia para poder detectar y corregir un error.

ABC/D Paridad	00	01	11	10
000	0		1	
001		2		3
011	6	2	7	
010	4	5		
110				
111				
101				
100	8		9	

■ Combinación válida
■ Combinación inválida



- Palabra válida enviada
- Palabra válida
- Palabra inválida modificada por ruido, múltiples orígenes
- Palabra inválida modificada por ruido, único origen (01001 en este caso)



Detección y corrección de errores

Afortunadamente existe una relación directa entre la cantidad de errores que se desean detectar y corregir y la distancia del código. Esta relación está dada por la siguiente inecuación.

Distancia mínima del código > Errores a detectar + Errores a corregir

(Dónde Errores a detectar \geq Errores a corregir)

Otra forma de representarlo

$$D > d+k$$

$$d \geq k$$

Por ejemplo si deseamos corregir un error, k debería ser 1 por lo tanto la desigualdad $d \geq k$ resulta de la siguiente forma:

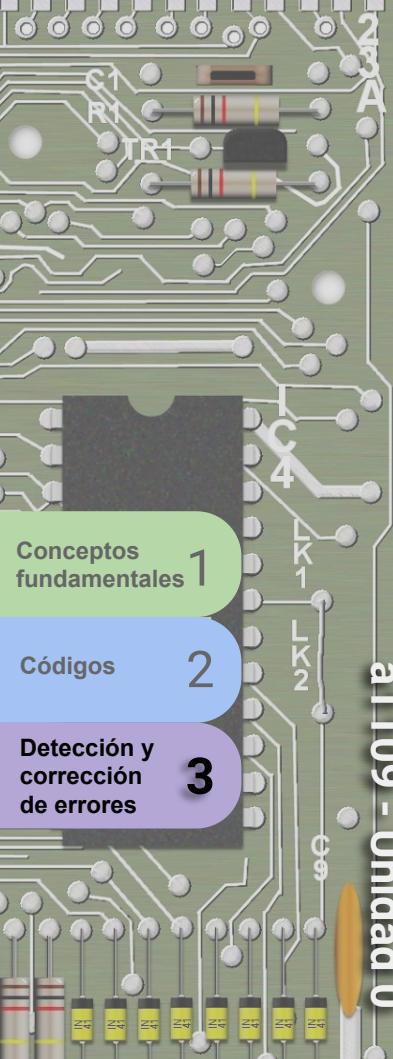
$$d \geq 1$$

Para que esta desigualdad se cumpla, d debería ser como mínimo 1. La inecuación $D > d+k$ ahora se puede solucionar

$$D > 1+1$$

$$D > 2$$

Para que se cumpla esta desigualdad, D debe ser como mínimo 3. Por lo tanto la distancia mínima que debe tener un código para corregir y detectar un error es **3**



Código de Hamming

Un método para ampliar **cualquier** código de distancia uno en distancia 3, y así poder detectar y corregir un error es el propuesto por Richard Hamming. Este método plantea formular ecuaciones parciales de paridad **par en los unos** siguiendo ciertos lineamientos.

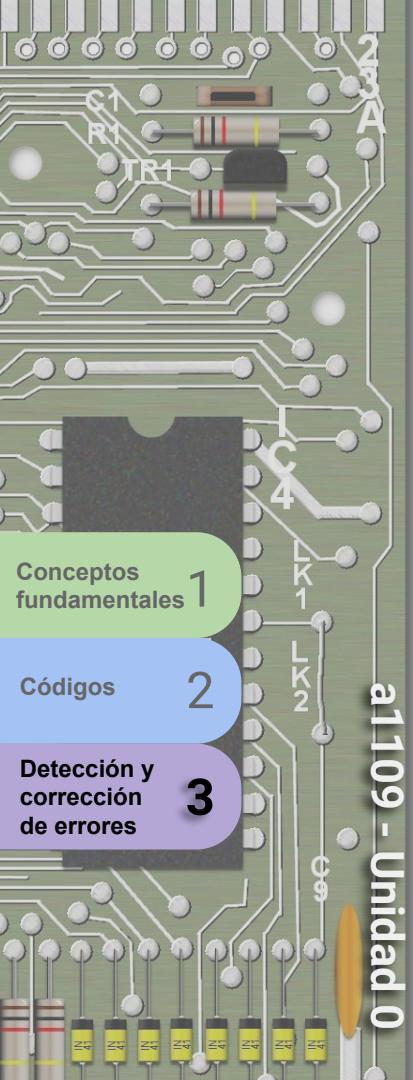
El primer lineamiento es definir cuántas ecuaciones se necesitarán y cuántos bits de paridad serán requeridos. Para este fin Hamming define la siguiente inecuación

cantidad de bits de datos + bits de paridad < 2^{bits de paridad}

Expresado de otra forma

$$d+p < 2^p$$

La cantidad de bits de datos + los bits de paridad serán la cantidad de bits en total a enviar



Conceptos
fundamentales 1

Códigos 2

Detección y
corrección
de errores 3

Código de Hamming

cantidad de bits a enviar + bits de paridad < 2^bits de paridad

Para solucionar esta inecuación de forma sencilla se debe proceder de la forma que se muestra en el siguiente ejemplo. Continuando con el ejemplo supongamos que deseamos enviar el dígito 4 en BCD 8421 cuya representación en binario es 0100 (4 bits), por lo tanto la inecuación quedaría conformada de la siguiente forma

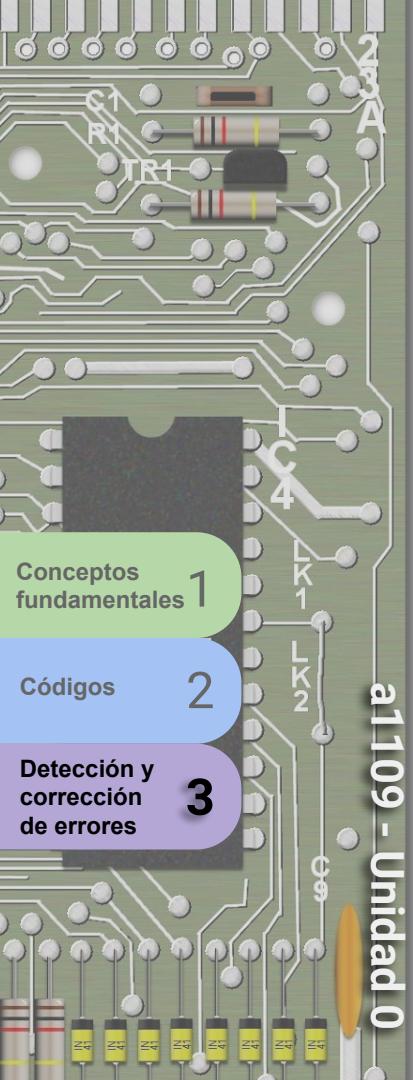
4 + bits de paridad < 2^bits de paridad

Solamente nos queda determinar los bits de paridad necesarios, aunque esta variable se encuentra a ambos lados de la inecuación. Por lo tanto probaremos con valores distintos hasta cumplir con la inecuación

$$4 + 1 < 2^1 \text{ (bits de paridad = 1)} \longrightarrow 5 < 2 \text{ NO CUMPLE}$$

$$4 + 2 < 2^2 \text{ (bits de paridad = 2)} \longrightarrow 6 < 4 \text{ NO CUMPLE}$$

$$4 + 3 < 2^3 \text{ (bits de paridad = 3)} \longrightarrow 7 < 8 \text{ CUMPLE}$$

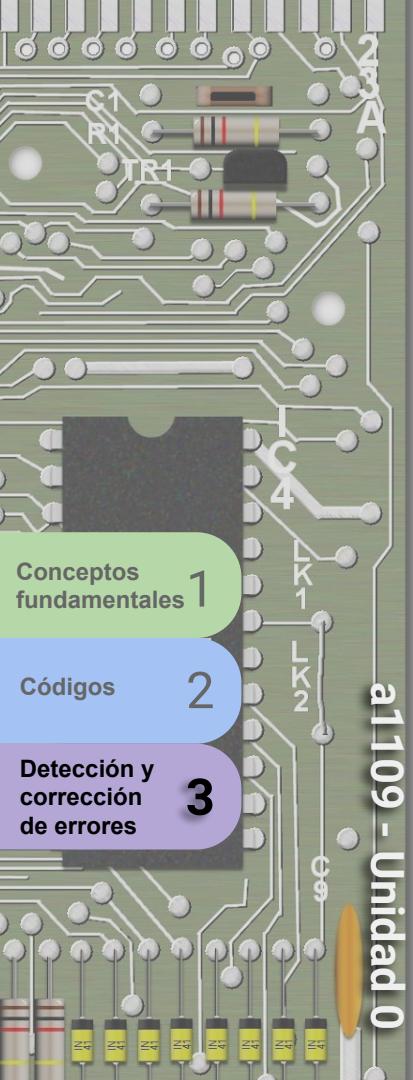


Código de Hamming

$$4 + 3 < 2^3 \text{ (bits de paridad} = 3) \longrightarrow 7 < 8 \text{ CUMPLE}$$

Por lo tanto se necesitarán enviar 7 bits ya que utilizaremos 3 bits de paridad. El segundo lineamiento es como confeccionar la trama a enviar (o la palabra del código de Hamming). Solamente nos restaría determinar cómo calcular las **paridades y en qué posición** se colocarán dentro de los 7 bits los bits de paridad. Una forma relativamente sistemática de solucionar estos problemas es la que se muestra en el siguiente ejemplo:

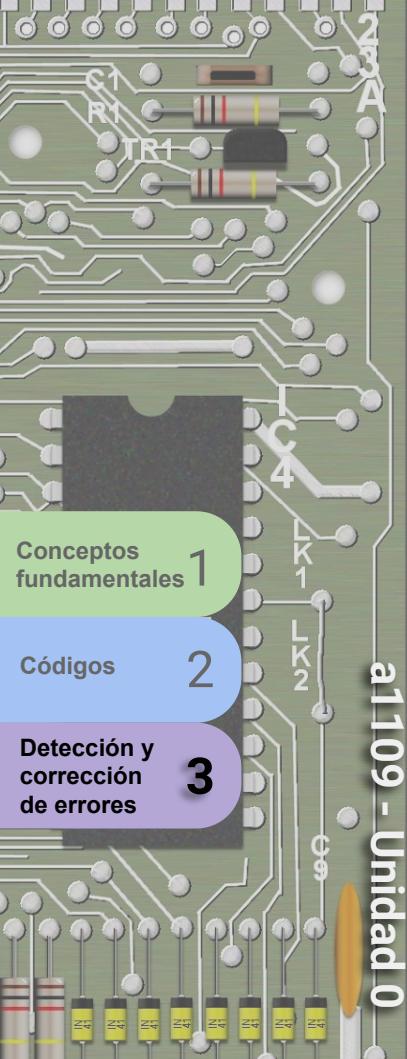
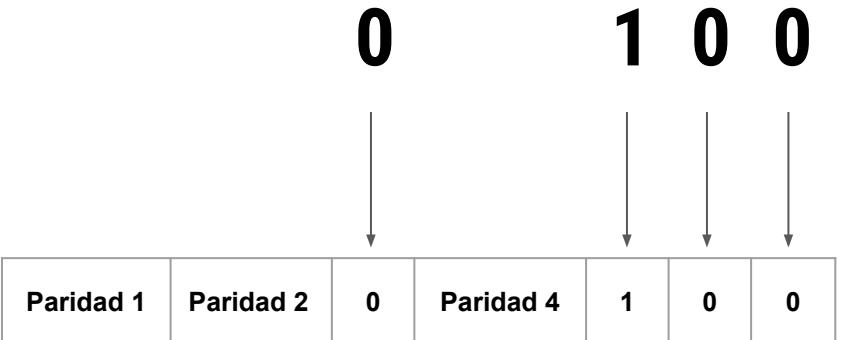
Recordemos que queríamos enviar el dígito 4 en BCD 8421 y que dicha representación en binario ocupa 4 bits . Junto con esta información se necesitaran enviar los bits de paridad. Utilizaremos entonces los bits cuya posición correspondan a una potencia de dos para colocar los bits de paridad, y los demás bits los utilizaremos para los bits que conforman el 4 en BCD 8421.



Código de Hamming

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6	Posición 7
$2^0=1$	$2^1=2$		$2^2=4$			

Las posiciones 1, 2, y 4 son las que utilizaremos para colocar los bits de paridad ya que dichas posiciones pueden ser representadas como potencias de dos. En las posiciones restantes (3,5,6 y 7) colocaremos la información del dígito BCD 8421

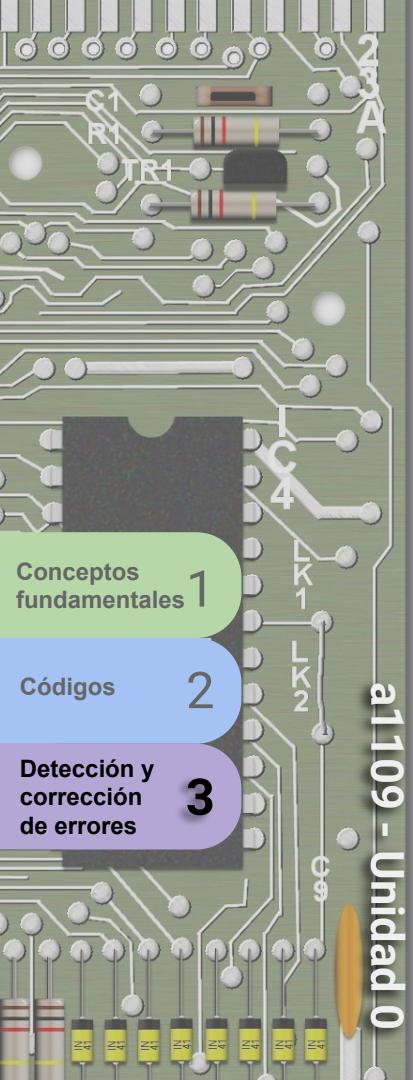


Código de Hamming

Lo único que nos resta es determinar cómo calcular los bits de paridad 1,2 y 4 para lo cual debemos crear ecuaciones independientes. Esto se podría realizar de forma arbitraria aunque aquí se propone una metodología sistemática que es la siguiente

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6	Posición 7
P1	P2	0	P4	1	0	0

Paridad	Posición 3	Posición 5	Posición 6	Posición 7	
P4= Paridad par (Posición 5	Posición 6	Posición 7)
P2=Paridad par (Posición 3		Posición 6	Posición 7)
P1=Paridad par (Posición 3	Posición 5		Posición 7)



Código de Hamming

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6	Posición 7
P1	P2	0	P4	1	0	0

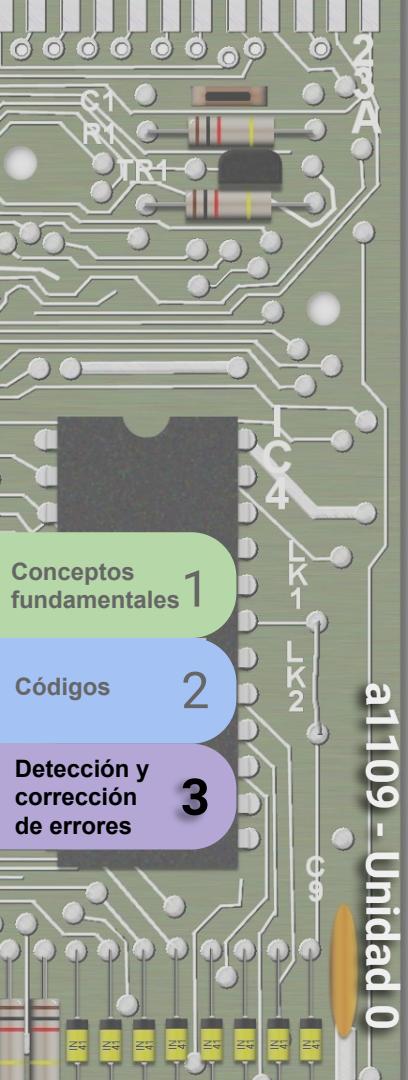
Paridad	Posición 3	Posición 5	Posición 6	Posición 7	
P4= Paridad par (1	0	0)
P2=Paridad par (0		0	0)
P1=Paridad par (0	1		0)

Calculamos la paridad par en los unos

$$\mathbf{P4} = \text{Paridad Par } (100) = 1$$

$$\mathbf{P2} = \text{Paridad Par } (000) = 0$$

$$\mathbf{P1} = \text{Paridad Par } (010) = 1$$



Código de Hamming

P4= Paridad Par (100) = 1

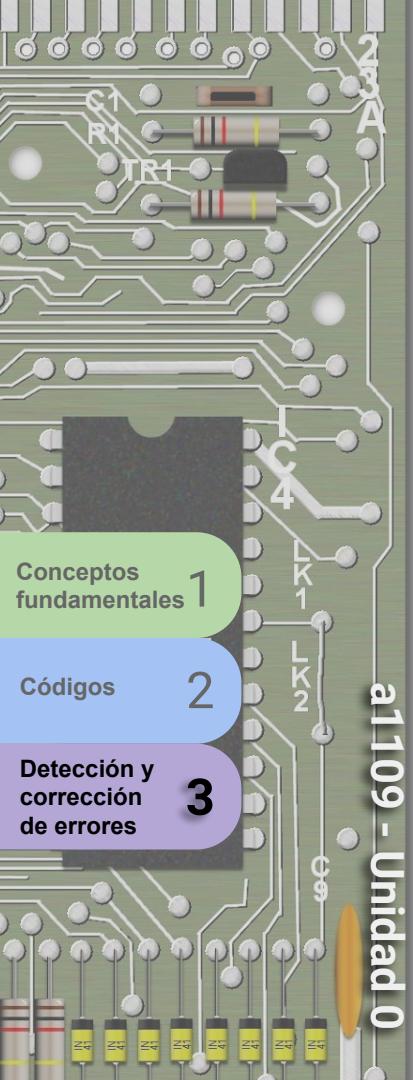
P2= Paridad Par (000) = 0

P1= Paridad Par (010) = 1

Ya con el cálculo de la paridad se puede finalmente confeccionar la palabra a enviar en la que se podrá detectar y corregir un error

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6	Posición 7
1	0	0	1	1	0	0

1001100 será la palabra correspondiente al 4 en BCD 8421 con distancia 3 (con 3 bits de paridad) a enviar en la cual vamos a poder detectar un error y corregir un error si ocurre.

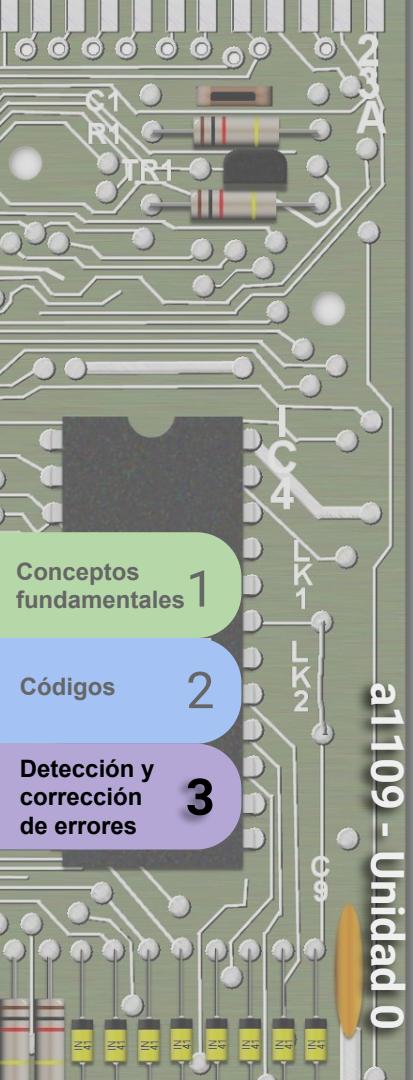


Código de Hamming

	P1	P2	A	P4	B	C	D
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1

Si calculamos Hamming como hicimos anteriormente para cada uno de los dígitos del código BCD 8421, obtenemos la tabla que se muestra a la izquierda. Este nuevo código BCD 8421 tendrá una distancia mínima de 3

Ya teniendo este nuevo código, analicemos gráficamente la distribución de combinaciones válidas e inválidas.

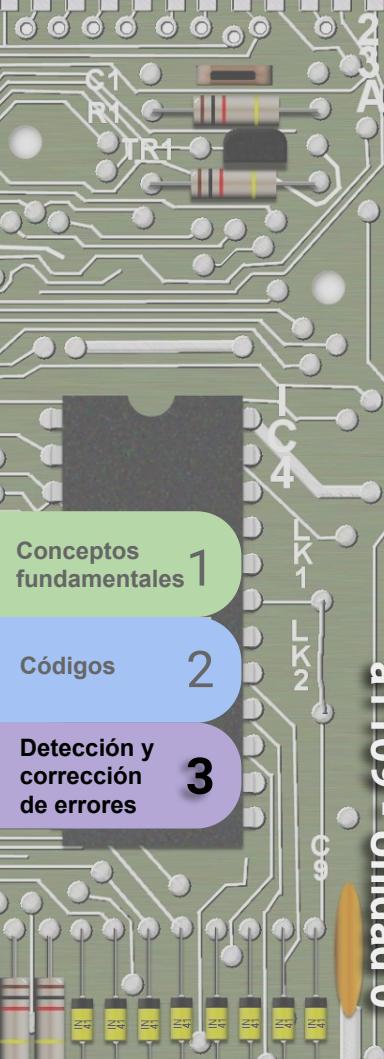


Código de Hamming

En primer lugar podemos observar que la distancia mínima del código es sin lugar a dudas 3 como se puede ver entre 6 y 5

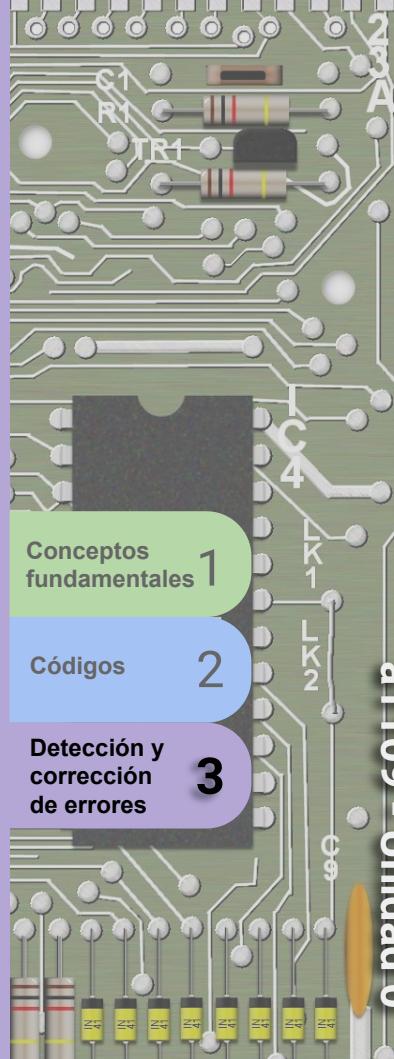
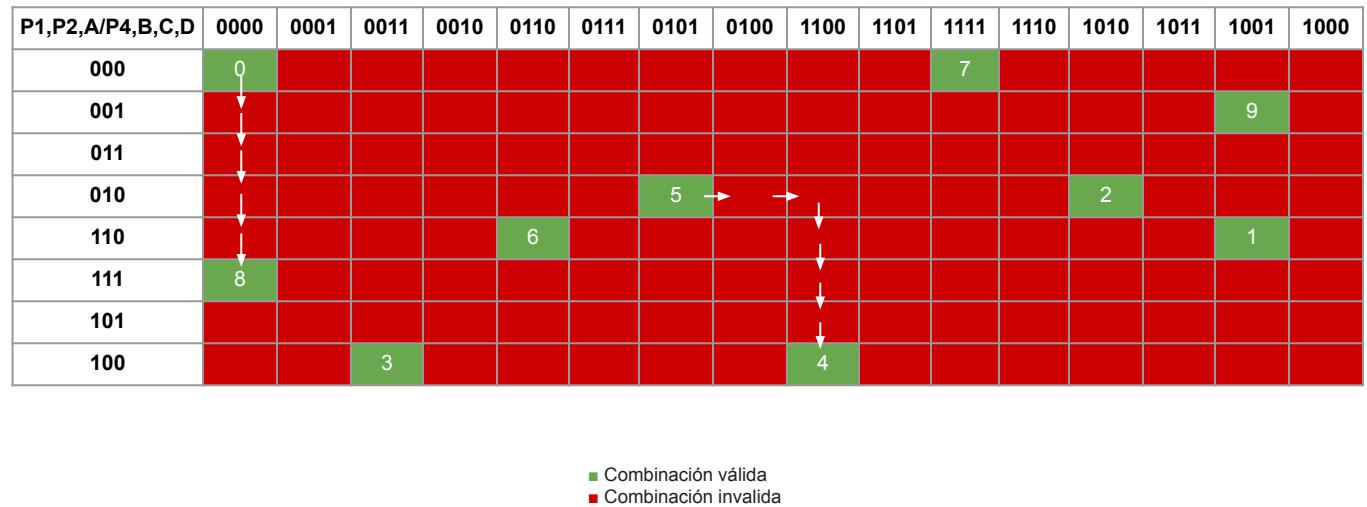
P1,P2,A/P4,B,C,D	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0										7					
001														9		
011																
010							5						2			
110				6	→	→								1		
111	8															
101																
100			3					4								

- Combinación válida
- Combinación inválida



Código de Hamming

Existirán otras distancias entre combinaciones válidas aún mayores como entre 0 y 8 con distancia 5 o como entre 5 y 4 con distancia 6

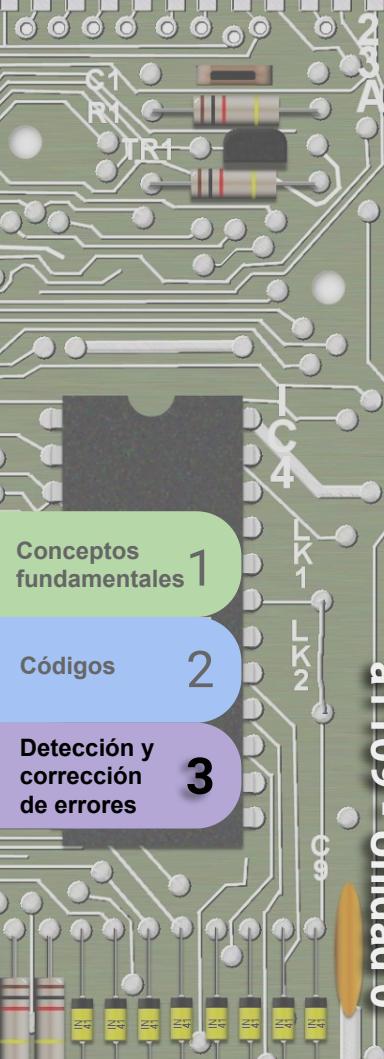


Código de Hamming

Por último y más importante podemos observar que si el número 4 en BCD 8421 con distancia 3 posee un error en uno de sus bits, cualquiera de las combinaciones resultantes (marcadas con x) sería un código inválido que solo se podría provocar si la palabra 1001100 fue enviada

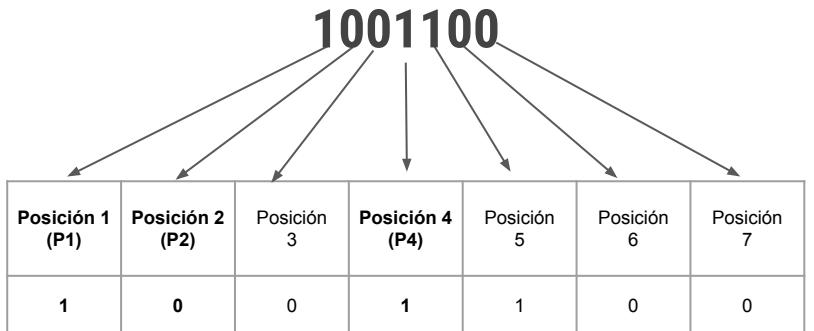
P1,P2,A/P4,B,C,D	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0								x		7					
001															9	
011																
010							5					2				
110					6								1			
111	8															
101								x								
100			3					x	4	x						

- Combinación válida
- Combinación inválida

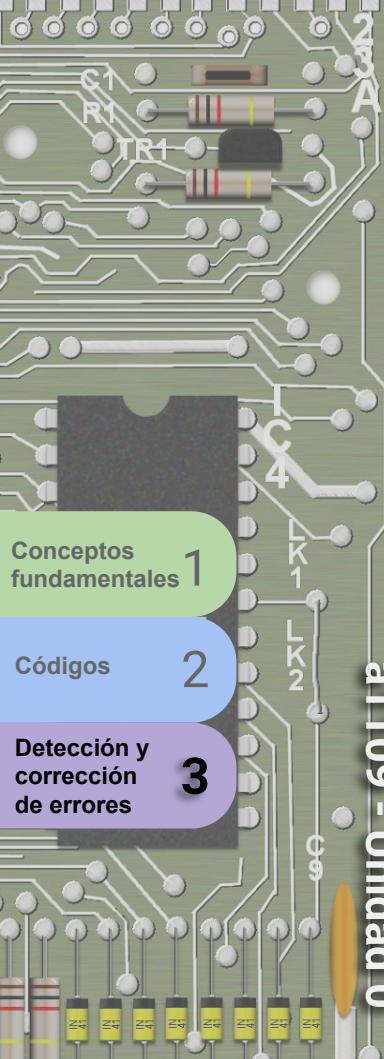


Código de Hamming

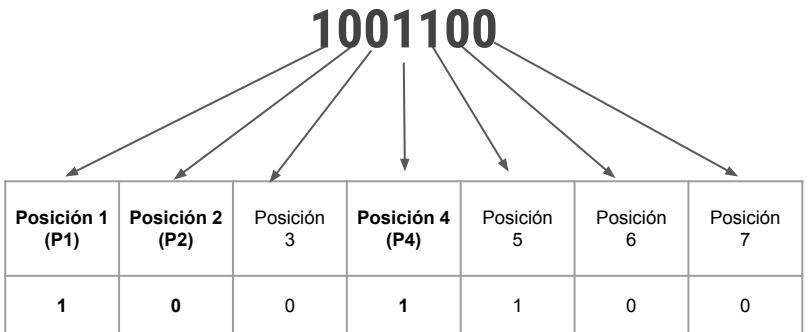
Para detectar errores lo único que se debe realizar es la verificación de la paridad, a partir de los datos recibidos. Se deben confeccionar las mismas ecuaciones para hallar la paridad que fueron explicadas anteriormente pero ahora teniendo en cuenta el bit de paridad. Si la palabra recibida es **1001100**, (4 en BCD 8421 después de aplicar Hamming) y calculamos la paridad par en los unos teniendo los bits de paridad en cuenta la paridad debería ser 0 en todos los casos



Ecuación	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		1	0	0)
E2=Paridad par(P2=0	0		0	0)
E1=Paridad par(P1=1	0	1		0)



Código de Hamming (Ningún error)



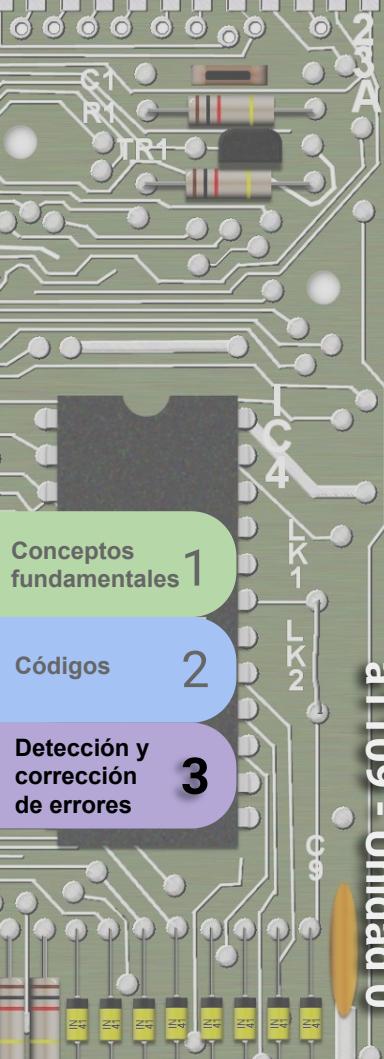
Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		1	0	0)
E2=Paridad par(P2=0	0		0	0)
E1=Paridad par(P1=1	0	1		0)

$$\mathbf{E4} = \text{Paridad Par (1100)} = 0$$

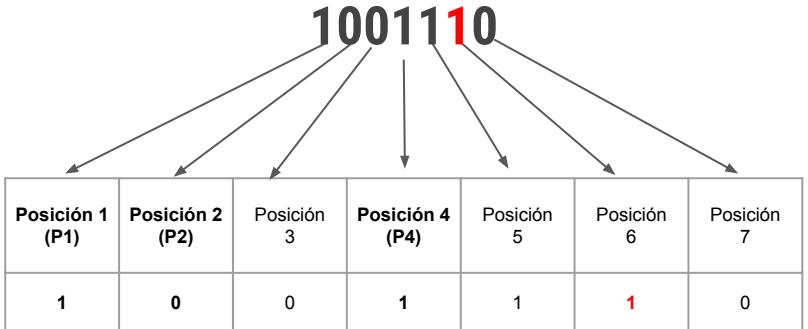
$$\mathbf{E2} = \text{Paridad Par (0000)} = 0$$

$$\mathbf{E1} = \text{Paridad Par (1010)} = 0$$

Calculamos la paridad par en los unos



Código de Hamming (Un error)



Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		1	1	0)
E2=Paridad par(P2=0	0		1	0)
E1=Paridad par(P1=1	0	1		0)

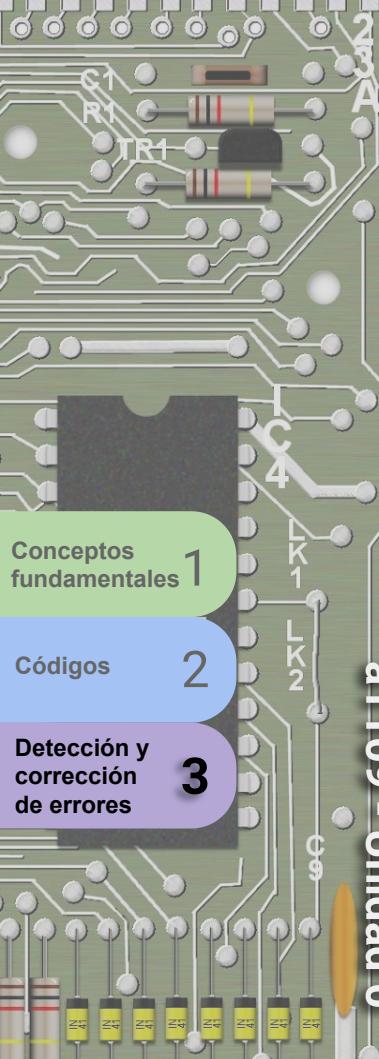
Calculamos la paridad par en los unos

$$\mathbf{E4 = Paridad Par (1110) = 1}$$

$$\mathbf{E2 = Paridad Par (0010) = 1}$$

$$\mathbf{E1 = Paridad Par (1010) = 0}$$

**ERROR EN
LA POSICIÓN**



Código de Hamming (Un error en bit de paridad)

1101100

Posición 1 (P1)	Posición 2 (P2)	Posición 3	Posición 4 (P4)	Posición 5	Posición 6	Posición 7
1	1	0	1	1	0	0

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		1	0	0)
E2=Paridad par(P2=1	0		0	0)
E1=Paridad par(P1=1	0	1		0)

$$\mathbf{E4 = Paridad Par (1100) = 0}$$

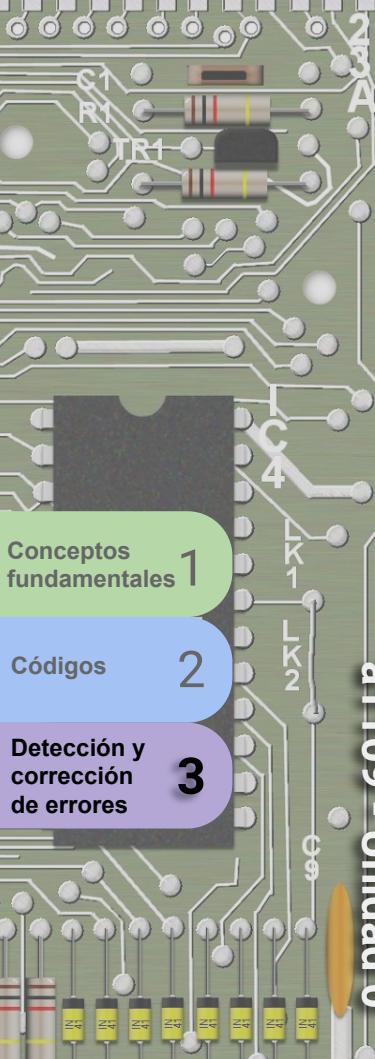
$$\mathbf{E2 = Paridad Par (1000) = 1}$$

$$\mathbf{E1 = Paridad Par (1010) = 0}$$

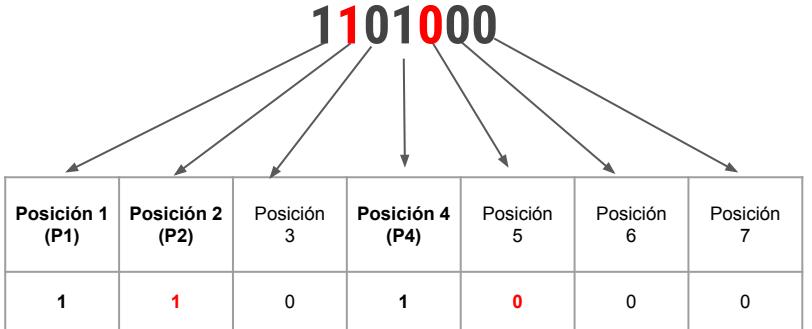
Calculamos la paridad par en los unos

**ERROR EN
LA POSICIÓN**

2



Código de Hamming (Dos errores)



Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		0	0	0)
E2=Paridad par(P2=1	0		0	0)
E1=Paridad par(P1=1	0	0		0)

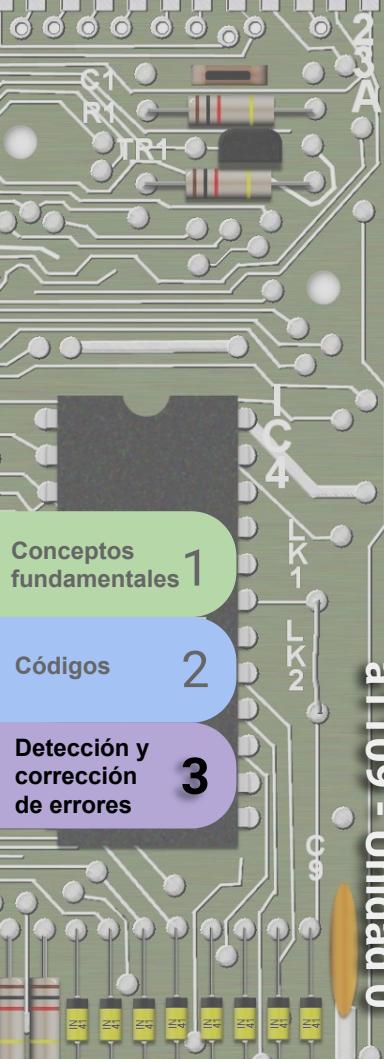
Calculamos la paridad par en los unos

$$\mathbf{E4 = Paridad Par (1000) = 1}$$

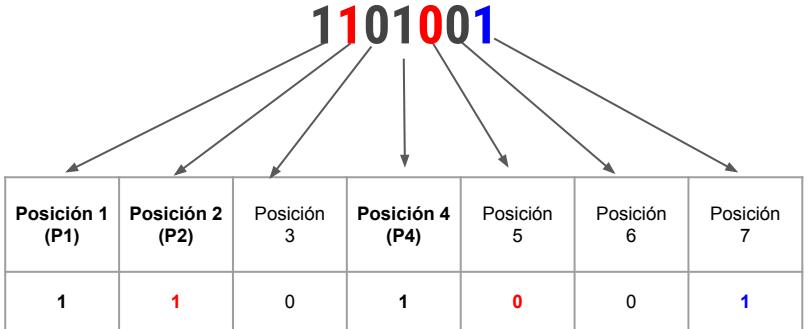
$$\mathbf{E2 = Paridad Par (1000) = 1}$$

$$\mathbf{E1 = Paridad Par (1000) = 1}$$

**ERROR EN
LA POSICIÓN**



Código de Hamming (Dos errores)



Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	
E4=Paridad par(P4=1		0	0	1)
E2=Paridad par(P2=1	0		0	1)
E1=Paridad par(P1=1	0	0		1)

Calculamos la paridad par en los unos

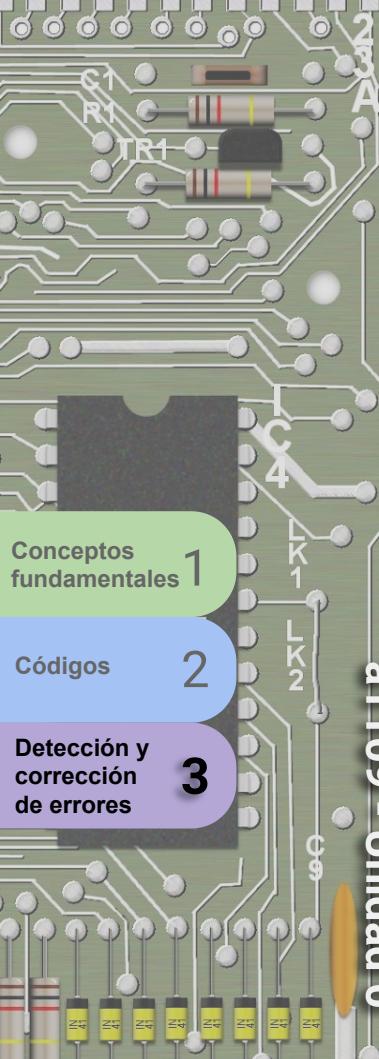
$$E4 = \text{Paridad Par } (1001) = 0$$

$$E2 = \text{Paridad Par } (1001) = 0$$

$$E1 = \text{Paridad Par } (1001) = 0$$

NO SE
ENCONTRARON
ERRORES

AUNQUE LOS HAY



Código de Hamming

En resumen al verificar si una palabra codificada con los criterios de Hamming para detectar y corregir un único error posee errores, podemos encontrarnos con tres diferentes situaciones:

- * Que todas las ecuaciones de paridad den por resultado 0 y por lo tanto ningún error fue detectado.
- * Que todas las ecuaciones de paridad NO den por resultado 0 y la posición en la cual se marca el error esté DENTRO de los bits recibidos. Esto quiere decir que se detectó un error (como vimos puede ser que exista más de uno)
- * Que todas las ecuaciones NO den por resultado 0 y la posición en la cual se marque el error esté POR FUERA de los bits recibidos. Es posible que exista más de un error.

P1 P2 D3 P4 D5 D6 D7 P8 D9 D10 D11 D12

P: Paridad
D: Datos

No se detectaron errores

$$E8 = \text{Paridad Par (P8 D9 D10 D11 D12)} = 0$$

$$E4 = \text{Paridad Par (P4 D5 D6 D7 D12)} = 0$$

$$E2 = \text{Paridad Par (P2 D3 D6 D7 D10 D11 D12)} = 0$$

$$E1 = \text{Paridad Par (P1 D3 D5 D7 D9 D11 D12)} = 0$$

Error en la posición N

$$E8 = \text{Paridad Par (P8 D9 D10 D11 D12)} = 0$$

$$E4 = \text{Paridad Par (P4 D5 D6 D7 D12)} = 1$$

$$E2 = \text{Paridad Par (P2 D3 D6 D7 D10 D11 D12)} = 1$$

$$E1 = \text{Paridad Par (P1 D3 D5 D7 D9 D11 D12)} = 1$$

Possiblemente más de un error

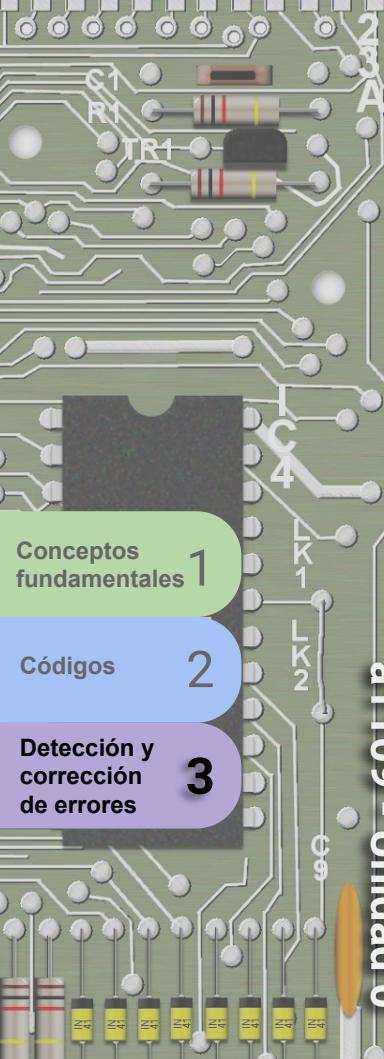
$$E8 = \text{Paridad Par (P8 D9 D10 D11 D12)} = 1$$

$$E4 = \text{Paridad Par (P4 D5 D6 D7 D12)} = 1$$

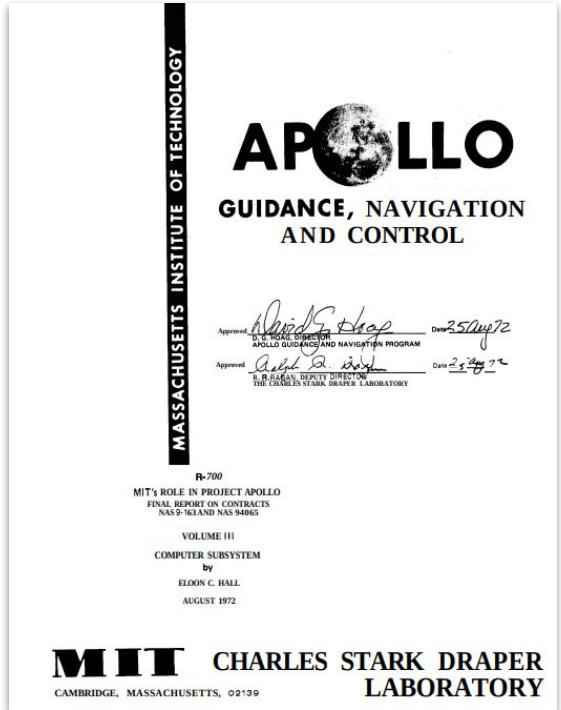
$$E2 = \text{Paridad Par (P2 D3 D6 D7 D10 D11 D12)} = 1$$

$$E1 = \text{Paridad Par (P1 D3 D5 D7 D9 D11 D12)} = 0$$

(La posición del error está por fuera de la cantidad de bits recibidos)

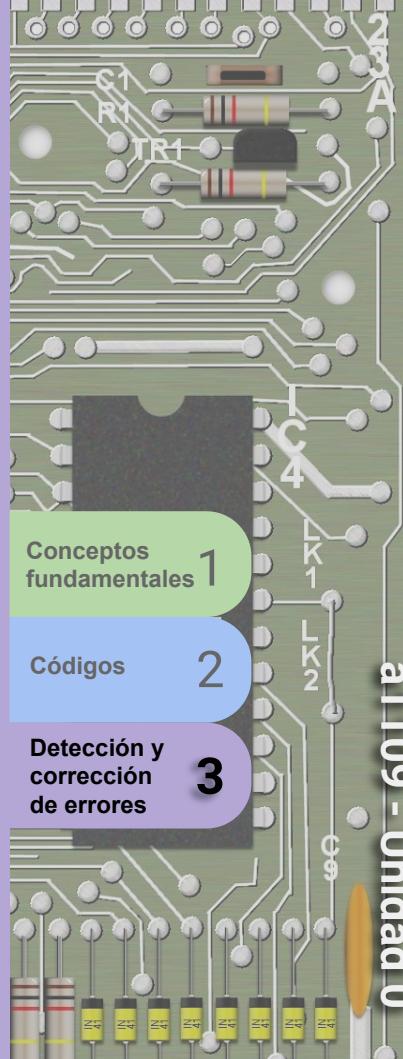


Código de Hamming



*"All tape records utilize both lateral (vertical) and longitudinal parity along with a two bit sequence code included in each byte of the tape. Three simplex copies of a program are written on the tape to provide a means of error recovery. This format was adopted because errors due to tape imperfections could cause multiple errors rather than a single bit error. The alternatives, such as multiple error correction (**such as Hamming codes**), did not appear to justify the extra logic or effective loss in packing density."*

Todas las cintas utilizan paridad lateral (vertical) y longitudinal junto con un código de secuencia de dos bits incluido en cada byte de la cinta. Para proporcionar un medio de recuperación de errores se utilizan tres copias de un programa. Este formato se adoptó porque los errores debido a las imperfecciones en las cintas podrían causar múltiples errores en lugar de un solo error de bit. Las alternativas, como la corrección de errores múltiples (**como los códigos de Hamming**), no parecen justificar la lógica adicional o la pérdida efectiva de la densidad del empaquetado.



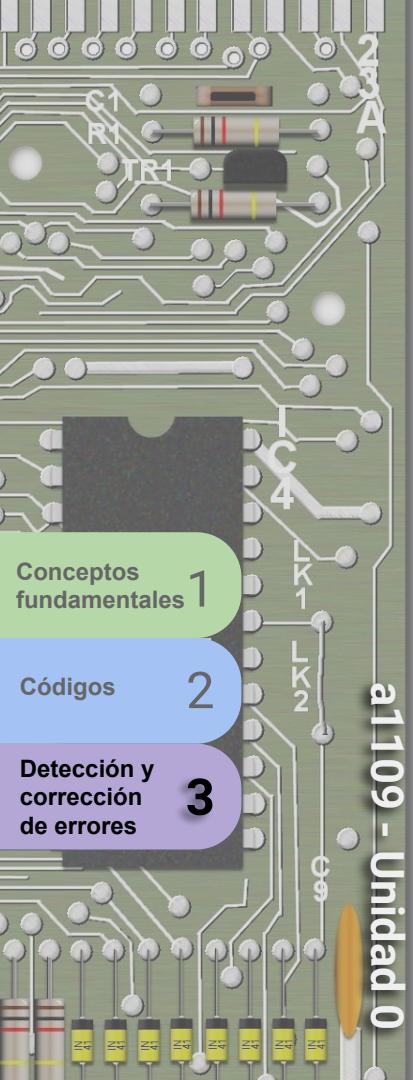
Código de Hamming

1-Obtenga un código de Hamming de los siguientes números 1791,1815 y 1923 codificando los mismos en BCD 8421 empaquetado

2- En una comunicación se recibieron las siguientes cadenas de bits de las cuales se saben que son códigos de Hamming y que contienen códigos ASCII extendido. Determine si algunas de las cadenas posee un error y si existe alguno corrijalo.

- 0 0 0 0 0 1 1 0 0 1 1 0
- 1 1 1 0 1 0 0 0 0 1 0 1
- 0 1 0 1 1 0 0 1 0 0 0 1
- 0 0 0 0 1 0 1 0 0 1 0 1
- 1 1 0 1 1 0 0 0 1 0 1 0
- 1 1 0 1 0 1 1 1 0 1 0 1

3-¿Cuantos bit se necesitarán enviar en total si se desea transmitir una palabra de 16 bits en un código de Hamming? ¿Cuantos bits de paridad serán necesarios?



Conceptos
fundamentales 1

Códigos 2

Detección y
corrección
de errores 3