Arquitectura de Computadoras

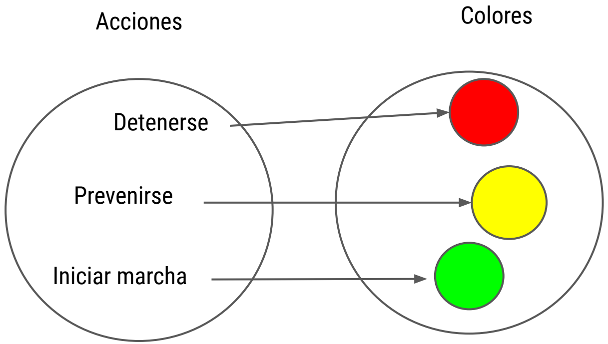
Unidad 0
Códigos

Martín Ferreyra Biron

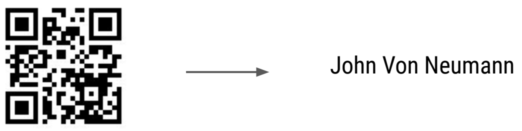
Códigos: conceptos fundamentales

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.



Relación entre las acciones y los colores de un semáforo



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

ξ	1010
η	0011
γ	1011
θ	0101

Palabras 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

31416

1010 1101 0111 1101 1100

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

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

Representación del número 31416 en binario puro

31416 → **0111 1010 1011 1000**

↓

0111 1010 1011 1000

≠

1010 1101 0111 1101 1100 → **1010 1101 0111 1101 1100**

Representación del número 31416 en un código BCD arbitrario

Ambas representaciones no tienen por qué tener algún tipo de relación

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

Convirtamos 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

$$\underbrace{0010}_{2}\underbrace{1001}_{9}\ \underbrace{0001}_{1}\underbrace{0111}_{7}$$

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 para definirlos 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 que además de ser progresivo su primera y última palabra también 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 digitos del 0-9, necesitaremos 5 bits (2*5=10)

Código Johnson

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

Para crear el código Johnson de 5 bits , comencemos 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. Por lo tanto la para generar la palabra correspondiente al 2 se le debe *inyectar* un uno desde la derecha a la palabra anterior 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 están 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. 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 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 conformará el 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.

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 obtendriamos un código BCD que cumpliria con todas las caracterisitcas de códigos ya vistas. 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 combinación y sus simétricas, desde el eje de simetría o desde los extremos. 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 código, 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

C_{B-1} (0)=9
&
C_{B-1} (0000)=1111

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

C_{B-1} (1)=8
&
C_{B-1} (0001)=1110

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 todos sus palabras , se puede decir que el código es **autocomplementado**.

Códigos autocomplementados- Código BCD Exceso 3 (XS 3)

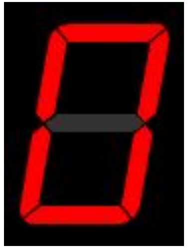
DECIMAL				
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

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 representar como 0011.

¿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

		1 1 1			
2961	=	0010	1001	0110	0001
9989	=	1001	1001	1000	1001
		1100	**0010**	**1110**	**1010**
		12	**18**	**14**	**10**

Suma en BCD 8421

Realice la siguiente suma en BCD 8421 2961+9989

		1 1 1			
2961	=	0010	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

Suma en BCD 8421

Realice la siguiente suma en BCD 8421 2961+9989

		1 1 **1**			
2961 =		0010	1001	0110	0001
9989 =		1001	1001	1000	1001
			1	1	
	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
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

0011 → 0 (En XS 3)

\+ 0011 → 0 (En 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	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

$$
\begin{array}{r l}
{}^{1\,1} & \\
0011 & \longrightarrow 0 \text{ (En XS 3)} \\
+\ 0011 & \longrightarrow 0 \text{ (En XS 3)} \\
\hline
0110 & \longrightarrow \mathbf{3} \textbf{ (En XS 3)}
\end{array}
$$

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

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	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

  1 1
0011 → 0 (En XS 3)
\+
0011 → 0 (En XS 3)
____
0110 → **3 (En XS 3)**
−
0011
____
0011 → **0 (En 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
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

1
0100 → 1 (En XS 3)
\+
1100 → 9 (En XS 3)

0001 0000

Suma en BCD XS 3

Al obtener el resultado de la suma 1+9 en XS 3 podemos observar que el resultado no se encuentra en XS 3

DECIMAL				
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

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

0100 → 1 (En XS 3)

1100 → 9 (En XS 3)

0001 0000 → **NO CORRESPONDE A XS 3**

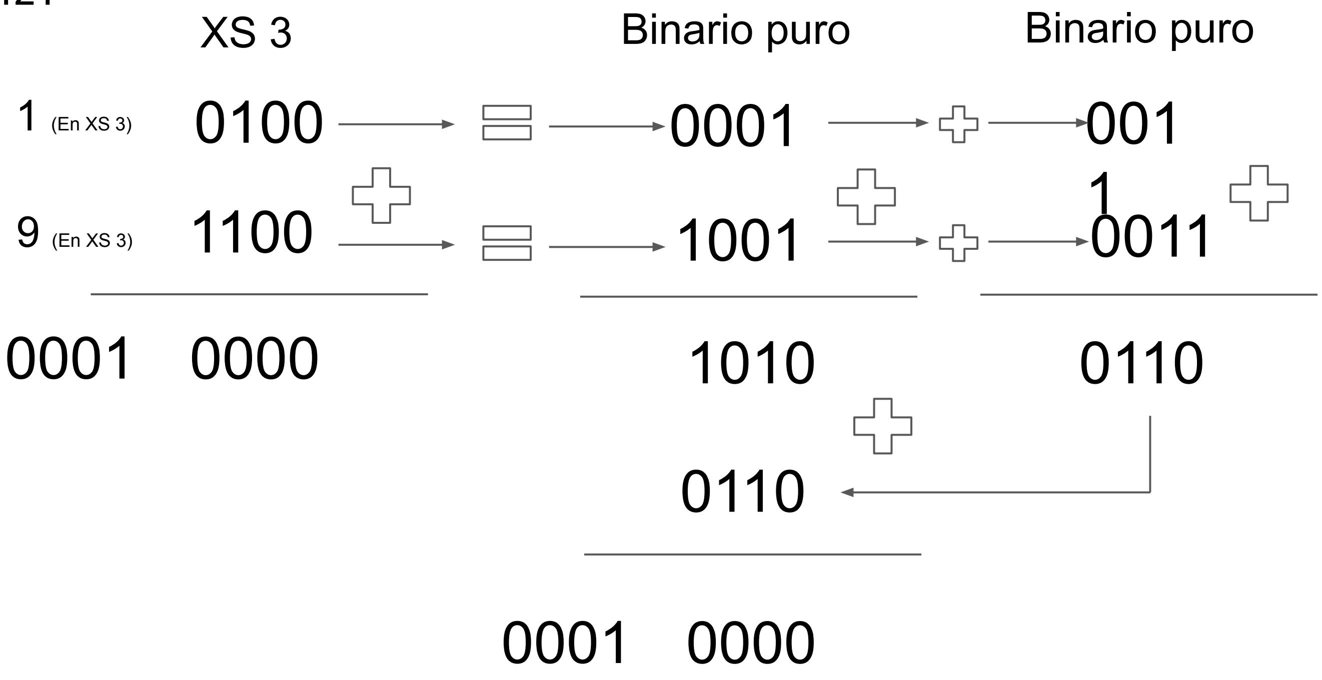
Suma en BCD XS 3

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

	XS 3		Binario puro		Binario puro
1 (En XS 3)	0100	=	0001	+	0011
9 (En XS 3)	+ 1100	=	+ 1001	+	+ 0011
	0001 0000		1010		0110
			+ 0110		
			0001 0000		

Suma en BCD XS 3

Observemos que al sumar 1+9 en binario puro obtenemos 1010 (10 en binario puro). Al sumarle 0110 (la suma de los excesos de 1 y 9 en XS 3) 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 en binario puro como si fuera BCD 8421, debido a que al sumar dos veces el exceso de 3 unidades, estamos sumando seis , como si se estuviera corrigiendo en BCD 8421 cuando el resultado de una suma es mayor a 9, como se vio en diapositivas anteriores.

Corrección de la suma 1+9 en BCD 8421

0 0001

\+

0 1001

1

.0 **1010**

\+

0 110

)1 0000

Suma de 1+9 en XS 3 descompuesta

XS 3		Binario puro		Binario puro
0100	=	0001	→ + →	001
		+		1 +
1100	=	1001	→ + →	0011
		1010		0110
		+ 0110	←	
		0001 0000		

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 para transformar esos numero a XS 3 y así corregir el resultado.

DECIMAL				
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

```
     1
     0100        →  1 (En XS 3)
  +
     1100        →  9 (En XS 3)
----------------
0001 0000        →  NO CORRESPONDE A XS 3
  +
0011 0011        →  Corrección
----------------
0100 0011

   1         0
(En XS 3) (En XS 3)
```

NO CORRESPONDE A XS 3

Corrección

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 misma son:

- Si el resultado de la suma de dos dígitos en BCD XS 3 es menor o igual a nueve (no hubo carry), entonces se le debe restar 3 al resultado obtenido

- Si el resultado de la suma de dos dígitos en BCD XS 3 es mayor a 9 (hubo 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	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

	1		1	1 1 1	1 11	
52389 =		1000	0101	0110	1011	1100
61720=		1001	0100	1010	0101	0011
	0001	0001	1010	0001	0000	1111

+

Suma en BCD XS 3

Ya habiendo sumado todos los números procederemos a corregir los resultados sumando o restando según hemos explicado. Marcamos en rojo los números que no se encuentran dentro del código BCD XS 3 en el resultado

DECIMAL				
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

	1		1	1 1 1	1 11	
52389 =		1000	0101	0110	1011	1100
61720=		1001	0100	1010	0101	0011
	0001	0001	1010	0001	0000	1111
	+0011	+0011	-0011	+0011	+0011	-0011

Suma en BCD XS 3

Sumamos y obtenemos 114109 en XS 3 que es el resultado esperado

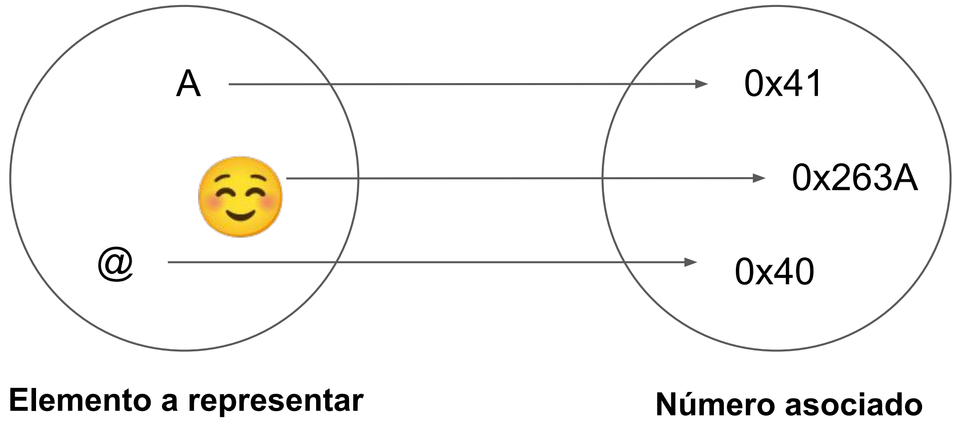
DECIMAL				
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Código BCD XS 3

	1		1	1 1 1	1 11	
52389 =		1000	0101	0110	1011	1100
61720=		1001	0100	1010	0101	0011
	0001	0001	1010	0001	0000	1111
	+0011	+0011	-0011	+0011	+0011	-0011
	0100	0100	0111	0100	0011	1100
	1	1	4	1	0	9

Códigos alfanuméricos

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 alfanuméricos- 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	@	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

Códigos alfanuméricos- ASCII extendido

Dec	Hex	Oct	Car	Dec	Hex	Oct	Car	Dec	Hex	Oct	Car	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	â	163	A3	243	ú	195	C3	303	+	227	E3	343	Ò
132	84	204	ä	164	A4	244	ñ	196	C4	304	-	228	E4	344	õ
133	85	205	à	165	A5	245	Ñ	197	C5	305	+	229	E5	345	Õ
134	86	206	å	166	A6	246	ª	198	C6	306	ã	230	E6	346	µ
135	87	207	ç	167	A7	247	º	199	C7	307	Ã	231	E7	347	þ
136	88	210	ê	168	A8	250	¿	200	C8	310	+	232	E8	350	Þ
137	89	211	ë	169	A9	251	®	201	C9	311	+	233	E9	351	Ú
138	8A	212	è	170	AA	252	¬	202	CA	312	-	234	EA	352	Û
139	8B	213	ï	171	AB	253	½	203	CB	313	-	235	EB	353	Ù
140	8C	214	î	172	AC	254	¼	204	CC	314	¦	236	EC	354	ý
141	8D	215	ì	173	AD	255	¡	205	CD	315	-	237	ED	355	Ý
142	8E	216	Ä	174	AE	256	«	206	CE	316	+	238	EE	356	¯
143	8F	217	Å	175	AF	257	»	207	CF	317	¤	239	EF	357	´
141	90	220	ì	176	B0	260	_	208	D0	320	ð	240	F0	360	-
145	91	221	æ	177	B1	261	_	209	D1	321	Ð	241	F1	361	±
146	92	222	Æ	178	B2	262	_	210	D2	322	Ê	242	F2	362	_
147	93	223	ô	179	B3	263	¦	211	D3	323	Ë	243	F3	363	¾
148	94	224	ö	180	B4	264	¦	212	D4	324	È	244	F4	364	¶
149	95	225	ò	181	B5	265	Á	213	D5	325	i	245	F5	365	§
150	96	226	û	182	B6	266	Â	214	D6	326	Í	246	F6	366	÷
151	97	227	ù	183	B7	267	À	215	D7	327	Î	247	F7	367	¸
152	98	230	ÿ	184	B8	270	©	216	D8	330	Ï	248	F8	370	°
153	99	231	Ö	185	B9	271	¦	217	D9	331	+	249	F9	371	¨
154	9A	232	Ü	186	BA	272	¦	218	DA	332	+	250	FA	372	·
155	9B	233	ø	187	BB	273	+	219	DB	333	_	251	FB	373	¹
156	9C	234	£	188	BC	274	+	220	DC	334	_	252	FC	374	³
157	9D	235	Ø	189	BD	275	¢	221	DD	335	¦	253	FD	375	²
158	9E	236	×	190	BE	276	¥	222	DE	336	Ì	254	FE	376	_
159	9F	237	ƒ	191	BF	277	+	223	DF	337	_	255	FF	377	(Espacio)

Códigos alfanuméricos

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.

0x0000263A

La cara sonriente corresponde a 0X0000263A en UNICODE

Códigos alfanuméricos

00000000 00000000 00000000 01000001

A Unicode

00000000 00000000 00000000 11010001

Ñ Unicode

00000000 00000000 00100110 00111010

☺️ Unicode

Caracteres de tamaño variable

Sería ideal utilizar la cantidad mínima posible de bits para representar cada carácter. Si un texto posee el 99% de caracteres que pueden representarse en ASCII 7 bits y solo un 1% de los caracteres requieren más de 7 bits, al forzar todos los caracteres a UNICODE de 32 bits, estaríamos agregando 3 bytes extras en 0 para el 99% del documento.

Para solucionar este problema, se crea la codificación **UTF (Unicode Transformation Format)** . En el caso de **UTF-8**, cada carácter ocupa una cantidad variable de bytes. Si el carácter puede ser representado con 7 bits, entonces solo ocupará un byte. Si el carácter puede ser representado por 11 bits, entonces ocupará 2 bytes, y así sucesivamente.

En el caso del UNICODE la letra 'A' (sin representar los ceros a la izquierda) equivale a 1000001 (7 bits) por lo tanto la 'A' se representa en UTF-8 como **0**1000001. Cualquier carácter que se represente con 7 bits en UNICODE ocupará un solo byte en UTF-8 y su bit más significativo será siempre **0**.

Caracteres de tamaño variable

En el caso que el carácter requiera más de 7 bits, por ejemplo Ñ 11010001 (8 bits), se utilizan 2 bytes. El byte más significativo comienza con 110 y continua con 5 bits (hhhhh), y el byte menos significativo comienza con 10 y continua con 6 bits (llllll). Por ende el formato sería **110**hhhhh **10**llllll. Luego hhhhhllllll son 11 bits que pueden representar códigos unicode desde el 80_{16} hasta el $7FF_{16}$. Vemos que Ñ 11010001 equivale a $D1_{16}$ con lo cual es mayor a 80_{16} y menor a $7FF_{16}$. Su representación en UTF-8 equivale a **110**00011 **10**010001.

En el caso del emoji cuyo código UNICODE es 00100110 00111010, el mismo requiere 14 bits para representarse. La codificación UTF-8 define entonces 3 bytes, el primero comienza con **1110** hhhh , el segundo con **10**mmmmmm y el tercero **10**llllll. De esta forma se codifican hhhhmmmm mmllllll (16 bits UNICODE) usando 3 bytes UTF-8. Luego el UNICODE 00100110 00111010 se codifica en UTF-8 como **1110**0010 **10**011000 **10**111010.

En el caso de requerir más de 16 bits, UTF-8 define 4 bytes donde el primero comienza con 11110, pero sigue la misma regla general para el resto.

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

1001

Distancia 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**

0000
0001

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

Paridad

Análogamente un código está definido con paridad **impar** en los unos si se configuran todas las combinaciones del mismo con un número impar 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 impar (1 uno) →	0 →	0010 →	IMPAR
011 →	Paridad de unos par (2 unos) →	1 →	0111 →	IMPAR
111 →	Paridad de unos impar (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

Emisor

1100100000011

Error

1100100001011

Canal

Receptor

1100100001011

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

Observemos el siguiente inconveniente: 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**

0101

El número 5 codificado en BCD 8421 podría llegar correctamente

0100
0111
0001

Combinaciones válidas en BCD 8421

1101

Combinación invalida en BCD 8421

Pero también podrían ocurrir un error y llegar al receptor cualquiera de estas combinaciones

Existen combinaciones que incluso con un error siguen formando parte del código y pueden ser entendidas como válidas

Detección y corrección de errores

Esto nos quiere decir algo. 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.

0100
0111
0001

Combinaciones válidas en BCD 8421

1101

Combinación invalida en BCD 8421

$$1101 \longrightarrow 0101 \rightarrow \{0100,\ 0111,\ 0001,\ 1101\}$$

Pero la distancia del código BCD 8421 es también de 1, y es por ese motivo que no podemos discernir cuando una palabra recibida contiene un error o no con total certeza (menos en los casos donde la palabra esté fuera del código como el caso de 1101)

Detección y corrección de errores

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

Imaginemos ahora que se desea enviar una palabra de un código con distancia dos. Se sabe que el canal puede producir como máximo un error. 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

Aumentamos la distancia en 1. Ahora la distancia es 2

01010

El número 5 codificado en BCD 8421 podría llegar correctamente

01011
01000
01110
00010
11010

Pero también puede ocurrir que un bit cambie, debido a un error

Pero **ninguna** de estas combinaciones pertenecen al código BCD 8421 con paridad. Ahora sí podemos detectar un error

Detección y corrección de errores

Lo que se explicó hasta ahora es solamente una forma de detectar errores. Sin embargo se puede encontrar una relación directa entre la cantidad de errores que se desean detectar, 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 ⩾ Errores a corregir)

Otra forma de representarlo

$$D>d+k$$

$$d\geqslant k$$

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

$$d\geqslant 1$$

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 1 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

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. Supongamos que deseamos enviar la letra 'U' en código ASCII cuya representación en binario es 01010101 (8 bits), por lo tanto la inecuación quedaría conformada de la siguiente forma

8 + 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

8 + 1 < 2^1 (bits de paridad = 1) → **9 < 2 NO CUMPLE**

8 + 2 < 2^2 (bits de paridad = 2) → **10 < 4 NO CUMPLE**

8 + 3 < 2^3 (bits de paridad = 3) → **11 < 8 NO CUMPLE**

8 + 4 < 2^4 (bits de paridad = 4) → **12 < 16 CUMPLE**

Código de Hamming

8 + 4 < 2^4 (bits de paridad = 4) → **12 < 16 CUMPLE**

Por lo tanto se necesitarán enviar 12 bits ya que utilizaremos 4 bits de paridad. Continuando con el ejemplo, el segundo lineamiento es como confeccionar la trama a enviar (o la palabra del código de Hamming). Solamente nos restaría determinar en qué **posición** se colocarán dentro de los 12 bits los bits de paridad y como calcular las **paridades**. Una forma relativamente sistemática de solucionar el primer problema es la que se observa en el siguiente ejemplo.

Recordemos que queríamos enviar la letra ‘U’ codificada en ASCII (01010101) y que dicha representación en binario ocupa 8 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 ASCII de la letra ‘U’.

Código de Hamming

Posición 1	**Posición 2**	Posición 3	**Posición 4**	Posición 5	Posición 6	Posición 7	**Posición 8**	Posición 9	Posición 10	Posición 11	Posición 12
$2^0=1$	$2^1=2$		$2^2=4$				$2^3=8$				

Las posiciones 1, 2, 4 y 8 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 (3,5,6,7,9,10,11 y 12) colocaremos la información de la letra 'U'

0 1 0 1 0 1 0 1

Paridad 1	**Paridad 2**	**0**	**Paridad 4**	**1**	**0**	**1**	**Paridad 8**	**0**	**1**	**0**	**1**

Código de Hamming

Lo único que nos resta es determinar cómo calcular los bits de paridad 1,2,4 y 8, para esto 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	**Posición 8**	Posición 9	Posición 10	Posición 11	Posición 12
P1	**P2**	0	**P4**	1	0	1	**P8**	0	1	0	1

P8=					Posición 9	Posición 10	Posición 11	Posición 12
P4=		Posición 5	Posición 6	Posición 7				Posición 12
P2=	Posición 3		Posición 6	Posición 7		Posición 10	Posición 11	
P1=	Posición 3	Posición 5		Posición 7	Posición 9		Posición 11	

Código de Hamming

Posición 1	**Posición 2**	Posición 3	**Posición 4**	Posición 5	Posición 6	Posición 7	**Posición 8**	Posición 9	Posición 10	Posición 11	Posición 12
P1	**P2**	0	**P4**	1	0	1	**P8**	0	1	0	1

Paridad	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
P8=					0	1	0	1
P4=		1	0	1				1
P2=	0		0	1		1	0	
P1=	0	1		1	0		0	

Calculamos la paridad par en los unos

P8= Paridad Par (0101) = 0

P4= Paridad Par (1011) = 1

P2= Paridad Par (00110) = 0

P1= Paridad Par (01100) = 0

Código de Hamming

P8= Paridad Par (0101) = 0

P4= Paridad Par (1011) = 1

P2= Paridad Par (00110) = 0

P1= Paridad Par (01100) = 0

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	**Posición 8**	Posición 9	Posición 10	Posición 11	Posición 12
0	**0**	0	**1**	1	0	1	**0**	0	1	0	1

000110100101

Será la palabra a enviar, pero

¿Cómo detectar un error y corregirlo?

Código de Hamming

Para detectar errores lo único que se debe realizar es la verificación de la paridad, con 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. La palabra que enviamos fue **000110100101**, si calculamos la paridad par en los unos teniendo los bits de paridad en cuenta la paridad debería ser 0 en todos los casos

000110100101

Posición 1 (P1)	**Posición 2 (P2)**	Posición 3	**Posición 4 (P4)**	Posición 5	Posición 6	Posición 7	**Posición 8 (P8)**	Posición 9	Posición 10	Posición 11	Posición 12
0	**0**	0	**1**	1	0	1	**0**	0	1	0	1

Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
P8=0					0	1	0	1
P4=1		1	0	1				1
P2=0	0		0	1		1	0	
P1=0	0	1		1	0		0	

Código de Hamming (Ningún error)

000110100101

Posición 1 (P1)	**Posición 2 (P2)**	Posición 3	**Posición 4 (P4)**	Posición 5	Posición 6	Posición 7	**Posición 8 (P8)**	Posición 9	Posición 10	Posición 11	Posición 12
0	**0**	0	**1**	1	0	1	**0**	0	1	0	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	0	1
E4	P4=1		1	0	1				1
E2	P2=0	0		0	1		1	0	
E1	P1=0	0	1		1	0		0	

Calculamos la paridad par en los unos

E8= Paridad Par (**0**0101) = 0

E4= Paridad Par (**1**1011) = 0

E2= Paridad Par (**0**00110) = 0

E1= Paridad Par (**0**01100) = 0

Código de Hamming (Un error)

000100100101

Posición 1 (P1)	Posición 2 (P2)	Posición 3	Posición 4 (P4)	Posición 5	Posición 6	Posición 7	Posición 8 (P8)	Posición 9	Posición 10	Posición 11	Posición 12
0	**0**	0	**1**	**0**	0	1	**0**	0	1	0	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	0	1
E4	P4=1		0	0	1				1
E2	P2=0	0		0	1		1	0	
E1	P1=0	0	0		1	0		0	

Calculamos la paridad par en los unos

E8= Paridad Par (**0**0101) = 0

E4= Paridad Par (**10**011) = 1

E2= Paridad Par (**0**00110) = 0

E1= Paridad Par (**000**100) = 1

ERROR EN LA POSICIÓN 5

Código de Hamming (Un error)

010110100101

Posición 1 (P1)	Posición 2 (P2)	Posición 3	Posición 4 (P4)	Posición 5	Posición 6	Posición 7	Posición 8 (P8)	Posición 9	Posición 10	Posición 11	Posición 12
0	**1**	0	**1**	1	0	1	**0**	0	1	0	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	0	1
E4	P4=1		1	0	1				1
E2	P2=1	0		0	1		1	0	
E1	P1=0	0	1		1	0		0	

Calculamos la paridad par en los unos

E8= Paridad Par (**0**0101) = 0

E4= Paridad Par (**1**1011) = 0

E2= Paridad Par (**1**00110) = 1

E1= Paridad Par (**0**01100) = 0

ERROR EN LA POSICIÓN 2

Código de Hamming (Dos errores)

010110000101

Posición 1 (P1)	**Posición 2 (P2)**	Posición 3	**Posición 4 (P4)**	Posición 5	Posición 6	Posición 7	**Posición 8 (P8)**	Posición 9	Posición 10	Posición 11	Posición 12
0	**1**	0	**1**	1	0	**0**	**0**	0	1	0	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	0	1
E4	P4=1		1	0	**0**				1
E2	P2=**1**	0		0	**0**		1	0	
E1	P1=0	0	1		**0**	0		0	

Calculamos la paridad par en los unos

E8= Paridad Par (**0**0101) = 0

E4= Paridad Par (**1**10**0**1) = 1

E2= Paridad Par (**1**00**0**10) = 0

E1= Paridad Par (**0**01**0**00) = 1

ERROR EN

LA POSICIÓN

5

Código de Hamming (Dos errores)

010110000101

Posición 1 (P1)	Posición 2 (P2)	Posición 3	Posición 4 (P4)	Posición 5	Posición 6	Posición 7	Posición 8 (P8)	Posición 9	Posición 10	Posición 11	Posición 12
0	**1**	0	**1**	**0**	0	**0**	**0**	0	1	0	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	0	1
E4	P4=1		**0**	0	**0**				1
E2	P2=**1**	0		0	**0**		1	0	
E1	P1=0	0	**0**		**0**	0		0	

Calculamos la paridad par en los unos

E8= Paridad Par (**0**0101) = 0

E4= Paridad Par (**1**0**0**01) = 0

E2= Paridad Par (**1**00**0**10) = 0

E1= Paridad Par (**0**0**0**000) = 0

NO SE ENCONTRARON ERRORES EN

AUNQUE LOS HAY

Código de Hamming (Dos errores)

0001**0**01001**1**1

Posición 1 (P1)	Posición 2 (P2)	Posición 3	Posición 4 (P4)	Posición 5	Posición 6	Posición 7	Posición 8 (P8)	Posición 9	Posición 10	Posición 11	Posición 12
0	**0**	0	**1**	**0**	0	1	**0**	0	1	**1**	1

Cálculo de paridad	Bits de paridad recibidos	Posición 3	Posición 5	Posición 6	Posición 7	Posición 9	Posición 10	Posición 11	Posición 12
E8	P8=0					0	1	**1**	1
E4	P4=1		**0**	0	1				1
E2	P2=0	0		0	1		1	**1**	
E1	P1=0	0	**0**		1	0		**1**	

Calculamos la paridad par en los unos

E8= Paridad Par (**00111**) = 1

E4= Paridad Par (**10011**) = 1

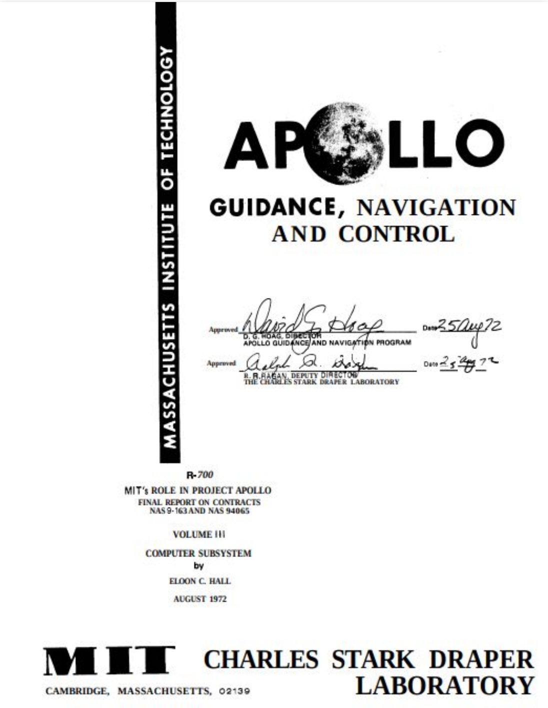
E2= Paridad Par (**000111**) = 1

E1= Paridad Par (**000101**) = 0

ERROR EN LA POSICIÓN 14

FUERA DE RANGO.MÁS DE UN ERROR

Código de Hamming


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE, NAVIGATION AND CONTROL

Approved ______ Date ______
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved ______ Date ______
R. R. RAGAN, DEPUTY DIRECTOR
THE CHARLES STARK DRAPER LABORATORY

R-700
MIT's ROLE IN PROJECT APOLLO
FINAL REPORT ON CONTRACTS
NAS 9-163 AND NAS 94065

VOLUME III

COMPUTER SUBSYSTEM
by
ELDON C. HALL

AUGUST 1972

MIT CHARLES STARK DRAPER LABORATORY
CAMBRIDGE, MASSACHUSETTS, 02139

*"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 debidos a imperfecciones en cinta 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. 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?