Tema I. Sistemas Numéricos y Códigos Binarios

- Números binarios.
- Aritmética binaria.
- Números en complemento-2.
- Códigos binarios (BCD, alfanuméricos, etc)

Números binarios

- El bit. Representación de datos mediante bits. Número mínimo de bits para representar un conjunto finito de datos.
- Representación de números en binario mediante pesos. Conversión de números entre base 10 y base 2.
- Representación de números en base 8 (octal) y base 16 (hexadecimal).

- Bit (Binary unit). Mínima cantidad de información que se puede representar. Toma dos valores: 0, 1.

  Para representar más información hay que usar conjuntos de bits. Agrupaciones de bits:

  - Nibble (4 bits), Byte (8 bits).
  - Kbit (1024 bits), Kbyte (1024 bytes).
  - 64 Kbytes (65536 bytes).

- Representación de datos mediante bits:

  Colores = {Blanco, Amarillo, Rojo, Verde, Azul, Negro}

Blanco	<=> 000;	Verde	<=> 100;
Amarillo	<=> 010;	Azul	<=> 101;
Rojo	<=> 011;	Negro	<=> 111;

- ¿Cuál es número de bits mínimo que se necesita para representar un conjunto de datos?.

  - 1 bit. 2 combinaciones (0, 1). Hasta 2 datos.
  - 2 bits. 4 combinaciones: (00, 01, 10, 11).
    Hasta 4 datos.
  - 3 bits. 8 combinaciones:
    (000, 001, 010, 011, 100, 101, 110, 111).
    Hasta 8 datos.
  - N bits: 2^N combinaciones. Hasta 2^N datos.

  Dado un conjunto de datos de M elementos se necesitan para codificarlo en binario al menos N bits, donde: $M \leq 2^N$, o

  $$N \geq \lceil \log_2(M) \rceil$$

- Representación de números en punto fijo. Se supone el punto decimal fijo en una posición.

  Normalmente los números se describen en base decimal o base 10:

$$\sum_i a_i \, 10^i \qquad a_i \in [0,9]$$

- En cualquier otra base "r": $\sum_i a_i \, r^i \qquad a_i \in [0, r-1]$

- En base 2: $\sum_i a_i \, 2^i \qquad a_i \in [0,1]$

  Los términos a_i corresponden a los dígitos del número mientras que los términos r^i (10^i o 2^i) corresponden al peso del dígito en el número.

Números en base 10

i	...	3	2	1	0	.	-1	-2	-3	...
10^i		1000	100	10	1	.	0.1	0.01	0.001	
270.75		0	2	7	0	.	7	5	0	

Números en base 2

i	...	8	7	6	5	4	3	2	1	0	.	-1	-2	-3	...
2^i		256	128	64	32	16	8	4	2	1	.	0.5	0.25	0.125	
270.75		1	0	0	0	0	1	1	1	0	.	1	1	0	
27		0	0	0	0	1	1	0	1	1	.	0	0	0	
85		0	0	1	0	1	0	1	0	1	.	0	0	0	
163		0	1	0	1	0	0	0	1	1	.	0	0	0	

- Paso de la parte entera de base 10 a base 2 por división iterativa: al dividir un número por la base aparece el coeficiente más bajo en el resto de la división. Dividir iterativamente hasta que el cociente de la división sea 0.

$(a_n \ldots a_3\ a_2\ a_1\ a_0)/\ r = (a_n \ldots a_3\ a_2\ a_1\ .\ a_0)$

$23/\ 2 = 11$. Resto 1; $a_0 = 1$

$11/\ 2 = 5$. Resto 1; $a_1 = 1$

$5/\ 2 = 2$. Resto 1; $a_2 = 1$

$2/\ 2 = 1$. Resto 0; $a_3 = 0$

$1/\ 2 = \mathbf{0}$. Resto 1; $a_4 = 1$. Fin de División

$(23)_{10} = (10111)_2 = 16 + 4 + 2 + 1$

- Paso de la parte fraccionaria de base 10 a base 2 por multiplicación iterativa: al multiplicar un número por la base aparece el coeficiente más bajo en la parte entera del producto. Multiplicar iterativamente hasta que la parte fraccionaria del producto sea 0.

$(0.a_{-1}a_{-2}\ a_{-3}\ \ldots\ a_{-n}) * r = (a_{-1}.a_{-2}\ a_{-3}\ \ldots\ a_{-n})$

$0.34375 * 2 = 0.6875.$ Parte entera 0; $a_{-1} = 0$

$0.6875 * 2 = 1.375.$ Parte entera 1; $a_{-2} = 1$

$0.375 * 2 = 0.750.$ Parte entera 0; $a_{-3} = 0$

$0.75 * 2 = 1.50.$ Parte entera 1; $a_{-4} = 1$

$0.50 * 2 = 1.\mathbf{00}.$ Parte entera 1; $a_{-5} = 1$

Parte fraccionaria es 0. Fin de Multiplicación

$(0.34375)_{10} = (0.01011)_2$

Formato Octal y Hexadecimal

- El uso de estas bases reduce el número de dígitos para representar un número binario.
- La base 8 (octal) utiliza dígitos de 0 a 7. Se pasa de base 2 a base 8 agrupando los bits de 3 en 3, y de base 8 a base 2 desagrupando los dígitos de 3 en 3 bits:

$$(101001110011)_2 \Rightarrow \overset{4\ 2\ 1}{(101)}\overset{4\ 2\ 1}{(001)}\overset{4\ 2\ 1}{(110)}\overset{4\ 2\ 1}{(011)} \Rightarrow (5\ 1\ 6\ 3)_8$$

$$(2704)_8 \Rightarrow \overset{4\ 2\ 1}{(010)}\overset{4\ 2\ 1}{(111)}\overset{4\ 2\ 1}{(000)}\overset{4\ 2\ 1}{(100)} \Rightarrow (10111000100)_2$$

- La base 16 (hexadecimal) utiliza dígitos de 0 a 15, para ello toma 0-9, A (10), B (11), C (12), D (13), E (14) y F (15). Se pasa de base 2 a base 16 agrupando los bits de 4 en 4 y de base 16 a base 2 desagrupando los dígitos de 4 en 4 bits :

$$(101001110011)_2 \Rightarrow \overset{8\ 4\ 2\ 1}{(1010)}\overset{8\ 4\ 2\ 1}{(0111)}\overset{8\ 4\ 2\ 1}{(0011)} \Rightarrow (A\ 7\ 3)_{16}$$

$$(E51C)_{16} \Rightarrow \overset{8\ 4\ 2\ 1}{(1110)}\overset{8\ 4\ 2\ 1}{(0101)}\overset{8\ 4\ 2\ 1}{(0001)}\overset{8\ 4\ 2\ 1}{(1100)} \Rightarrow (1110010100011100)_2$$

Aritmética binaria

- Sumadores: Semisumador (“Half-Adder”) y sumador completo (“Full-Adder”). Estructura “ripple” de un circuito de suma.
- Restadores.
- Multiplicadores. Multiplicación de un bit. Multiplicadores mediante suma de filas.

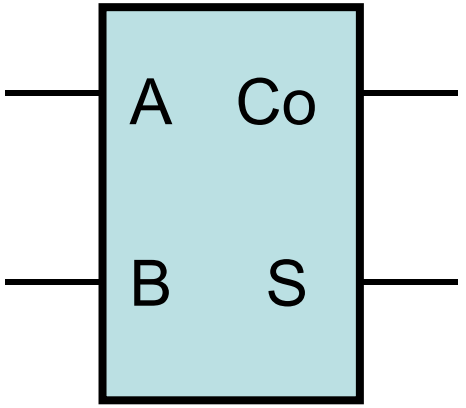
Sumadores

- Cuando sumamos dos operandos A y B de 1 bit se genera la siguiente tabla según los valores de A y B. En ella 1 + 1 = 2, que debe codificarse en dos bits (1 0). Luego hay dos bits de salida:

  - Suma S de peso 1.

  - Acarreo de salida Co ("carry out") de peso 2.

A + B	Co S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

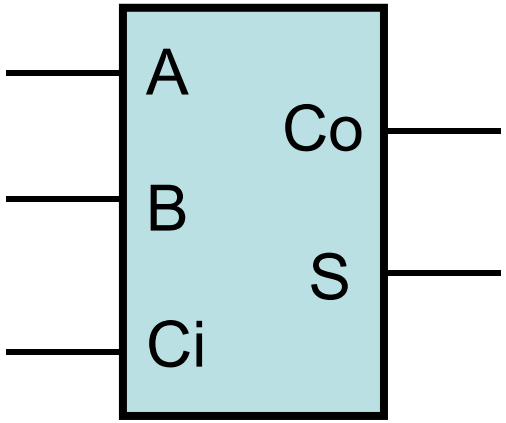


Semisumador o
"Half-Adder"

- Cuando sumamos dos operandos A y B de más de 1 bit, su primer bit se suma mediante un semisumador.

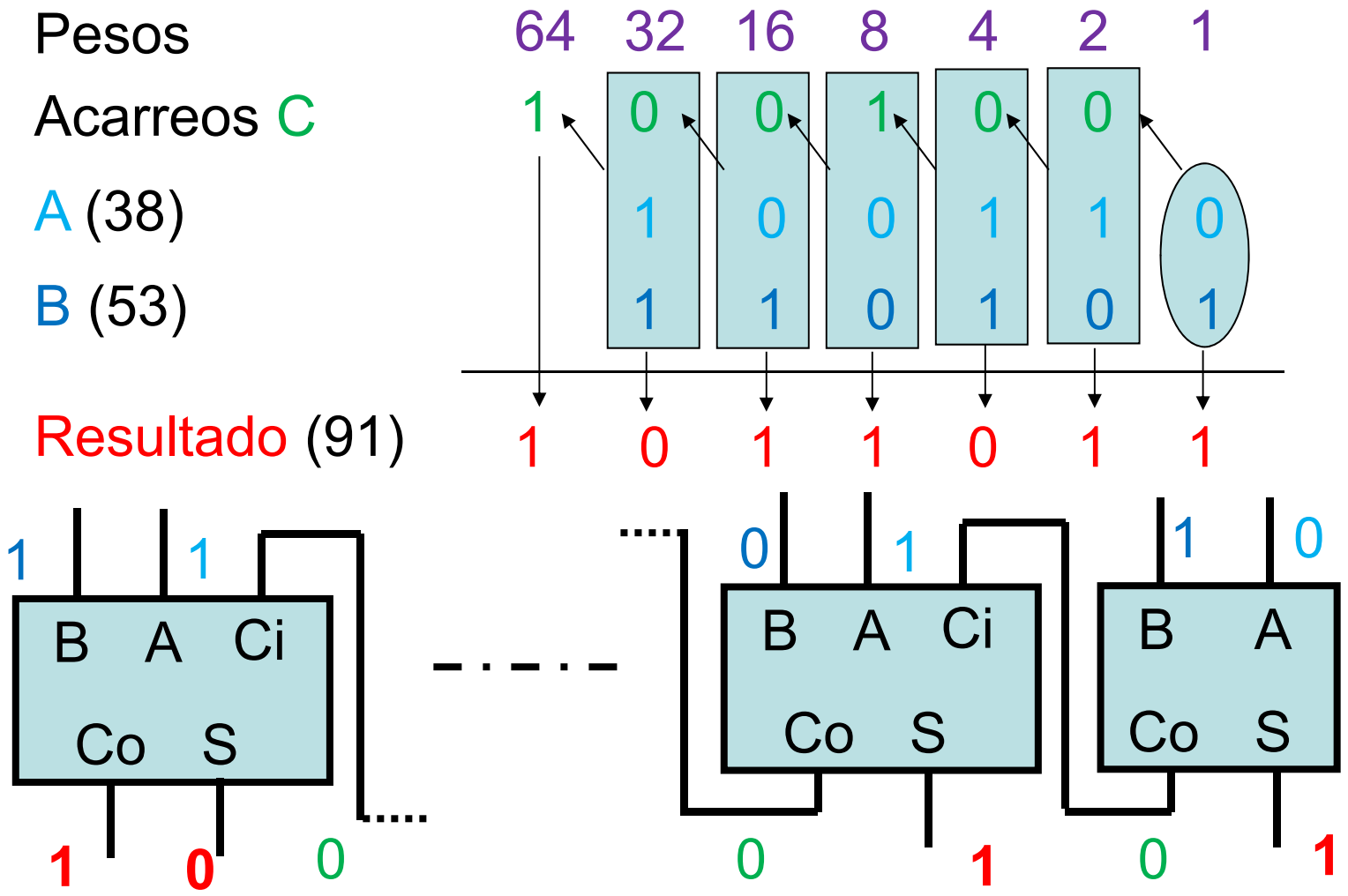
  El resto de los bits requieren tres bits de suma: los dos bits de los operandos A y B, y la salida de acarreo del bit anterior que opera como acarreo de entrada Ci ("carry in"). El máximo valor de la salida es 3 luego se requieren dos bits de salida como en el semisumador.

A +	B +	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Sumador completo o "Full-Adder"

Ahora se puede sumar bit por bit desde el menos significativo (LSB, menor peso) hacia el más significativo (MSB, mayor peso)



Restadores

- Suponemos una resta M – S con dos operandos: Minuendo (M) y Sustraendo (S), donde M ≥ S.

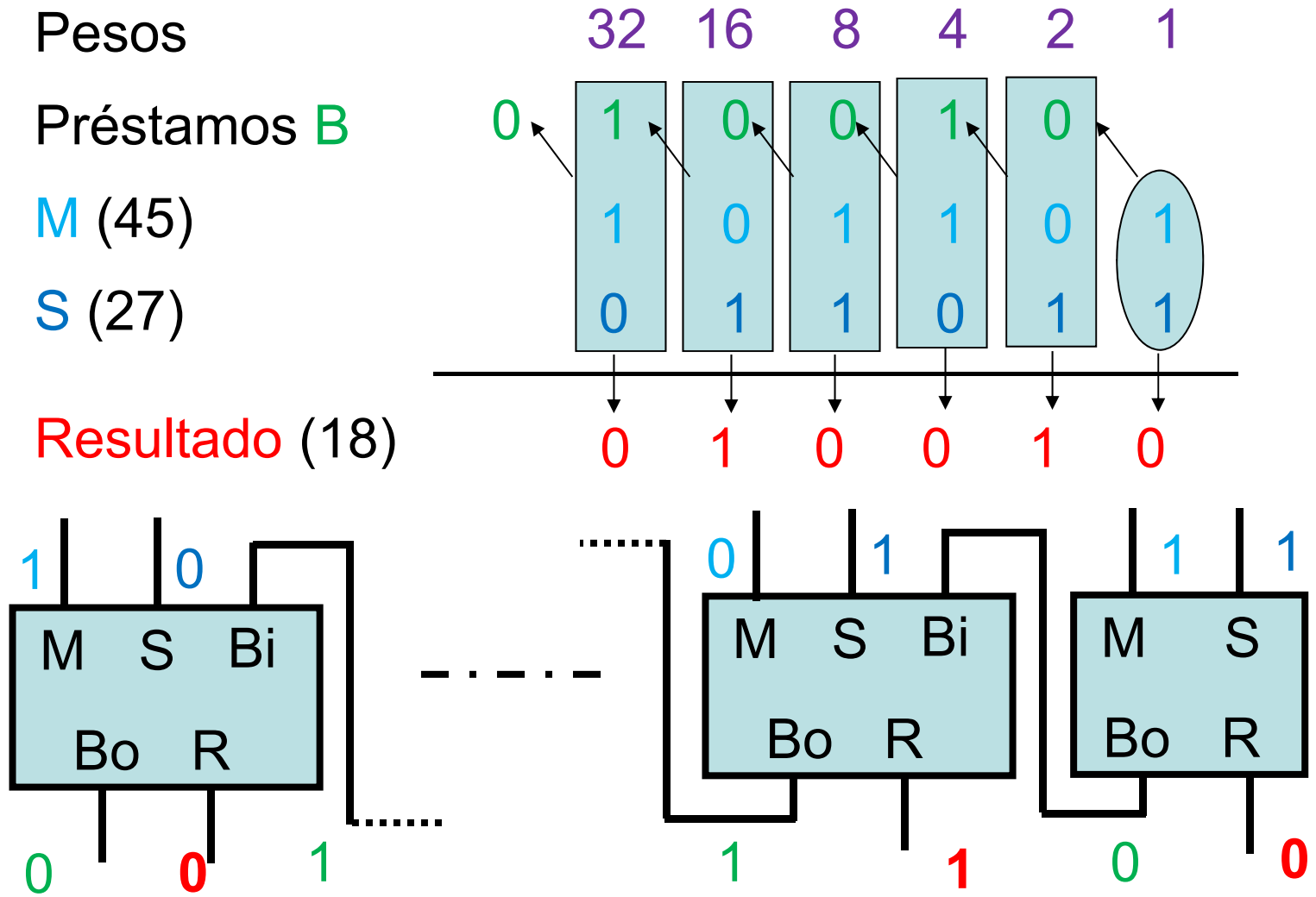
  Al restar de LSB a MSB, en 1 bit puede pasar que S > M (0 - 1), lo que obliga a pedir un bit de préstamo B ("Borrow") al siguiente bit, para realizar 10 - 1 = 1.

  En el bit que ha realizado el préstamo, se tiene que hacer primero M – B, (con la posibilidad de generar préstamo de salida), al resultado se le resta S.

M	-	S	Bo	R
0		0	0	0
0		1	1	1
1		0	0	1
1		1	0	0

(M	-	Bi)	-	S	Bo	R
0		0		0	0	0
0		0		1	1	1
0		1		0	1	1
0		1		1	1	0
1		0		0	0	1
1		0		1	0	0
1		1		0	0	0
1		1		1	1	1

Ahora se puede restar bit por bit desde el menos significativo (LSB, menor peso) hacia el más significativo (MSB, mayor peso)



Multiplicadores

- Cuando se multiplican dos operandos A y B de 1 bit se genera un resultado P de 1 bit según los valores de A y B.
- Para multiplicar operandos de más de 1 bit se utiliza el mismo algoritmo que en la multiplicación decimal, se generan los productos parciales y luego se suman.

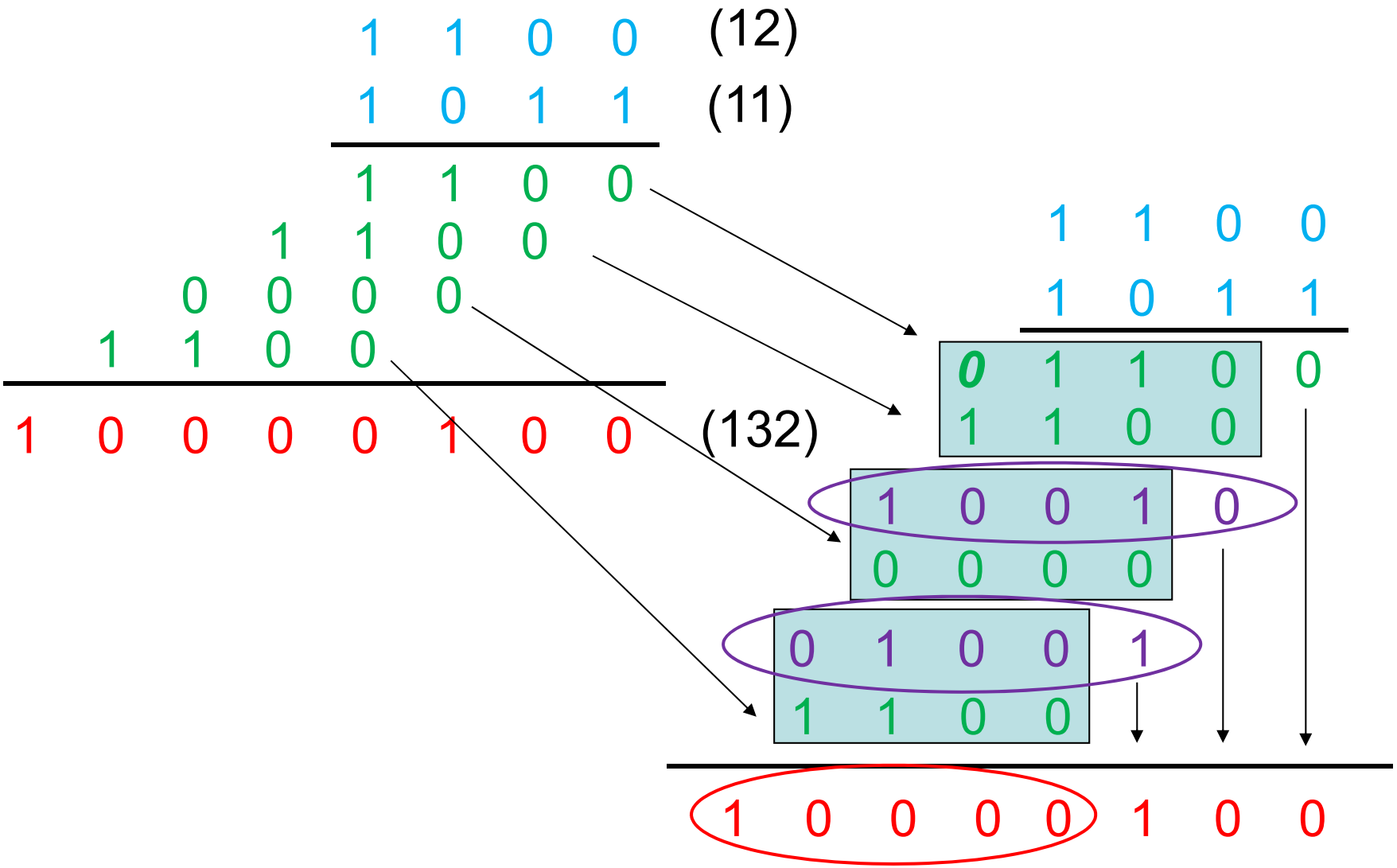
A * B	P
0 0	0
0 1	0
1 0	0
1 1	1

pij = bi * aj

a3 a2 a1 a0
b3 b2 b1 b0

p03 p02 p01 p00
p13 p12 p11 p10
p23 p22 p21 p20
p33 p32 p31 p30

P7 P6 P5 P4 P3 P2 P1 P0


1 1 0 0 (12)
1 0 1 1 (11)
1 1 0 0
1 1 0 0
0 0 0 0
1 1 0 0
1 0 0 0 0 1 0 0 (132)
1 1 0 0
1 0 1 1
0 1 1 0 0
1 1 0 0
1 0 0 1 0
0 0 0 0
0 1 0 0 1
1 1 0 0
1 0 0 0 0 1 0 0

Números en complemento-2

- Números binarios positivos y negativos. Definición mediante un bit de signo.
- Números binarios en complemento-2. Definición y generación del complemento-2 de un número binario. Definición de números negativos y positivos mediante complemento-2. Sumas y restas en complemento-2. Desbordamiento.
- Números binarios en complemento-1. Comparación con el complemento-2.

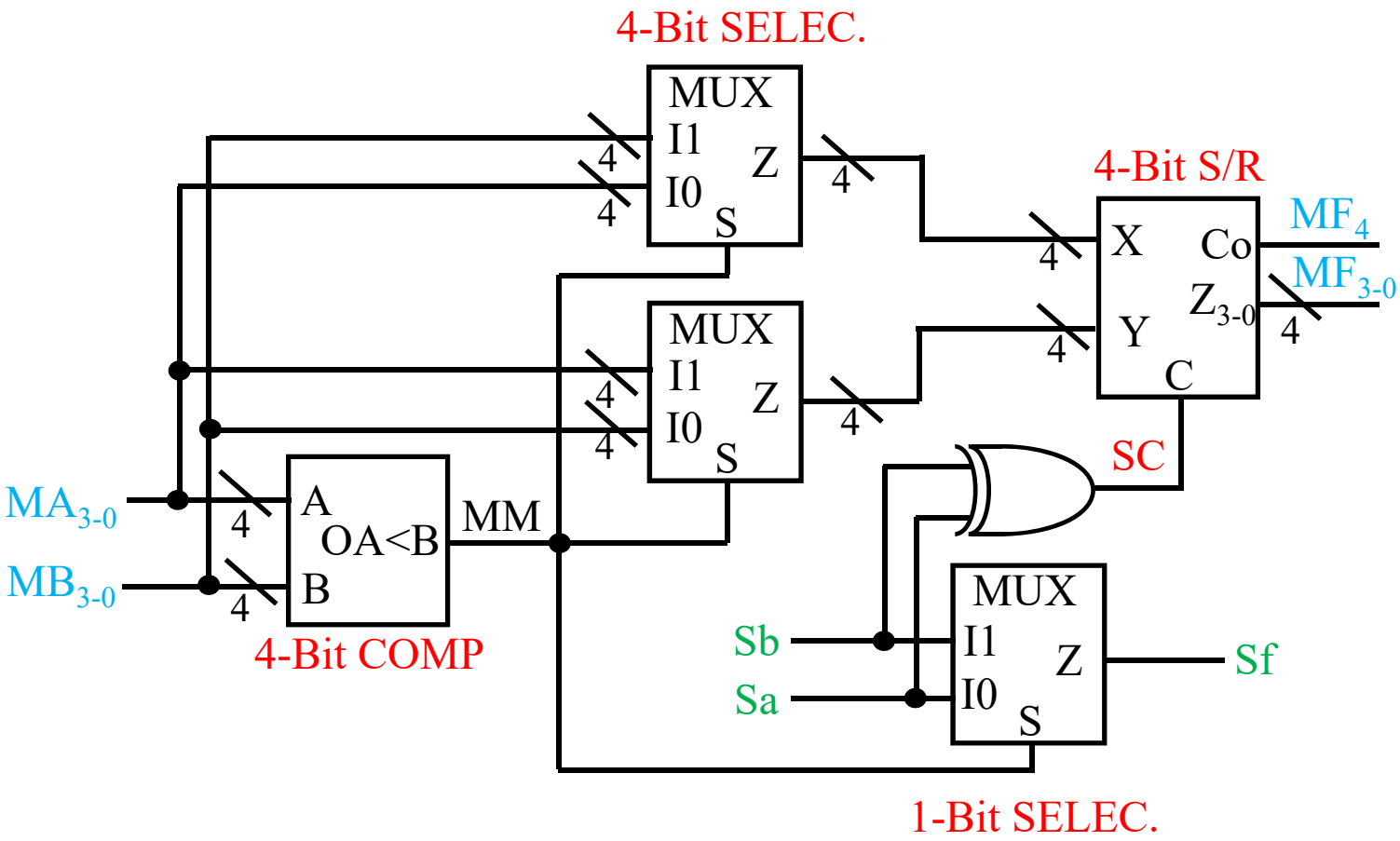
Números con signo

- Un método conceptualmente sencillo de definir números positivos y negativos consiste en un formato (S M), donde S es un bit de signo que indica si el número es positivo (0) o negativo (1), y M el módulo del número:

  +9: 0 1001 +26: 0 11010

  -9: 1 1001 -26: 1 11010

- Este método complica el diseño de circuitos para las operaciones de suma y resta. Por ejemplo, para la suma hay que comprobar si los operandos son del mismo o de distinto signo. Si son del mismo signo se debe seleccionar la suma de sus módulos, pero si no se selecciona la resta del mayor módulo menos el de menor módulo, para lo que hay que comparar primero cuál es el de mayor módulo.



4-Bit S/R => Sumador/Restador de 4 bits.
Puede construirse con un sumador de 4 bits, un restador de 4 bits y un selector de 4 bits que selecciona la salida del sumador o del restador.

Definición del complemento-2

- Para un número N descrito en una base cualquiera r con n dígitos en su parte entera y m dígitos en su parte fraccionaria (N_{n-1} … N_1 N_0 . N_{-1} … N_{-m}) se define el complemento en la base c-a-r como:

  $(N)_{r,c} = r^n - N$.

- Para un número binario r = 2, si el número tiene n bits en su parte entera entonces su c-a-2 es $(N)_{2,c} = 2^n - N$.

  Por ejemplo, si n = 5, m = 0;

  $(10010)_{2,c} = 100000 - 10010 = 01110$ =>

  $(18)_{2,c} = 32 - 18 = 14$

  Se cumple que $(N)_2 + (N)_{2,c} = 2^n$, de lo que se deduce que si un número A es $(B)_{2,c}$, entonces B es $(A)_{2,c}$.

  También se deduce que el doble complemento de A es A.

Generación del c-a-2 de un número binario

- Método rápido: de derecha a izquierda se mantienen todos los ceros hasta el primer 1 que también se mantiene; para el resto de los bits se cambian 0s por 1s y 1s por 0s.

  $(10010)_{2,c} = 01110$

- Método utilizado por los circuitos digitales: se cambian todos los bits 0s por 1s, y 1s por 0s y se suma 2^{-m} al resultado (un 1 en el bit menos significativo):

  $(10010)_{2,c} = 01101 + 1 = 01110$

  Se usa este método porque la notación en complemento se usa para definir números y operar con ellos mediante sumas. Una vez que se tiene un sumador el añadir "+ 1" a una suma es sencillo.

Números con signo en c-a-2

- Los sistemas digitales usan normalmente la notación en complemento-2 para definir los números positivos y negativos. En este sistema $(N)_{2,c} = -N$.
- La definición de números se corresponde a un sistema de pesos en binario en el que el bit de más peso tiene peso negativo. Por ejemplo, para 5 bits:

  -16 8 4 2 1

  0 0 1 0 1 => +5 = 4 + 1

  1 1 0 1 1 => -5 = -16 + 8 + 2 + 1

  Si nos fijamos en el resultado $(00101)_{2,c} = (11011)$ y $(11011)_{2,c} = (00101)$ => realizar el c-a-2 equivale a un cambio de signo.

  Los números positivos tiene el bit más significativo a 0, los negativos a 1.

- Un sistema de números enteros en c-a-2 de n bits puede definir números en el intervalo [-2^{n-1}, $2^{n-1}-1$]. Para 5 bits el intervalo está entre:

  [-16 (10000), +15 (01111)]

- Las sumas se realizan utilizando los operandos en notación c-a-2. Al realizar las sumas y restas en c-a-2 el resultado debe tener el mismo número de bits que los operandos de entrada (el bit de exceso Cout se descarta). En 7 bits (-64 32 16 8 4 2 1)

  24 = 0011000; -24 = 1101000
  10 = 0001010; -10 = 1110110
  24 – 10 = 24 + (-10) => (0011000) + (1110110) =
  (**1** 0001110); (se descarta el bit de exceso) => (0001110) => 14
  10 - 24 = 10 + (-24) = (0001010) + (1101000) =
  (**1** 1110010); (se descarta el bit de exceso) => (1110010) = -14
  -10 - 24 = (-10) + (-24) = (1110110) + (1101000) =
  (**1** 1011110); (se descarta el bit de exceso) => (1011110) = -34.

		-64	32	16	8	4	2	1
C	0	0	1	1	0	0	0	
24		0	0	1	1	0	0	0
10	+	0	0	0	1	0	1	0
34		**0**	**1**	**0**	**0**	**0**	**1**	**0**

		-64	32	16	8	4	2	1
C	1	1	1	0	0	0	0	
24		0	0	1	1	0	0	0
-10	+	1	1	1	0	1	1	0
14		**0**	**0**	**0**	**1**	**1**	**1**	**0**

		-64	32	16	8	4	2	1
C	0	0	0	1	0	0	0	
-24		1	1	0	1	0	0	0
10	+	0	0	0	1	0	1	0
-14		**1**	**1**	**1**	**0**	**0**	**1**	**0**

		-64	32	16	8	4	2	1
C	1	1	0	0	0	0	0	
-24		1	1	0	1	0	0	0
-10	+	1	1	1	0	1	1	0
-34		**1**	**0**	**1**	**1**	**1**	**1**	**0**

- Un problema añadido es el desbordamiento ("Overflow"): en 5 bits 14 + 14 = 28, luego no entra en el rango válido de números [-16, +15]. Soluciones:

  - Aumentar el número de bits mediante extensión de signo. Pasar un número a un número mayor de bits requiere extender el signo hacia la izquierda. De 5 a 6 bits:

  +14 = 01110 => 001110; -14 = 10010 => 110010

  - Detectar desbordamiento: mirar el bit de signo de los operandos, si son distintos no hay desbordamiento, si son iguales lo hay si el resultado es de distinto signo.

- Una resta no es más que la suma de un número negativo => 34 – 15 = 34 + (-15). Por tanto no es necesario utilizar circuitos sumadores y circuitos restadores, solo sumadores y un circuito que realice el c-a-2: cambiar los valores de los bits y añadir 1 a la suma.

Complemento-1

- Para un número N descrito en una base cualquiera r con n dígitos en su parte entera y m dígitos en su parte fraccionaria (N_{n-1}, …, N_1, N_0. N_{-1}, …, N_{-m}) se define el complemento en la base menos 1, c-a-(r-1) como:

  $(N)_{r-1,c} = r^n - N - r^{-m}$.

- Para un número binario r = 2, su c-a-1 es

  $(N)_{1,c} = 2^n - N - 2^{-m}$.

  Por ejemplo, si n = 5, m = 0;

  $(10010)_{1,c} = 100000 - 10010 - 1 = 01101$ =>

  $(18)_{1,c} = 32 - 18 - 1 = 13$

- Cumple reglas similares a las del c-a-2 y se puede utilizar para definir sistemas numéricos.

- El c-a-1 se calcula cambiando en todos los bits 0s por 1s, y 1s por 0s.

  $(10010)_{1,c} = 01101$

- La definición de números se corresponde a un sistema de pesos en binario en el que el bit de más peso tiene un peso negativo del tipo $-(2^n - 1)$. Por ejemplo, para 5 bits (-15 8 4 2 1)

  El intervalo de números válidos es $[-(2^{n-1} - 1), 2^{n-1} - 1]$, con dos valores para 0. Para 5 bits el intervalo está entre:

  [-15 (10000), +15 (01111)], 0 = {00000, 11111}

- Se pueden sumar números positivos y negativos como en c-a-2 pero se necesitan dos sumas: se suman los operandos y se añade 1 si el bit de exceso generado es 1. Luego se requieren dos pasos en lugar de uno como en el c-a-2.

Códigos binarios

- Códigos BCD.
- Códigos distancia unidad. Código Gray.
- Códigos alfanuméricos. Código ASCII.

Códigos BCD

- Son códigos "Binary-Coded-Decimal". Codifican en binario los dígitos decimales del 0 al 9. El conjunto de datos tiene 10 elementos => se necesitan al menos 4 bits.
- Los códigos BCD se usan en los circuitos digitales que trabajan en aritmética decimal.
- Los códigos más conocidos son el NBCD (Natural BCD) que corresponde a una codificación mediante el sistema numérico normal (pesos 8421) y el Exceso-3 que se corresponde con el NBCD + 3 (0 se codifica por 3 en binario, 1 por 4, etc), aunque se pueden buscar otras codificaciones.

	NBCD	EXC-3	7421	2-of-5
0	0000	0011	0000	00011
1	0001	0100	0001	00101
2	0010	0101	0010	00110
3	0011	0110	0011	01001
4	0100	0111	0100	01010
5	0101	1000	0101	01100
6	0110	1001	0110	10001
7	0111	1010	1000	10010
8	1000	1011	1001	10100
9	1001	1100	1010	11000

257 = 0010 0101 0111 NBCD
0101 1000 1010 Exc-3
0010 0101 1000 7421
00110 01100 10010 2-of-5

- Cada código tienes sus propiedades: el código NBCD puede utilizarse casi directamente para aritmética decimal en binario; el código EXC-3 es un código autocomplementado, el código que 7421 tiene el menor número de 1s (menor potencia disipada), el código 2-of-5 tiene siempre dos 1s, luego puede servir para detectar errores.

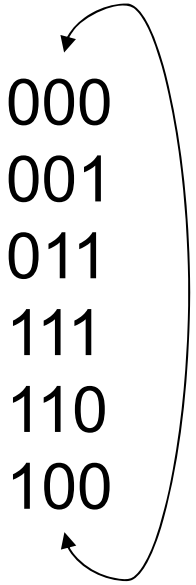
- En un código A de 4 bits (a3a2a1a0) con peso a cada bit de la codificación se le asigna un peso Wi (i de 0 a 3). El valor decimal del dígito D puede calcularse como:

  D = W3 a3 + W2 a2 + W1 a1 + W0 a0 (ai => [0, 1])

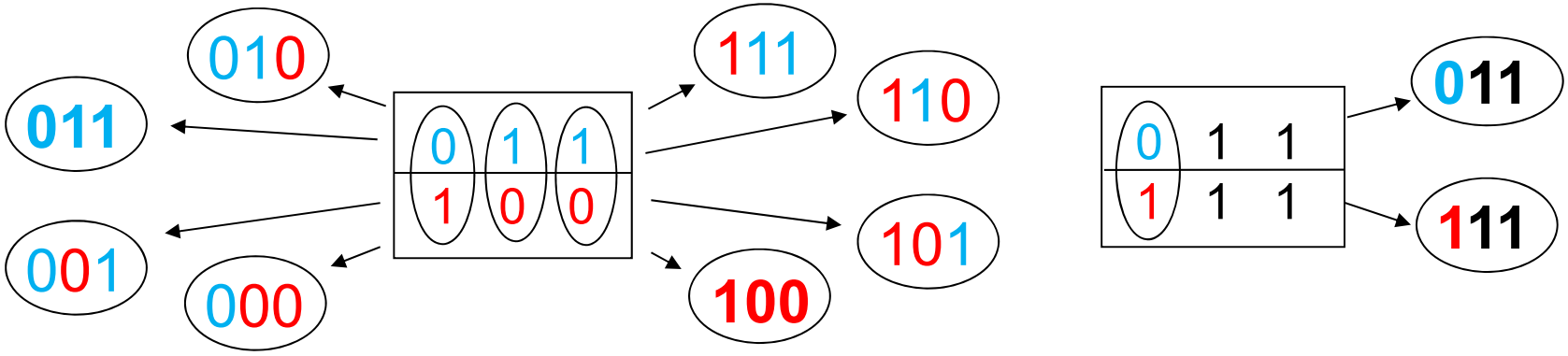
  El código NBCD es un código con peso, el Exc-3 no.
- Un código autocomplementado es aquel en el que se puede pasar del dígito D al dígito 9 – D (0 ⇔ 9, 1 ⇔ 8, 2 ⇔ 7, 3 ⇔ 6, 4 ⇔ 5) cambiando todos sus bits. Esto es útil para trabajar en c-a-9 en aritmética decimal. El código Exc-3 es autocomplementado.
- Si el paso del dígito D al dígito 9 – D del código se realiza cambiando un único bit se dice que el código es reflejado.

Códigos distancia unidad

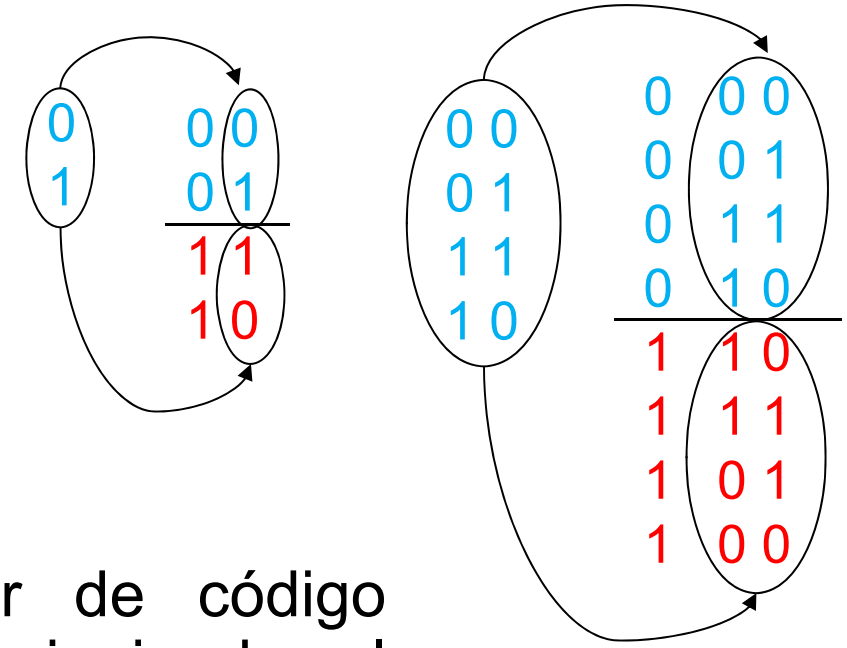
- Se define la distancia de Hamming (DH) entre dos palabras de un código como el número de posiciones en las que difiere el valor del bit: “0010” y “1011” difieren en el primer y en el último bit, luego su DH es 2.
- En un código distancia unidad dos palabras consecutivas del código tienen siempre DH = 1. La última palabra del código también tiene que tener distancia 1 con la primera.



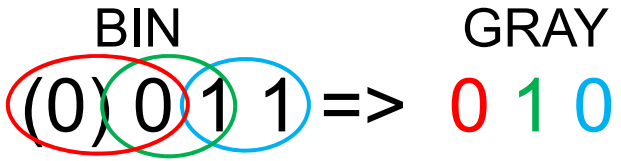
- Los códigos distancia unidad permiten reducir la existencia de error al tomar medidas. Si dos medidas consecutivas difieren en más de un bit no se puede asegurar que en algún momento no generen un resultado erróneo. En el ajuste crítico entre medidas cada bit puede tomar uno de los valores independientemente, luego puede pasar que en la separación entre 011 y 100 el valor sea 111, lo que es claramente erróneo. En códigos distancia unidad esto no puede pasar ya que solo cambia 1 bit.



- Un código binario típico de distancia unidad es el código Gray. Se forma “reflejando” los bits y rellenando con un bit nuevo los códigos superiores a 0 y los inferiores a 1.



- Se puede pasar de código binario a Gray siguiendo el número binario de derecha a izquierda: si el siguiente bit es igual al actual se pone un 0, si no un 1. Se añade un 0 a la izquierda del código binario.



Código alfanuméricos

- Los códigos alfanuméricos codifican no solo números sino también caracteres. Uno de los más conocidos es el código ASCII en 7 bits.

USASCII code chart

b_4	b_3	b_2	b_1	Row ↓ / Column →	000	001	010	011	100	101	110	111
					0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

'A' = $(41)_{16}$ = 100 0001

'3' = $(33)_{16}$ = 011 0011

'a' = $(61)_{16}$ = 110 0001

'>' = $(3E)_{16}$ = 011 1110

Códigos para transmisión de datos

- Códigos para detección de error simple: se añade al código original un bit de paridad, de forma que todas las palabras del código tengan un número de bits a 1 0 o par (paridad par), o impar (paridad impar). Un error en un bit significa un error en la paridad que es detectado al recibir el código.

	nº 1s	B.P.	Código + B.P.	nº 1s
'A' = 100 0001	2	0	1000001 0	2
'a' = 110 0001	3	1	1100001 1	4
'3' = 011 0011	4	0	0110011 0	4
'>' = 011 1110	5	1	0111110 1	6