Grado en Ingeniería de Tecnologías de Telecomunicación.
Electrónica Digital I. Problemas resueltos. Tema I

Página 1_1. Pasar de base 2 a base 10: $(1011010)_2$, $(0100111001)_2$

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

N1 = 64 + 16 + 8 + 2 = 90

512	256	128	64	32	16	8	4	2	1
0	1	0	0	1	1	1	0	0	1

N2 = 256 + 32 + 16 + 8 + 1 = 313

Página 1_2. Pasar de base 10 a base 2: 21, 58, 73, 142, 196, 273

21 = 16 + 4 + 1 => $(1\ 0\ 1\ 0\ 1)_2$
58 = 32 + 16 + 8 + 2 => $(1\ 1\ 1\ 0\ 1\ 0)_2$
73 = 64 + 8 + 1 => $(1\ 0\ 0\ 1\ 0\ 0\ 1)_2$

142. 142 > 128 => 142 = 128 + A, donde A es 142 – 128 = 14; siendo 14 = 8 + 4 + 2 =>
142 = 128 + 8 + 4 + 2 => $(1\ 0\ 0\ 0\ 1\ 1\ 1\ 0)_2$

196. 196 > 128 => 196 = 128 + A, donde A es 196 – 128 = 68; siendo 68 = 64 + 4 =>
196 = 128 + 64 + 4 => $(1\ 1\ 0\ 0\ 0\ 1\ 0\ 0)_2$

273. 273 > 256 => 273 = 256 + A, donde A es 273 – 256 = 17; siendo 17 = 16 + 1 =>
273 = 256 + 16 + 1 => $(1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)_2$

Página 1_3. Pasar de base 10 a base 2, octal y hexadecimal: 35, 97

35 = 32 + 2 + 1 => $(1\ 0\ 0\ 0\ 1\ 1)_2 = (1\ 0\ 0)\ (0\ 1\ 1) = (43)_8$;
$= (0\ 0\ 1\ 0)\ (0\ 0\ 1\ 1) = (23)_{16}$

97 = 64 + 32 + 1 => $(1\ 1\ 0\ 0\ 0\ 0\ 1)_2 = (0\ 0\ 1)\ (1\ 0\ 0)\ (0\ 0\ 1) = (141)_8$;
$= (0\ 1\ 1\ 0)\ (0\ 0\ 0\ 1) = (61)_{16}$

Página 1_4. Pasar a base 2 y a base 10: $(157)_8$, $(430)_8$

$(157)_8 = (0\ 0\ 1)\ (1\ 0\ 1)\ (1\ 1\ 1) = (1101111)_2 = (111)_{10} = 1*8^2 + 5*8 + 7$

$(430)_8 = (1\ 0\ 0)\ (0\ 1\ 1)\ (0\ 0\ 0) = (100011000)_2 = (280)_{10} = 4*8^2 + 3*8 + 0$

Página 1_5. Pasar a base 2 y a base 10: $(3B)_{16}$, $(DF)_{16}$

$(3B)_{16} = (0\ 0\ 1\ 1)\ (1\ 0\ 1\ 1) = (111011)_2 = (59)_{10} = 3*16 + 11$

$(DF)_{16} = (1\ 1\ 0\ 1)\ (1\ 1\ 1\ 1) = (11011111)_2 = (223)_{10} = 13*16 + 15$

Página 1_6. Realizar las siguientes sumas en binario: 21 + 27, 75 + 43, 98 + 87

C es el acarreo en cada bit.

	32	16	8	4	2	1
C	1	1	1	1	1	
21		1	0	1	0	1
27		1	1	0	1	1
	1	1	0	0	0	0

$21 + 27 = (110000)_2 = 48$

	128	64	32	16	8	4	2	1
C	0	0	0	1	0	1	1	
75		1	0	0	1	0	1	1
43		0	1	0	1	0	1	1
	0	1	1	1	0	1	1	0

$75 + 43 = (1110110)_2 = 118$

	128	64	32	16	8	4	2	1
C	1	0	0	0	1	1	0	
98		1	1	0	0	0	1	0
87		1	0	1	0	1	1	1
	1	0	1	1	1	0	0	1

$98 + 87 = (10111001)_2 = 185$

Página 1_7. Realizar las siguientes restas en binario: 25 – 22 , 58 – 31, 69 – 43

B es el préstamo en cada bit.

	16	8	4	2	1
B	0	1	1	0	
25	1	1	0	0	1
22	1	0	1	1	0
	0	0	0	1	1

$25 - 22 = (00011)_2 = 3$

	32	16	8	4	2	1
B	1	1	1	1	1	
58	1	1	1	0	1	0
31	0	1	1	1	1	1
	0	1	1	0	1	1

$58 - 31 = (011011)_2 = 27$

	64	32	16	8	4	2	1
B	1	1	1	0	1	0	
69	1	0	0	0	1	0	1
43	0	1	0	1	0	1	1
	0	0	1	1	0	1	0

69 - 43 = $(0011010)_2$ = 26

Página 1_8. Realizar las siguientes multiplicaciones en binario para números de 5 bits: 24 * 15, 27 * 23

Con operandos de 5 bits el resultado puede ser de hasta 10 bits (5 + 5). Los productos parciales se pueden sumar de varias formas (por filas, por columnas). En los ejemplos están sumados por columnas, de forma que 1 son acarreos generados. Si en una columna hubiese muchos 1s se pueden generar acarreos en varias columnas. Por ejemplo: 1+1+1+1 = (100), el resultado es 0 y genera acarreo no en la columna siguiente de la izquierda, sino en la columna dos posiciones a la izquierda Otro ejemplo: 1+1+1+1+1+1 = (110); el resultado es 0 y genera acarreo en las dos columnas siguientes a la izquierda.

```
   1 1 0 0 0  (24)          1 1 0 1 1  (27)
   0 1 1 1 1  (15)          1 0 1 1 1  (23)
 ---------------          ---------------
   1 1 1 0 0 0                1 1 0 1 1
 1 1 1 0 0 0              1   1 1 0 1 1
 1 1 1 0 0 0              1 1 1 0 1 1
 1 1 1 0 0 0            1 1 0 0 0 0 0 1
 1 1 1 0 0 0          1 1 1 0 1 1 1 1
 0 0 0 0 0 0                  1
---------------          ---------------
 0 1 0 1 1 0 1 0 0 0      1 0 0 1 1 0 1 1 0 1
```

512	256	128	64	32	16	8	4	2	1	N
0	1	0	1	1	0	1	0	0	0	360
1	0	0	1	1	0	1	1	0	1	621

Página 2_1. Realizar las siguientes operaciones en c-a-2 utilizando el número mínimo de bits necesario para que no haya desbordamiento: –3 + 7, 5 – 7, 10 – 6, –13 – 8.

Sabiendo que las operaciones se tienen que hacer como sumas se usan tantos bits como el máximo entre el número de bits de los operandos y el número de bits del resultado. Si los operandos son de distinto signo (una resta) se usa el número de bits mayor de los operandos; si los operandos son del mismo signo (una suma) se usa el número de bits necesario para el resultado, extendiendo los operandos si fuese necesario un bit a la izquierda (el bit más significativo se repite hacia la izquierda). Los números se codifican en c-a-2; aunque hay varias formas de hacerlo, he utilizado la notación de pesos binarios en el que el bit más significativo tiene paso negativo.

-3 + 7 = (-3) + 7 = -4. Para codificar +7 en c-a-2 se necesitan 4 bits (0111) . La fila C indica acarreos. El acarreo final se desprecia.

5 – 7 = 5 + (-7) = -2. Para codificar -7 se necesitan 4 bits (1111). La fila C indica acarreos. El acarreo final se desprecia.

	-8	4	2	1
C	1	1	1	
-3	1	1	0	1
+7	0	1	1	1
	0	1	0	0

	-8	4	2	1
C	0	0	1	
5	0	1	0	1
-7	1	0	0	1
	1	1	1	0

10 - 6 = 10 + (-6) = 4. Para codificar 10 se necesitan 5 bits. La fila C indica acarreos. El acarreo final se desprecia.

	-16	8	4	2	1
C	1	0	1	0	
10	0	1	0	1	0
-6	1	1	0	1	0
	0	0	1	0	0

-13 - 8 = (-13) + (-8) = -21. Para codificar -13 se necesitan 5 bits, pero para codificar -21 hay que usar 6 bits, por lo que los operandos se extienden 1 bit a la izquierda. La fila C indica acarreos. El acarreo final se desprecia.

	-32	16	8	4	2	1
C	1	0	0	0	0	
-13	1	1	0	0	1	1
-8	1	1	1	0	0	0
	1	0	1	0	1	1

Página 2_2. Realizar las siguientes operaciones en c-a-2 y c-a-1 para 8 bits: 75 – 43, 68 – 31, – 57 + 112, –28 – 80, –60 – 96.

En c-a-2 el método de sumar es igual que en binario, pero sin considerar el bit de acarreo final, que se desprecia. La fila C indica acarreos. Al sumar con un número fijo de bits en los operandos y el resultado puede producirse desbordamiento. El desbordamiento puede detectarse mirando los bits de signo (los más significativos) de los operandos y del resultado: si los bits de signo son iguales, y el bit de signo del resultado es distinto hay desbordamiento. Otra forma de detectar el desbordamiento utiliza el bit de acarreo final y el bit de acarreo anterior: si son distintos hay desbordamiento.

	OV	-128	64	32	16	8	4	2	1
C	1	1	0	1	1	1	1	1	
75		0	1	0	0	1	0	1	1
-43	+	1	1	0	1	0	1	0	1
32	NO	0	0	1	0	0	0	0	0

	OV	-128	64	32	16	8	4	2	1
C	1	1	0	0	0	0	0	0	
68		0	1	0	0	0	1	0	0
-31	+	1	1	1	0	0	0	0	1
37	NO	0	0	1	0	0	1	0	1

	OV	-128	64	32	16	8	4	2	1
C	1	1	0	0	0	0	0	0	
-57		1	1	0	0	0	1	1	1
112	+	0	1	1	1	0	0	0	0
55	NO	0	0	1	1	0	1	1	1

	OV	-128	64	32	16	8	4	2	1
C	1	1	1	0	0	0	0	0	
-28		1	1	1	0	0	1	0	0
-80	+	1	0	1	1	0	0	0	0
-108	NO	1	0	0	1	0	1	0	0

	OV	-128	64	32	16	8	4	2	1
C	1	0	0	0	0	0	0	0	
-60		1	1	0	0	0	1	0	0
-96	+	1	0	1	0	0	0	0	0
100	SI	0	1	1	0	0	1	0	0

El método de sumar en c-a-1 requiere un paso más que en c-a-2: el acarreo final se suma al bit menos significativo de la suma de los operandos. En el c-a-1 de números enteros de N bits el peso del bit de signo es $-2^{N-1}+1$, en vez del valor -2^{N-1} en c-a-2. El c-a-1 de un número X también se puede generar intercambiando los 0s ⇔ 1s de X. En los cálculos no se incluyen los acarreos de la suma con el acarreo final.

	OV	-127	64	32	16	8	4	2	1
C	**1**	1	0	0	0	0	0	0	
75		0	1	0	0	1	0	1	1
-43	+	1	1	0	1	0	1	0	0
		0	0	0	1	1	1	1	1
	+								**1**
32	NO	0	0	1	0	0	0	0	0

	OV	-127	64	32	16	8	4	2	1
C	**1**	1	0	0	0	0	0	0	
68		0	1	0	0	0	1	0	0
-31	+	1	1	1	0	0	0	0	0
		0	0	1	0	0	1	0	0
	+								**1**
37	NO	0	0	1	0	0	1	0	1

	OV	-127	64	32	16	8	4	2	1
C	**1**	1	0	0	0	0	0	0	
-57		1	1	0	0	0	1	1	0
112	+	0	1	1	1	0	0	0	0
		0	0	1	1	0	1	1	0
	+								**1**
55	NO	0	0	1	1	0	1	1	1

	OV	-127	64	32	16	8	4	2	1
C	**1**	1	1	0	1	1	1	1	
-28		1	1	1	0	0	0	1	1
-80	+	1	0	1	0	1	1	1	1
		1	0	0	1	0	0	1	0
	+								**1**
-108	NO	1	0	0	1	0	0	1	1

	OV	-127	64	32	16	8	4	2	1
C	**1**	0	0	1	1	1	1	1	
-60		1	1	0	0	0	0	1	1
-96	+	1	0	0	1	1	1	1	1
		0	1	1	0	0	0	1	0
	+								**1**
99	SI	0	1	1	0	0	0	1	1

Página 2_3. Realizar los códigos BCD con peso: (6, 3, 2, 1), (4, 2, 2, 1), (7, 3, 1, -2), (8, 7, -2, -4). Indicar si alguno de los códigos BCD es autocomplementario. Buscar esta condición en el caso de poder asignar varios códigos a un mismo dígito.

Para que el código sea autocomplementario se debe pasar entre todos los dígitos D ⇔ 9-D (0 ⇔ 9, 1 ⇔ 8, 2 ⇔ 7, 3 ⇔ 6, 4 ⇔ 5) cambiando todos los 0's ⇔ 1's de la codificación de los dos dígitos. Como en el código (6, 3, 2, 1) ninguna de las codificaciones del 9 es 1111, entre 0 (0000) y 9 no hay autocomplementación. Se elige una codificación para cada dígito y se genera el código BCD

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

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

El código (4, 2, 2, 1) puede codificarse de muchas maneras según se elijan las codificaciones de los dígitos 2, 3, 4, 5, 6 y 7. De estas codificaciones hay 8 combinaciones que forman códigos BCD autocomplementarios.

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

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

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

El código (7, 3, 1, -2) puede codificarse de varias maneras para los dígitos 1 y 8. Dos de estas codificaciones forman códigos BCD autocomplementarios.

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

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

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

Sólo existe un código BCD (8, 7, -2, -4) y es autocomplementario.

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

Página 2_4. Convertir las siguientes palabras de código binario a código Gray, y de código Gray a código binario.

Para pasar de binario a Gray vamos de derecha a izquierda mirando un bit i y el siguiente bit i+1, si son iguales el bit i del código de Gray es 0, si son distintos es 1. Se añade un bit a la izquierda del código binario.

BIN	(0)	0	1	1	1	0	1	1	0	0	1	0
GRAY		0	1	0	0	1	1	0	1	0	1	1

BIN	(0)	1	0	0	1	1	0	1	0	1
GRAY		1	1	0	1	0	1	1	1	1

Para pasar de Gray a binario vamos de izquierda a derecha mirando un bit i, si el valores del bit i del código de Gray es 0, indica que valor del bit i del código binario es igual al valor del bit i-1; si es 1 indica que el valor del bit i del código binario es distinto al del bit i. Se añade un bit 0 a la izquierda del código binario.

GRAY		0	1	1	1	0	1	1	0	0	1	0
BIN	(0)	0	1	0	1	1	0	1	1	1	0	0

GRAY		1	0	0	1	1	0	1	0	1
BIN	(0)	1	1	1	0	1	1	0	0	1

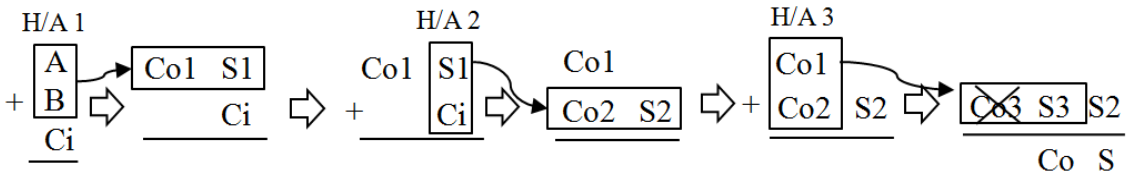
Página 2_5. Generar el código ASCII del mensaje “Hola”, añadiéndole un bit de paridad par.

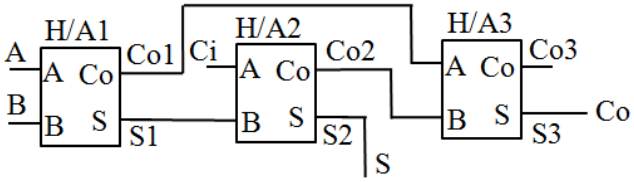
	ASCII (HEX)	ASCII (BIN)	Nº bits	Bit paridad	ASCII extendido
H	48	1001000	2	0	1001000 0
o	6F	1101111	6	0	1101111 0
l	6C	1101100	4	0	1101100 0
a	61	1100001	3	1	1100001 1

Página 3_1. Realizar el esquema de un circuito “full-adder” utilizando circuitos “half-adder”.

Un *full-adder* suma tres bits A, B, Ci generando salidas binarias de suma S y de acarreo Co (diapositiva 12 de teoría). El *half-addder* suma dos bits A y B, generando también suma S y acarreo Co (diapositiva 11 de teoría). Para sumar 3 bits con sumadores de dos bits puede hacerse los siguientes cálculos, donde las cajas verticales representan los sumadores utilizados y las cajas horizontales los resultados obtenidos. Aparecen 3 semisumadores (H/A 1, 2 y 3). Como solo se necesitan dos bits de salida el resultado Co3 no se usa y queda desconectado.

Una vez que se sabe cómo se va a realizar l circuito se puede diseñar usando tres *half-adders* y realizando las conexiones indicadas





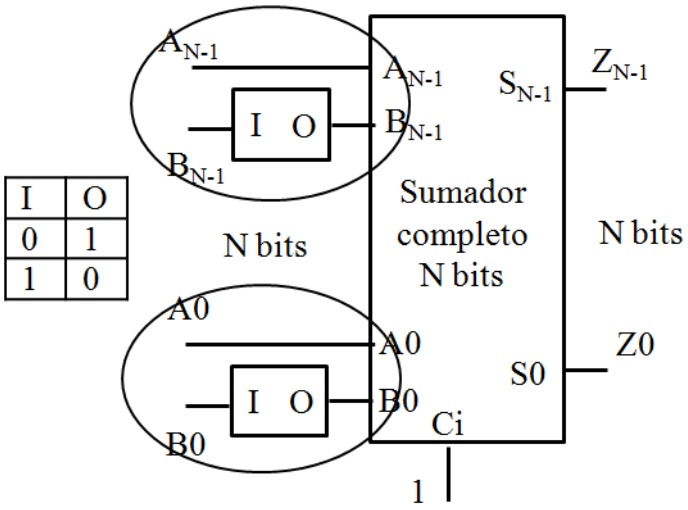
Cómo resolver problemas similares a este se discutirán en el tema IIIb. La idea es sumar los bits por columnas obteniendo al final un único bit de salida por columna; las sumas deben resolverse de los bits menos significativos a los más significativos para reducir en lo posible propagaciones de los acarreos.

Página 3_2. Construir un circuito restador en c-a-2 en base a un módulo cambiador de bits (a definir) y un sumador. Modificar el circuito para que se pueda sumar o restar según el valor de una señal de control S/R (0 Suma, 1 Resta) sustituyendo el módulo cambiador de bits por el módulo adecuado.

En c-a-2, Z = A – B = A + (-B), donde –B es $(B)_{2,c}$. $(B)_{2,c}$ se puede generar de varias formas pero la más utilizada en circuitos aritméticos es $(B)_{2,c}$ = B' + 1, donde B' significa cambiar 0s por 1s, y 1s por 0s en cada bit de B. Esta operación se llamará más adelante complementación. Para restadores de N bits (0, 1,…, N-1) se necesitarán N complementadores.

La suma se puede hacer con un sumador como en la diapositiva 13 de teoría, pero se sustituye en bit menos significativo el semisumador ("half-adder") por un sumador completo ("full-adder"), con lo que queda una entrada de acarreo que se conecta a 1 para generar el +1 del circuito que realiza el c-a-2.

Por último, al operar en c-a-2, los operandos de entrada y el resultado tienen el mismo número de bits, por lo que el acarreo de salida del bit más significativo del sumador no es necesario.

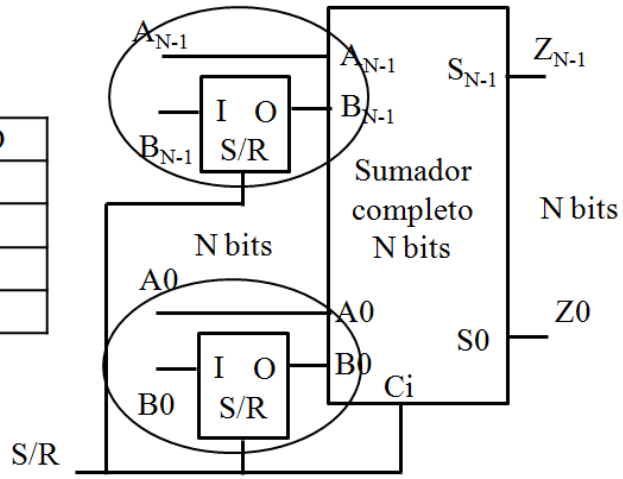


Para hacer un circuito que sume y que reste según el valor de una señal S/R, de forma que si S/R es 0 suma (Z = A + B + 0), y si S es 1 resta (Z = A + B' + 1). Es decir en función de S/R en un sumador completo las entradas se deben conectar así:

S/R	**A**	**B**	**Ci**
0	A	B	0
1	A	B'	1

La entrada A del sumador siempre será A (se conecta la entrada A a la entrada A del sumador), el valor de la entrada Ci del sumador es igual al valor de S/R, luego Ci = S/R (se conecta S/R a la entrada Ci del sumador. Para la entrada B del sumador hay que generar un circuito tal que si S/R es 0 se mantenga el valor de B y si S/R es 1 se complemente el valor de B; eso se puede poner en una tabla de 0s y 1s.

S/R	I	O
0	0	0
0	1	1
1	0	1
1	1	0



Página 3_3. Construir un circuito multiplicador de números de 4 bits, utilizando módulos de multiplicación de 1 bit y sumadores de 4 bits para realizar la suma de filas.

En las diapositivas 16 y 17 de teoría se plantea como se puede plantear este circuito. Un multiplicador A * B, donde A es de N bits y B de M bits, genera un producto P de N+M bits y requiere N*M productos parciales p_{ij} de 1 bit. Para dos operandos de 4 bits el producto P tendrá 8 salidas y se necesitan 16 circuitos para hacer la multiplicación de 1 bit p_{ij} = bi * aj, con i, j entre 0 y 3. El multiplicador de 1 bit se genera de la siguiente tabla de 1s y 0s, que corresponde a la multiplicación aritmética convencional.

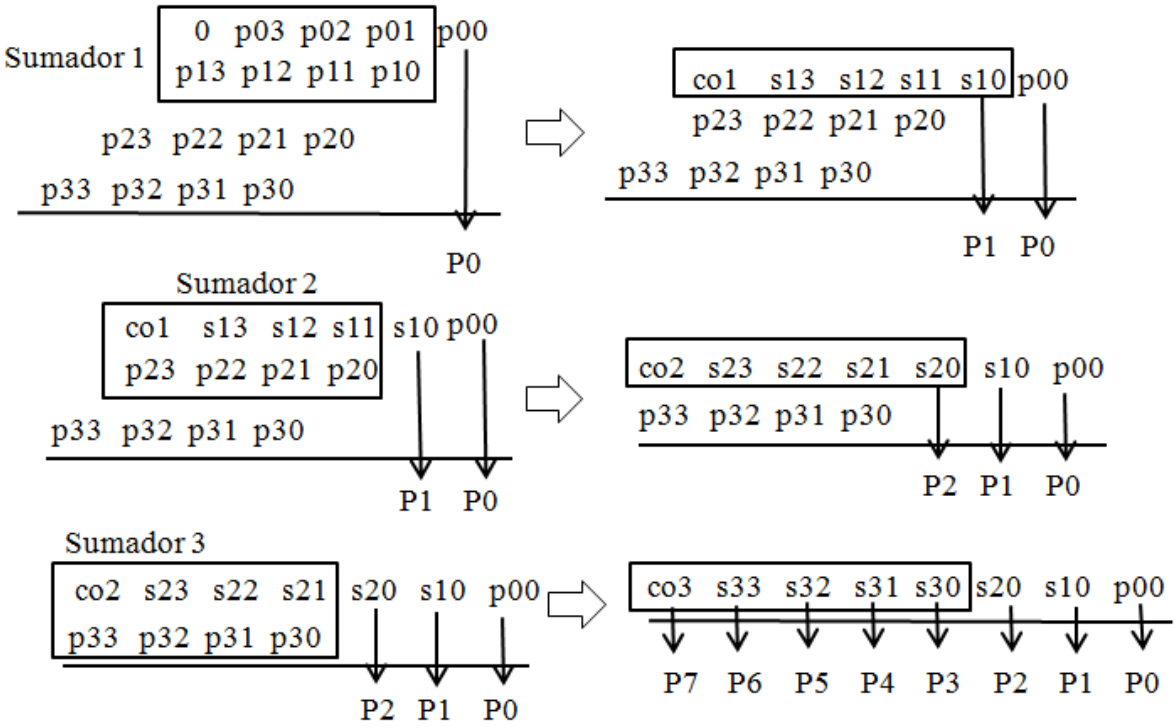
Bi	**Aj**	**Pij**
0	0	0
0	1	0
1	0	0
1	1	1

Una vez situados los productos parciales en los pesos correspondientes se deben sumar para obtener las 8 salidas del producto. Se puede sumar por filas usando tres sumadores de 4 bits ordenadamente. Los sumadores de dos operandos de varios bits aparecen en la diapositiva 13 de teoría. En este caso los sumadores tiene dos operandos de entrada X (x3 x2 x1 x0) e Y (y3 y2 y1 y0) y genera una salida Z de 5 bits (co s3 s2 s1 s0). Usando tres sumadores y la estructura

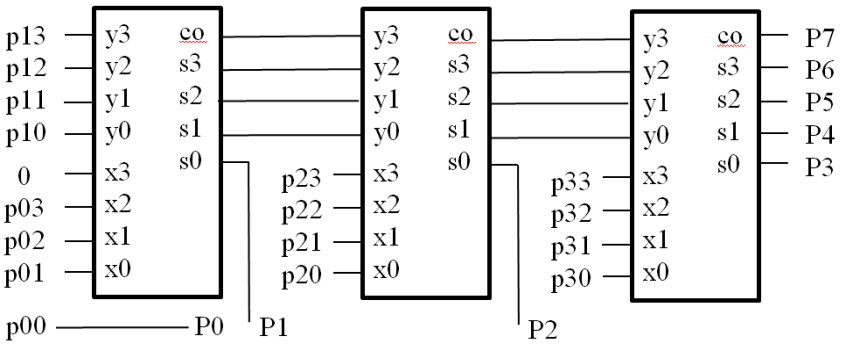
de productos parciales, se suma desde los bits menos significativos a los más significativos obteniendo un único bit en cada columna, que será el bit del resultado.

```
                    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
```

De la estructura de productos parciales se la ve que en la columna de la derecha solo hay un bit, luego ya es el bit del resultado y P0 = p_{00}. Si se suma por filas en el primer sumador X es (0 p_{03} p_{02} p_{01}), Y es (0 p_{13} p_{12} p_{11} p_{10}), y el resultado del sumador es (co_1 s_{13} s_{12} s_{11} s_{10}). Ahora s_{10} es el único bit en la segunda salida y es P1. Lo mismo se repite para los sumadores 2 y 3 como se muestra en la figura, hasta obtener el resultado final.



Conectando los sumadores se obtiene el circuito (faltan los 16 multiplicadores de 1 bit).



Página 3_4. Demostrar que la suma de los pesos de un código BCD autocomplementario es igual a 9.

En código BCD autocomplementario con pesos (W3, W2, W1, W0) se pasa del dígito D codificado por 4 bits (a3, a2, a1, a0) al dígito 9-D cambiando en los bits 0s ⇔ 1. Luego,

D = W3 a3 + W2 a2 + W1 a1 + W0 a0
9 – D = W3 a3' + W2 a2' + W1 a1' + W0 a0'

donde a' significa el cambio del valor de bit de a. Sumando las dos expresiones queda

D + 9 – D = 9 = W3 (a3 + a3') + W2 (a2 + a2') + W1 (a1 + a1') + W0 (a0 + a0'),

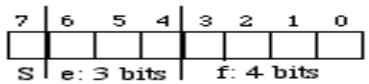
y como para cualquier ai, si ai = 0 => ai' = 1, y si ai = 0 => ai' = 1, entonces (ai + ai') = (0 + 1) = (1 + 0) = 1. Luego 9 = W3 + W2 + W1 + W0.

Página 4. Los números se pueden describir en un formato del tipo punto flotante con un formato del tipo:

$(-1)^s * 2^{e-bias} * (1.f)_2$

donde s es el bit de signo, e es el exponente, bias es una constante, y f es la parte fraccionaria de la mantisa.

En un sistema se describen números en punto flotante en 8 bits según el esquema de la figura, usando bias = 3.



a). Encontrar el valor del número (en base 10) correspondiente a los siguientes datos correspondientes a esos 8 bits dados en código hexadecimal: $(7D)_{16}$, $(A6)_{16}$

b). Indicar en este formato en punto flotante, mostrándolo en código hexadecimal los números: $(3.75)_{10}$, $(-0.625)_{10}$

a1) $(7D)_{16}$ **=>** $(0111\ 1101)_2$

S	**e**			**f**			
7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1

S = 0; e = $(111)_2 = (7)_{10}$; f = $(1101)_2$ => N1 = $(-1)^0 * 2^{7-3} * (1.\ 1101)_2 = 1 * 16 * 1.8125 = 29$

a2) $(A6)_{16}$ **=>** $(1010\ 0110)_2$

S	**e**			**f**			
7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	0

S = 1; e = $(010)_2 = (2)_{10}$; f = $(0110)_2$ => N2 = $(-1)^0 * 2^{2-3} * (1.0110)_2 = -1 * 0.5 * 1.375 =$ = -0.6875

b1) N3 = $(3.75)_{10} = (11.11)_2 = (1.1110)_2 * 2^1 = (-1)^0 * 2^{4-3} * (1.1110)_2$ => S = 0;
e = $(4)_{10} = (100)_2$; f = $(1110)_2$.

S	**e**			**f**			
7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0

N3 = $(0100\ 0110)_2$ => $(4E)_{16}$

b2) N4 = $-(0.625)_{10} = -(0.101)_2 = -(1.0100)_2 * 2^{-1} = (-1)^1 * 2^{2-3} * (1.0100)_2$ => S = 1;
e = $(2)_{10} = (010)_2$; f = $(0100)_2$.

S	**e**			**f**			
7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	0

N4 = $(1010\ 0100)_2$ => $(A4)_{16}$

Página 5. La representación de números con signo puede hacerse mediante un código de dígitos con signo, en el que cada dígito puede tomar tres valores: +1, 0 y -1. El valor +1 añade el peso positivo en binario natural (2^i: 1, 2, 4, 8, 16, …) del dígito al valor final del número, el valor -1 añade el peso negativo, y el valor 0 no añade peso. Cada dígito Di se representa en binario por dos bits (Si Ri), y el valor del dígito Di = Si – Ri: (1 0) es +1, (0 1) es –1; (0 0) y (1 1) son 0.
Un número N en código de dígitos con signo se puede pasar a un número X en complemento-2 en tres pasos:
- Para cada dígito Di, Yi es 1, si el dígito i es +1 ó –1, y 0 si el dígito es 0.
- Para cada dígito Di, Zi se obtiene del dígito menos significativo (D0) al más significativo (Dn). Z0 = 0; para el resto de los dígitos Zi es 1 si el dígito D(i-1) toma el valor –1, si el dígito D(i-1) toma el valor +1 Zi es 0, y Zi = Z(i-1) si el dígito D(i-1) toma el valor 0.
- Xi es 1 si Yi y Zi son distintos, y Xi es 0 si Yi y Zi son iguales.

Indicar el valor de los siguientes números descritos en código redundante y convertirlos a su correspondiente codificación en complemento-2-

a) (01) (11) (10) (01) (11) (11) (01) (10)

i	7	6	5	4	3	2	1	0	
Peso N	128	64	32	16	8	4	2	1	
SR	01	11	10	01	11	11	01	10	
D	-1	0	+1	-1	0	0	-1	+1	N = -113
Y	1	0	1	1	0	0	1	1	
Z	0	0	1	1	1	1	0	0	
X	1	0	0	0	1	1	1	1	X = -113
Peso X	-128	64	32	16	8	4	2	1	

b) (11) (10) (10) (00) (11) (01) (01) (00)

i	7	6	5	4	3	2	1	0	
Peso N	128	64	32	16	8	4	2	1	
SR	11	10	10	00	11	01	01	00	
D	0	+1	+1	0	0	-1	-1	0	N = +90
Y	0	1	1	0	0	1	1	0	
Z	0	0	1	1	1	1	0	0	
X	0	1	0	1	1	0	1	0	X = +90
Peso X	-128	64	32	16	8	4	2	1	

Página 6. El método de Booth permite la multiplicación de números con signo en complemento-2 . En este método la multiplicación de un multiplicando A de N bits (a_{n-1}, a_{n-2}, …, a_1, a_0) por un multiplicador B de M bits (b_{m-1}, b_{m-2}, …, b_1, b_0) produce un resultado R de N+M bits (r_{n+m-1}, r_{n+m-2}, …, r_1, r_0).
Para multiplicar se puede usar un algoritmo convencional de multiplicación sin signo con suma por filas, pero con sumas en complemento-2, y para cada bit i de B se debe tomar el par (b_i b_{i-1}), tomando b_{-1} como 0 y si:
- (b_i b_{i-1}) es (1 0): se suma el complemento-2 de A desplazado a la columna i (equivale a restar A).
- (b_i b_{i-1}) es (0 1): se suma A desplazado a la columna i.
- (bi bi-1) es (0 0) ó (1 1): se suma 0 desplazado a la columna i.
Para realizar correctamente todas las operaciones en las sumas por filas, el bit de signo de la primera fila debe extenderse dos bits a la izquierda, y el de las siguientes filas y el de los resultados de las sumas parciales debe extenderse solo un bit.
Realizar las multiplicaciones: -7 * -3, -6 * 6, -2 * 5, 5 * 7 para operandos de 4 bits por el método de Booth.

En c-a-2 un multiplicador P = A * B, donde A es de N bits y B de M bits, y generan un producto P de N+M bits. Para dos operandos de 4 bits el producto P tendrá 8 (-8 * -8 = +64, que en c-a-2 necesita 8 bits: 01000000). Los productos los realizo en una tabla organizada por filas, que se explica para el primer producto -7 * -3. La fila 1 indica el índice i de las posiciones de los bits; la fila 2 los pesos de los bits en números de 4 bits en c-a-2. Las filas 3 y 4 son los operandos A (-7) y B (-3) en complemento-2; el operando B se extiende por la derecha en la posición -1 añadiendo un 0. Los bits correspondientes a extensiones aparecen en notación cursiva.

1	i	7	6	5	4	3	2	1	0	-1
2	(-7)*(-3)					-8	4	2	1	
3	A = -7					1	0	0	1	
4	B = -3				*	1	1	0	1	*0*
5	-A			*0*	*0*	0	1	1	**1**	(10)
6	A		+	*1*	1	0	0	1		(01)
7			*1*	1	1	1	0	**0**		
8	-A	+	*0*	0	1	1	1			(10)
9		*0*	0	0	1	0	**1**			
10	0 +	*0*	0	0	0	0				(11)
11	P = 21	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	
12		-128	64	32	16	8	4	2	1	

En la fila 5 se leen las posiciones 0 y -1 de B, y genero el valor A, -A ó 0 a partir de la posición 0 y extendido dos posiciones a la izquierda: el valor de la columna 4 se repite en las columnas 5 y 6. En el ejemplo, los valores son (10) que genera –A (+7). En la fila 6 se hace lo mismo a partir de los bits 1 y 0 de B, y genero el valor A, -A ó 0 a partir de la posición 1 y extendido una posición a la izquierda: el valor de la columna 4 se repite en la columna 5. Para el ejemplo, los valores son (01) que genera A (-7).

En la fila 7 se suman en binario las filas 5 y 6 como números en c-a-2, despreciando el bit de acarreo final; las columnas con un único bit no hace falta sumarlas y generan directamente un bit del resultado que aparece en negrita. Al hacer la suma no aparecen en la tabla los acarreos intermedios. El resultado se extiende un bit a la izquierda.

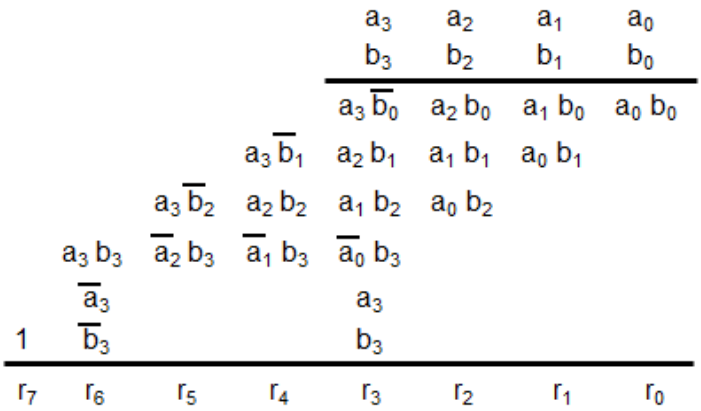
En la fila 8 se leen las posiciones 2 y 1 de B y genero el valor A, -A ó 0 a partir de la posición 2 y extendido una posición a la izquierda: el valor de la columna 5 se repite en la columna 6. Para el ejemplo, los valores son (10) que genera -A (+7). En la fila 9 se suman las filas 7 y 8, tal como se hizo en la fila 7. Esto se repite en las filas 10 y 11: se leen los bits 3 y 2 de B y genero el valor A, –A ó 0 a partir de la posición 3 (0 en el ejemplo) y extendido una posición a la izquierda: el valor de la columna 6 se repite en la columna 7, y sumo las filas 9 y 10 en la fila 11. Como esta es la última suma no se extiende su resultado por la izquierda. En la fila 12 aparecen los pesos de los bits en c-a-2 con lo que se puede calcular el resultado del producto.

1	(-6)*6	7	6	5	4	3	2	1	0	-1
2						-8	4	2	1	
3	A = -6					1	0	1	0	
4	B = 6				*	0	1	1	0	*0*
5	0			*0*	*0*	0	0	0	**0**	(00)
6	-A		+	*0*	0	1	1	0		(10)
7			*0*	0	0	1	1	**0**		
8	0	+	*0*	0	0	0	0			(11)
9		*0*	0	0	0	1	**1**			
10	A +	*1*	1	0	1	0				(01)
11	P = -36	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	
12		-128	64	32	16	8	4	2	1	

1	(-2)*5	7	6	5	4	3	2	1	0	-1
2						-8	4	2	1	
3	A = -2					1	1	1	0	
4	B = 5				*	0	1	0	1	*0*
5	-A			*0*	*0*	0	0	1	**0**	(10)
6	A		+	*1*	1	1	1	0		(01)
7			*1*	1	1	1	1	**1**		
8	-A	+	*0*	0	0	1	0			(10)
9		*0*	0	0	0	0	**1**			
10	A +	*1*	1	1	1	0				(01)
11	P = -10	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	
12		-128	64	32	16	8	4	2	1	

1	5 * 7	7	6	5	4	3	2	1	0	-1
2						-8	4	2	1	
3	A = -7					0	1	0	1	
4	B = -3				*	0	1	1	1	*0*
5	-A			*1*	*1*	1	0	1	**1**	(10)
6	0		+	*0*	0	0	0	0		(11)
7			*1*	1	1	1	0	**1**		
8	0	+	*0*	0	0	0	0			(11)
9		*1*	1	1	1	1	**0**			
10	A +	*0*	0	1	0	1				(01)
11	P = 35	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	
12		-128	64	32	16	8	4	2	1	

Página 7. El método de Baugh-Wooley permite la multiplicación de números con signo en complemento-2. En este método la multiplicación de un multiplicando A de N bits (a_{n-1}, a_{n-2}, …, a_1, a_0) por un multiplicador B de M bits (b_{m-1}, b_{m-2}, …, b_1, b_0) produce un resultado R de N+M bits (r_{n+m-1}, r_{n+m-2}, …, r_1, r_0). La fórmula que permite determinar este multiplicador está detallada en la hoja de problemas. Para dos operandos A (a3 a2 a1 a0) y B (b3 b2 b1 b0) de 4 bits, y resultado R (r7 r6 r5 r4 r3 r2 r1 r0) de 8 bits, los términos a sumar se muestran en la siguiente figura:

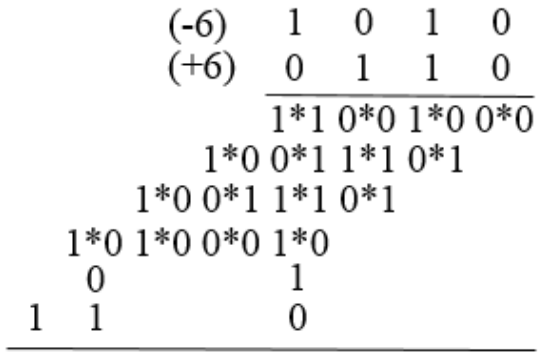


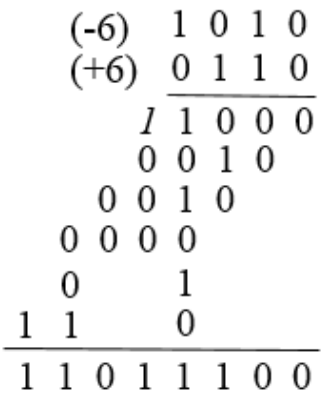
donde $\bar{a}$ *o* $\bar{b}$ indica el complemento (cambiar 0 por 1 y 1 por 0) y (a b) indica el producto aritmético de 1 bit que ya ha aparecido en otros problemas (3_3, por ejemplo).
Realizar las multiplicaciones: -7 * -3, -6 * 6, -2 * 5, 5 * 7 para operandos de 4 bits.

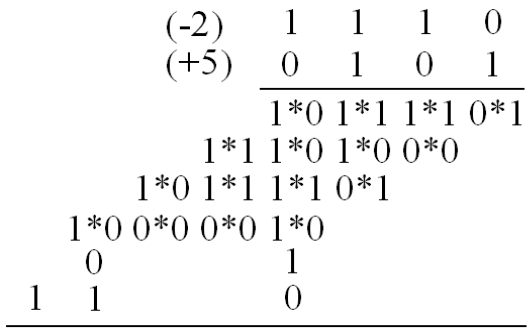
Para resolver la multiplicación se calculan los términos parciales y luego se suman. La suma se ha hecho por columnas, donde los acarreos aparecen en cursiva. Si se produce un bit de acarreo final se desprecia. Los resultados obtenidos se pueden comprobar en la solución del problema anterior.

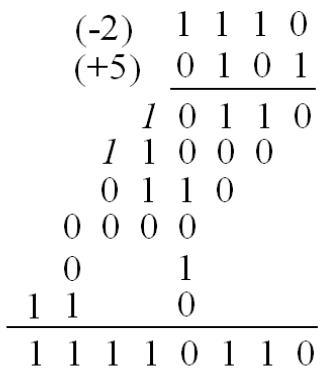
```
(-7)   1   0   0   1
(-3)   1   1   0   1
      1*0 0*1 0*1 1*1
    1*1 0*0 0*0 1*0
  1*0 0*1 0*1 1*1
1*1 1*1 1*1 0*1
  0       1
1 0       1
```

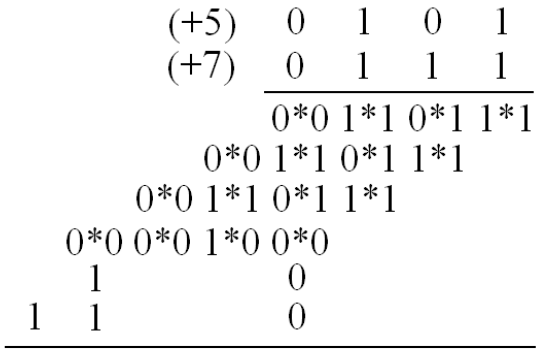
```
(-7)   1 0 0 1
(-3)   1 1 0 1
     1 0 0 0 1
   1 1 0 0 0
 1 0 0 0 1
 1 1 1 0
1 0     1
1 0     1
0 0 0 1 0 1 0 1
```

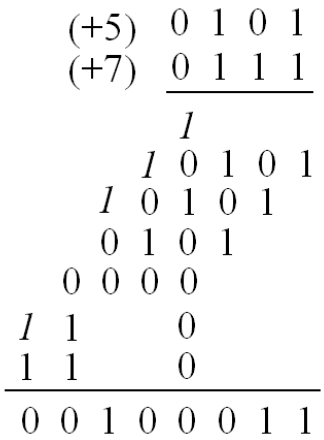












Página 8. La división A/B de dos números A (a3a2a1a0) y B (b3b2b1b0) de 4 bits, calculando el cociente Q (q3q2q1q0) y el resto R (r3r2r1r0), puede realizarse según el siguiente método:

- Tomar el dividendo como (000a3a2a1a0) y el divisor como (b3b2b1b0).
- Sea D1 los 4 bits de la izquierda (000a3) del dividendo, comparar D1 con el divisor B, si D1 ≥ B el bit del cociente q3 es 1 y se genera X (x3x2x1x0) = D1 – B; si no el cociente Q3 es 0 y X (x3x2x1x0) = D1.
- Tomar D2 como (x2x1x0a2), si D2 ≥ B el bit del cociente q2 es 1 y se genera Y (y3y2y1y0) = D2 – B; si no el cociente q2 es 0 e Y (y3y2y1y0) = D2.
- Tomar D3 como (y2y1y0a1), si D3 ≥ B el bit del cociente q1 es 1 y se genera Z (z3z2z1z0) = D3 – B; si no el cociente q1 es 0 y Z (z3z2z1z0) = D3.
- Tomar D4 como (z2z1z0a0), si D4 ≥ B el bit del cociente q0 es 1 y se genera el resto R (r3r2r1r0) = D4 – B; si no el cociente Q0 es 0 y el resto R (r3r2r1r0) = D4.

a) Indicar con ejemplos por qué el cociente y el resto deben tener 4 bits.
b) Realizar las divisiones 13/5, 11/3.

a) A toma valores entre 0 y 15, y B entre 1 y 15. El mayor cociente posible se produce en la división 15/1, produciendo Q = 15 que se codifica en 4 bits. El mayor resto se produce en la división 14/15, produciendo R = 14 que se codifica en 4 bits.

b1) 13/5

				a3	**a2**	**a1**	**a0**
Dividendo = 13	0	0	0	1	1	0	1

	b3	**b2**	**b1**	**b0**
Divisor = 5	0	1	0	1

D1 = 1	0	0	0	1	D1 < B
B = 5	0	1	0	1	X = D1
X	0	0	0	1	Q3 = 0

D2 = 3	0	0	1	1	D2 < B
B = 5	0	1	0	1	Y = D2
Y	0	0	1	1	Q2 = 0

D3 = 6	0	1	1	0	D3 ≥ B
B = 5	0	1	0	1	Z = D3 - B
Z	0	0	0	1	Q1 = 1

D4 = 3	0	0	1	1	D4 < B
B = 5	0	1	0	1	R = D4
R	0	0	1	1	Q0 = 0

Q = 2	0	0	1	0
R = 3	0	0	1	1

b2) 11/3

				a3	**a2**	**a1**	**a0**
Dividendo = 11	0	0	0	1	0	1	1

	b3	**b2**	**b1**	**b0**
Divisor = 3	0	0	1	1

D1 = 1	0	0	0	1	D1 < B
B = 3	0	0	1	1	X = D1
X	0	0	0	1	Q3 = 0

D2 = 2	0	0	1	0	D2 < B
B = 3	0	0	1	1	Y = D2
Y	0	0	1	0	Q2 = 0

D3 = 5	0	1	0	1	D3 ≥ B
B = 3	0	0	1	1	Z = D3 - B
Z	0	0	1	0	Q1 = 1

D4 = 5	0	1	0	1	D4 ≥ B
B = 3	0	0	1	1	R = D4 - B
R	0	0	1	0	Q0 = 1

Q = 3	0	0	1	1
R = 2	0	0	1	0

Página 9. El código Hamming es un código de corrección de error simple. Al código original (M0M1M2) se le añaden 3 bits de paridad P0, P1 y P2. El valor de P0, P1 y P2 se calcula según la siguiente tabla que indica en cada fila una condición de paridad par. Así cada fila F0 (P0, M0, M1), F1 (P1, M0, M2), F2 (P2, M1, M2) debe tener paridad par, con lo que del original (M0M1M2) se forma (P0P1M0P2M1M2)

1 P0	2 P1	3 M0	4 P2	5 M1	6 M2	
X		X		X		F0
	X	X			X	F1
			X	X	X	F2

Al recibir datos se comprueba si las paridades establecidas son correctas o no, en cada fila si lo están se asocia un 0, si no un 1. El código binario resultante (F2F1F0) indica la columna errónea, si el valor es 000 es que no hay error. Si hay error debe cambiarse el valor del bit de la columna afectada.
a) Calcular el código de Hamming para los tres bits M0M1M2. Comprobar que la DH entre las palabras es al menos 3.
b) Determinar si hay algún error en los siguientes datos recibidos en formato (P0P1M0P2M1M2) y decodificar el valor correcto de (M0M1M2): (101110), (111010), (001110).

a) Cálculo de los bits de paridad. Si el número de bits a 1 es 0 o par, el bit de paridad es 0, y si no es 1.

M0	M1	M2	Nº 1s (M0, M1)	P0	Nº 1s (M0, M2)	P1	Nº 1s (M1, M2)	P2
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1
0	1	0	1	1	0	0	1	1
0	1	1	1	1	1	1	2	0
1	0	0	1	1	1	1	0	0
1	0	1	1	1	2	0	1	1
1	1	0	2	0	1	1	1	1
1	1	1	2	0	2	0	2	0

Código de Hamming generado (las distancias de Hamming son mayores o iguales que 3):

P0	P1	M0	P2	M1	M2
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	1	0

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

b) Cálculo de las paridades de cada fila: Par/Impar

P0	P1	M0	P2	M1	M2	Nº 1s (P0, M0, M1)	F0
1	0	**1**	1	**1**	0	3 (I)	1
1	1	**1**	0	**1**	0	3 (I)	1
0	0	**1**	1	**1**	0	2 (P)	0

P0	P1	M0	P2	M1	M2	Nº 1s (P1, M0, M2)	F1
1	**0**	**1**	1	1	**0**	1 (I)	1
1	**1**	**1**	0	1	**0**	2 (P)	0
0	**0**	**1**	1	1	**0**	1 (I)	1

P0	P1	M0	P2	M1	M2	Nº 1s (P2, M1, M2)	F2
1	0	1	**1**	**1**	**0**	2 (P)	0
1	1	1	**0**	**1**	**0**	1 (I)	1
0	0	1	**1**	**1**	**0**	2 (P)	0

Cálculo de la columna errónea:

P0	P1	M0	P2	M1	M2	F2	F1	F0	Columna
1	0	1	1	1	0	0	1	1	3
1	1	1	0	1	0	1	0	1	5
0	0	1	1	1	0	0	1	0	2

Cálculo de los datos correctos. Se cambia el bit la columna indicada y se genera el valor final de los bits de datos M0, M1, M2.

1	2	3	4	5	6					
P0	**P1**	**M0**	**P2**	**M1**	**M2**	**F2**	**F1**	**F0**	**Columna**	**M0M1M2**
1	0	**0**	1	1	0	0	1	1	3	010
1	1	1	0	**0**	0	1	0	1	5	100
0	**1**	1	1	1	0	0	1	0	2	110

Página 10. Existen códigos binarios en los que los elementos del alfabeto se codifican con distintos números de bits. Estos códigos son útiles cuando cada elemento tiene distinta probabilidad de aparición. El código de Huffman se genera mediante el siguiente procedimiento:
Paso 1. Ordenar los elementos por probabilidad de aparición (tanto por 1) de mayor a menor.
Paso 2. Tomar los dos últimos elementos y asociar al último un bit a 0 y al penúltimo un bit a 1.

Paso 3. Sustituir los dos elementos del paso 3 por un nuevo elemento (X1, X2, …) cuya probabilidad de aparición sea la suma de la probabilidad de esos dos elementos. Mientras que haya más de 1 elemento volver al paso 1, si no ir al paso 4.
Paso 4. Generar la codificación binaria para cada elemento del alfabeto, asociándole de forma ordenada los bits que se hayan generado desde el elemento de probabilidad 1 (bit más significativo) hasta el elemento inicial (bit menos significativo), pasando por los Xi intermedios generados entre ellos (bits intermedios).

a) Obtener las codificaciones binarias mediante el código de Huffman para los elementos del alfabeto formado por A (0.3), B (0.25), C (0.20), D (0.15) y E (0.1). Entre paréntesis se incluye la probabilidad de aparición Pr de cada elemento.

b) Calcular el promedio P de bits por dato del código de Huffman.

$$\boldsymbol{P = \sum_{i} \mathrm{Pr}(i) * (n^{\underline{o}}\, bits)_i}$$

a) Paso 1a. Tabla inicial ordenada.

A	B	C	D	E
0.3	0.25	0.2	0.15	0.1

Paso 2a. Asociar 0 al último dato y 1 al penúltimo.

A	B	C	D	E
0.3	0.25	0.2	0.15	0.1
			1	0

Paso 3a. Se sustituye D y E por X1.

A	B	C	X1
0.3	0.25	0.2	0.25

Paso 1b. Tabla reordenada

A	B	X1	C
0.3	0.25	0.25	0.2

Paso 2b. Asociar 0 al último dato y 1 al penúltimo.

A	B	X1	C
0.3	0.25	0.25	0.2
		1	0

Paso 3b. Se sustituye X1 y C por X2.

A	B	X2
0.3	0.25	0.45

Paso 1c. Tabla reordenada

X2	A	B
0.45	0.3	0.25

Paso 2c. Asociar 0 al último dato y 1 al penúltimo.

X2	A	B
0.45	0.3	0.25
	1	0

Paso 3c. Se sustituye A y B por X3.

X2	X3
0.45	0.55

Paso 1d. Tabla reordenada

X3	X2
0.55	0.45

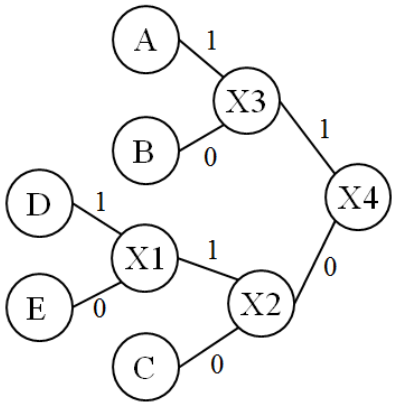
Paso 2d. Asociar 0 al último dato y 1 al penúltimo.

X3	X2
0.55	0.45
1	0

Paso 3d. Se sustituye X3 y A por X4. Su valor es 1.0 y se acaba

X4
1.0

Paso 4.



De este árbol se sacan las codificaciones de los elementos:

A: 11 (2 bits)
B: 10 (2 bits)
C: 00 (2 bits)
D: 011 (3 bits)
E: 010 (3 bits)

b) Promedio. Si se hubiese codificado en binario normal todos los elementos se codificarían en 3 bits, y en un mensaje se usarían como promedio 3 bits por elemento.

En la codificación por código de Hamming el promedio es:
P = 0.3 * 2 + 0.25 * 2 + 0.20 * 2 + 0.15 * 3 + 0.10 * 3 = 2.25 bits por elemento.