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

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 = 128 + 8 + 4 + 2 => $(1\ 0\ 0\ 0\ 1\ 1\ 1\ 0)_2$

196 = 128 + 64 + 4 => $(1\ 1\ 0\ 0\ 0\ 1\ 0\ 0)_2$

273 = 256 + 16 + 1 => $(1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)_2$

1.3. Pasar de base 10 a base 2, octal y hexadecimal: 35, 97

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

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

1.4. Pasar a base 2 y a base 10: $(157)_8$, $(430)_8$

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

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

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

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

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

1.6. Realizar las siguientes sumas en binario:

21 + 27, 75 + 43, 98 + 87.

	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

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

1.7. Realizar las siguientes restas en binario:

25 – 22 , 58 – 31, 69 – 43

	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

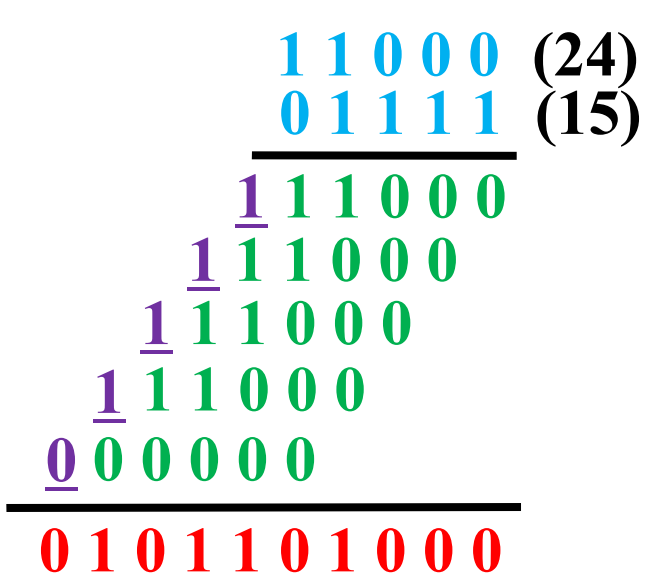
	32	16	8	4	2	1
B	1	1	1	1	1	
58	1	1	1	0	1	0
31	0	1	1	1	1	1
	0	1	1	0	1	1

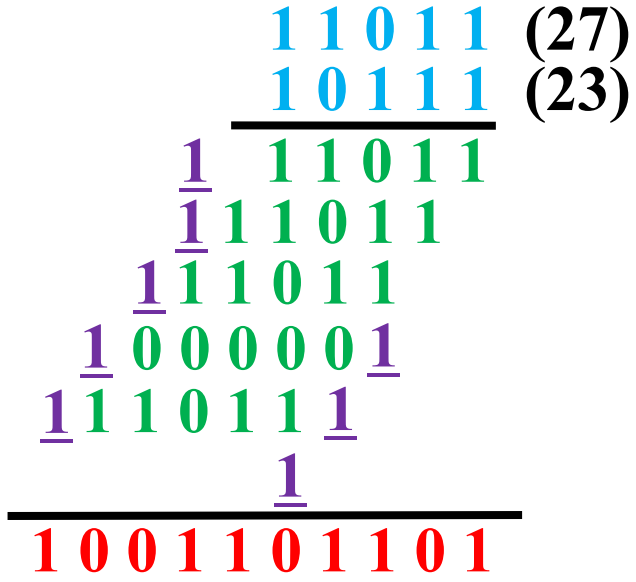
58 - 31= $(011011)_2$ = 27

	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

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





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

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.

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

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

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

	-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
-21	1	0	1	0	1	1

2.2. Realizar las siguientes operaciones en c-a-2 para 8 bits:

75 – 43, 68 – 31, – 57 + 112, –28 – 80, –60 – 96.

	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

2.2. Realizar las siguientes operaciones en c-a-2 para 8 bits: –28 – 80, –60 – 96.

	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

2.2. Realizar las siguientes operaciones en c-a-1 para 8 bits: 75 – 43, 68 – 31.

	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

2.2. Realizar las siguientes operaciones en c-a-1 para 8 bits: – 57 + 112, –28 – 80.

	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

2.2. Realizar las siguientes operaciones en c-a-1 para 8 bits: –60 – 96.

	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

2.3. Realizar los códigos BCD con peso:

(6, 3, 2, 1).

Indicar si alguno de los códigos BCD es autocomplementario. Buscar esta condición en el caso de poder asignar varios códigos a un mismo dígito.

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

Como ningún 9 tiene todos los bits a 1, el código (6, 3, 2, 1) NO puede ser autocomplementario

2.3. Realizar los códigos BCD con peso:

(4, 2, 2, 1).

Indicar si alguno de los códigos BCD es autocomplementario. Buscar esta condición en el caso de poder asignar varios códigos a un mismo dígito.

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

Al menos estos códigos (4, 2, 2, 1) son autocomplementarios.

2.3. Realizar los códigos BCD con peso:

(7, 3, 1, -2)

Indicar si alguno de los códigos BCD es autocomplementario. Buscar esta condición en el caso de poder asignar varios códigos a un mismo dígito.

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

Estos dos códigos (7, 3, 1, -2) son autocomplementarios.

2.4. Realizar los códigos BCD con peso:

(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.

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

El único código (8, 7, -2, -4) es autocomplementario.

2.4. Convertir las siguientes palabras de código binario a código Gray, y de código Gray a código binario.

(01110110010), (100110101)

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

De binario a Gray

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

De Gray a binario

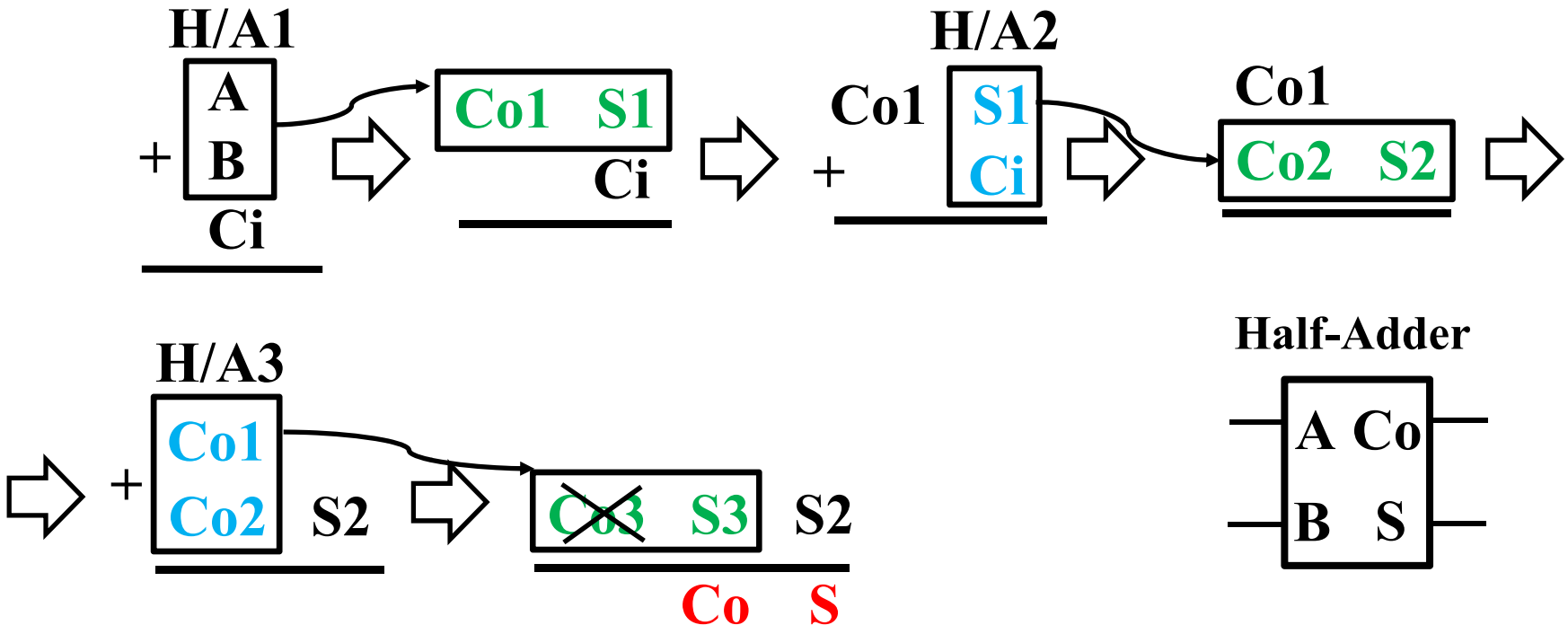
2.5. Generar el código ASCII del mensaje “Hola”, añadiéndole un bit de paridad par.

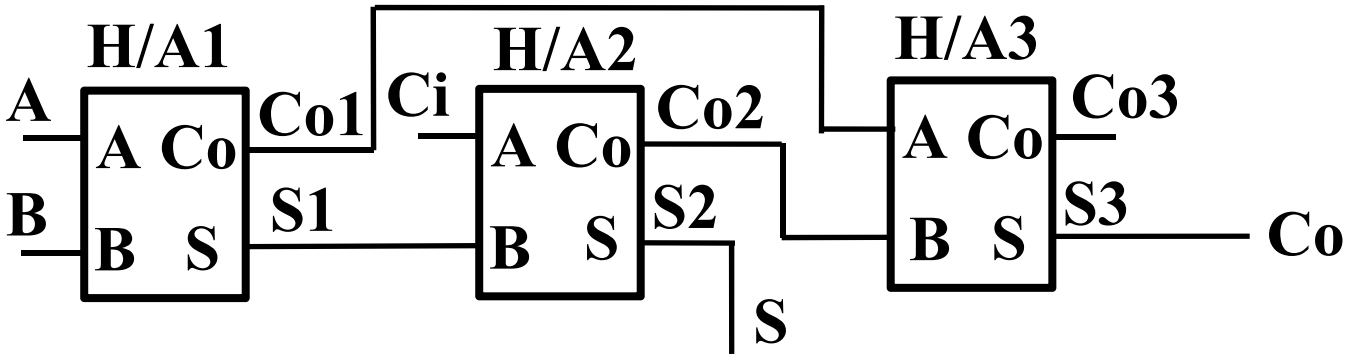
USASCII code chart

Bits $b_7 b_6 b_5$ →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	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

	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

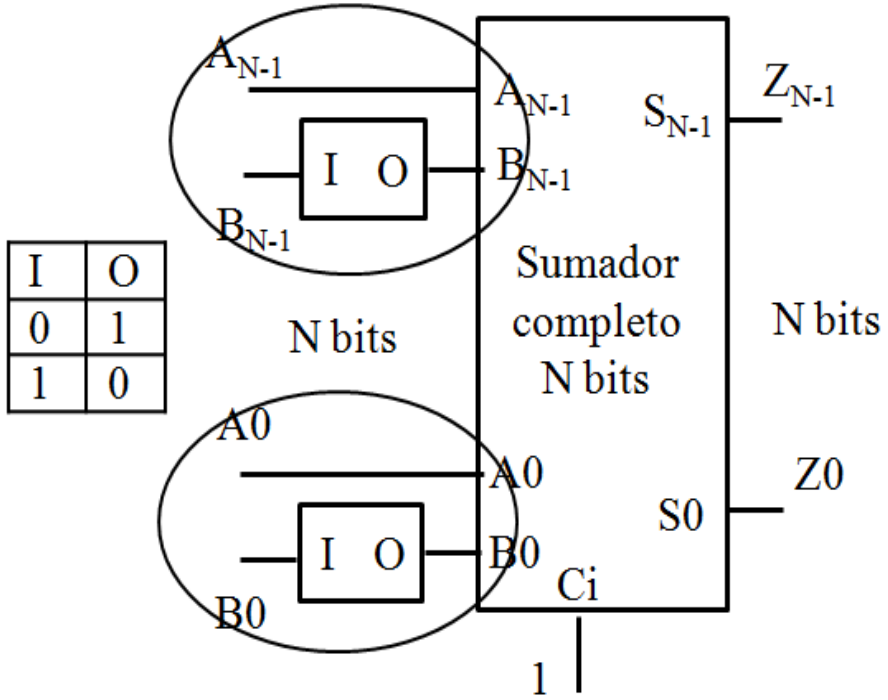
3.1. Realizar el esquema de un circuito “full-adder” utilizando circuitos “half-adder”.





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, $Z = A - B = A + (-B) = A + (B)_{2,c} = A + B' + 1$, donde B' es el cambio en B de 0s por 1s y 1s por 0s.



I	O
0	1
1	0

Restador

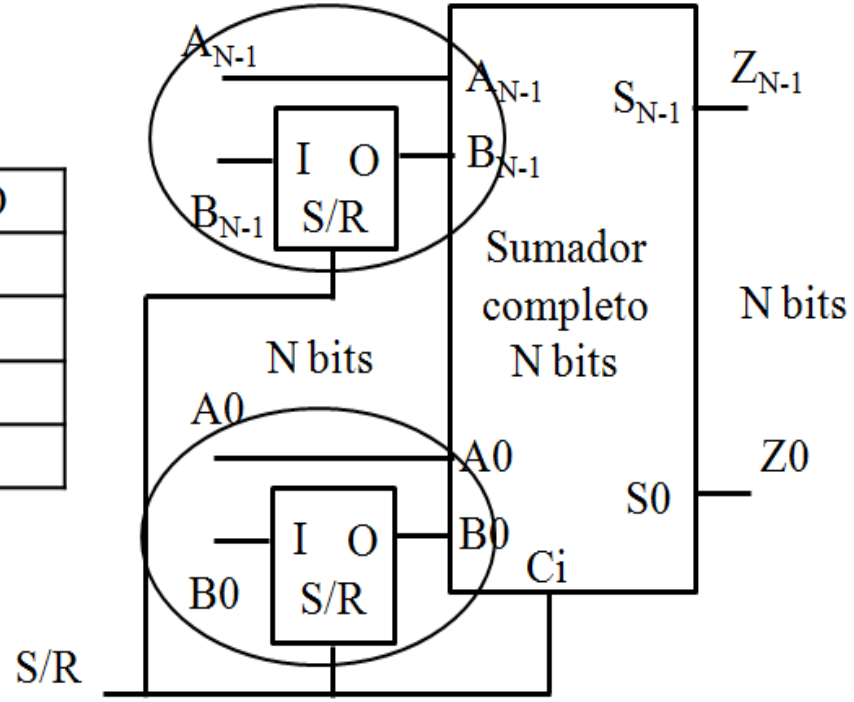
Modificar el circuito para que se pueda sumar o restar según el valor de una señal de control S/R (0 Suma, 1 Resta) sustituyendo el módulo cambiador de bits por el módulo adecuado.

Suma Z = A + B + 0

Resta Z = A + B' + 1

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

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



Sumador/Restador

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.

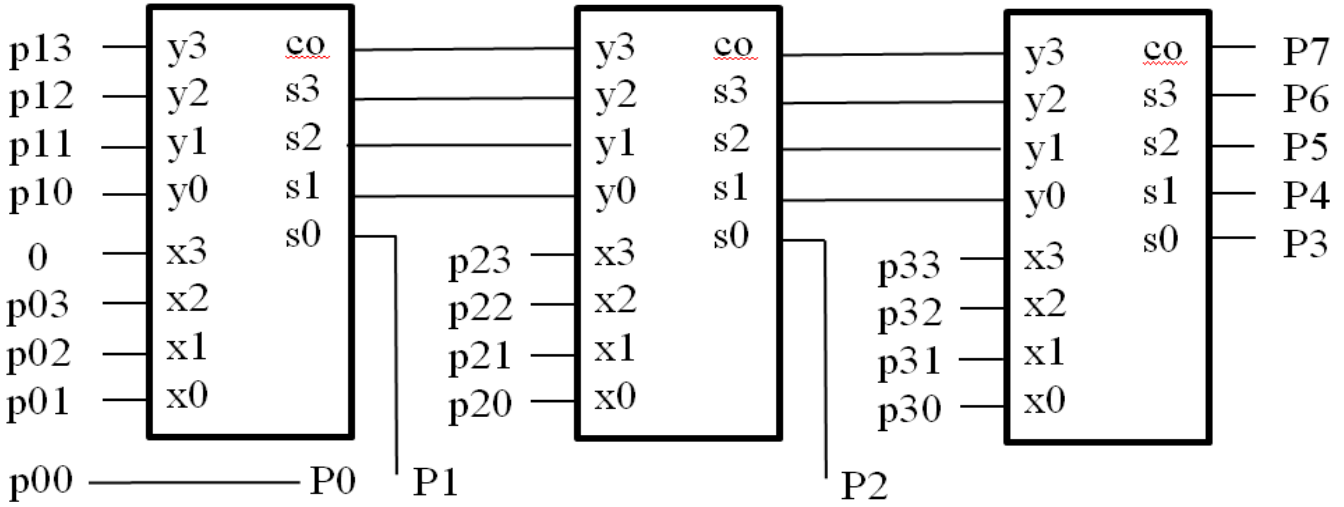
Multiplicador de 1 bit,

Pij = Bi * Aj

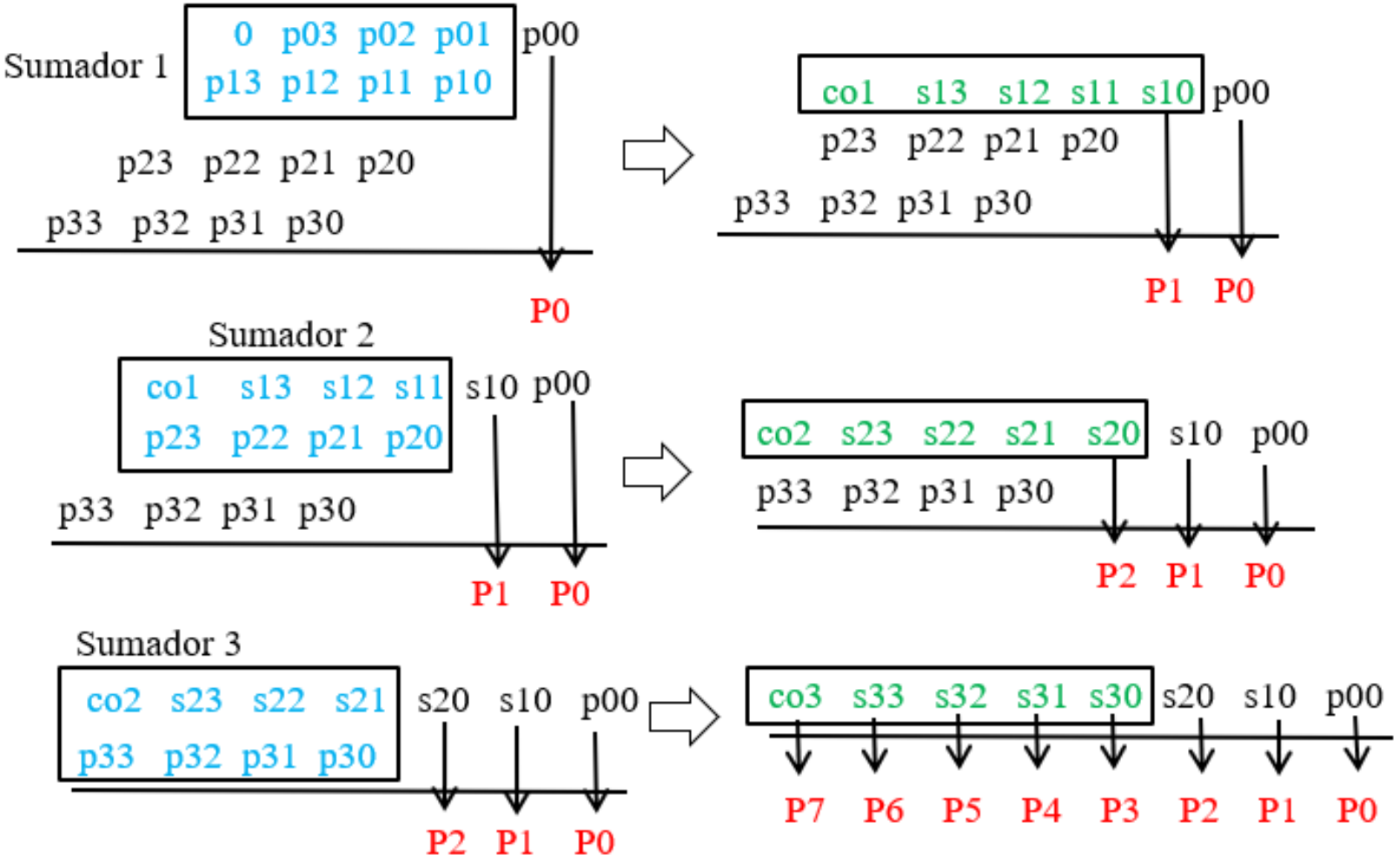
Bi	Aj	Pij
0	0	0
0	1	0
1	0	0
1	1	1

Multiplicador de 4 bits

```
          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
```



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.



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

$D = W3\ a3 + W2\ a2 + W1\ a1 + W0\ a0$

$9 - D = W3\ a3' + W2\ a2' + W1\ a1' + W0\ a0'$

donde ai' es el cambio en ai de 0s por 1s y 1s por 0s

Sumando ambos términos

$$D + 9 - D = 9 = W3\ (a3 + a3') + W2\ (a2 + a2') + W1\ (a1 + a1') + W0\ (a0 + a0')$$

Como $(ai + ai') = (0 + 1) = (1 + 0) = 1$, para ai = 0, o ai = 1

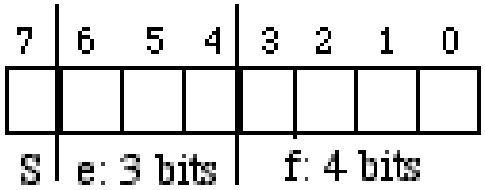
$9 = W3 + W2 + W1 + W0$

4.1 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: $(7D)_{16}$, $(A6)_{16}$

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

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 = $(1101)_2$

$N1 = (-1)^0 * 2^{7-3} * (1.1101)_2 = 1 * 16 * 1.8125 = 29$

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

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 = $(0110)_2$

$N2 = (-1)^1 * 2^{2-3} * (1.0110)_2 = -1 * 0.5 * 1.375 = -0.6875$

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

$N3 = (3.75)_{10} = (11.11)_2 = (1.1110)_2 * 2^1 = (-1)^0 * 2^{4-3} * (1.1110)_2$

$S = 0$; $e = (4)_{10} = (100)_2$; $f = (1110)_2$

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

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

$N4 = -(0.625)_{10} = -(0.101)_2 = -(1.0100)_2 * 2^{-1} =$
$= (-1)^1 * 2^{2-3} * (1.0100)_2$

$S = 1$; $e = (2)_{10} = (010)_2$; $f = (0100)_2$

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

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

5.1. 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 este 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 o –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 de dígitos con signo y convertirlos a su correspondiente codificación en complemento-2

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

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	

6.1. 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 b_{i-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.

a) -7 * -3

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	

b) -6 * 6

1	i	7	6	5	4	3	2	1	0	-1
2	(-6)*6					-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	

c) -2 * 5

1	i	7	6	5	4	3	2	1	0	-1
2	(-2)*5					-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	

d) 5 * 7

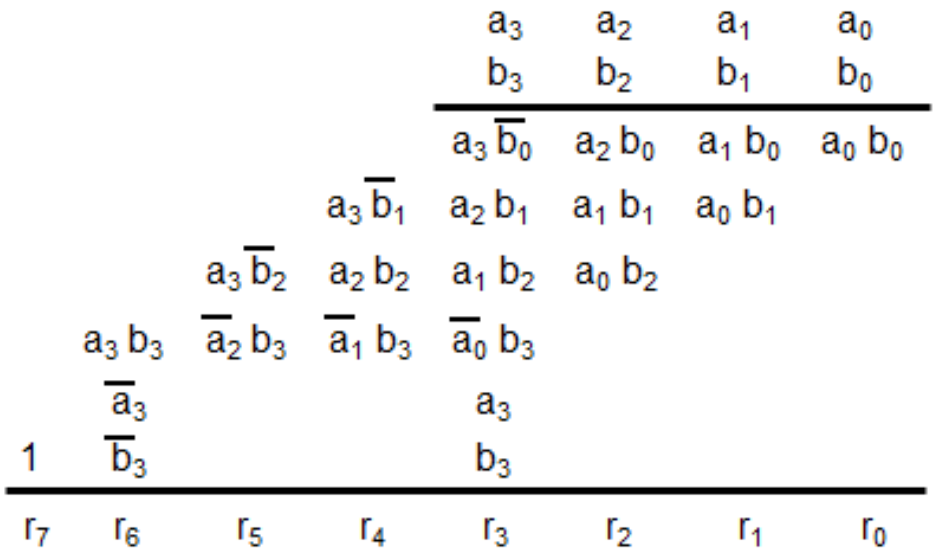
1	i	7	6	5	4	3	2	1	0	-1
2	5 * 7					-8	4	2	1	
3	A = 5					0	1	0	1	
4	B = 7				*	0	1	1	1	*0*
5	-A			*1*	*1*	1	0	1	**1**	(10)
6	0		+	*0*	0	0	0	0		(11)
7			*1*	1	1	1	0	**1**		
8	0	+	*0*	0	0	0	0			(11)
9		*1*	1	1	1	1	**0**			
10	A +	*0*	0	1	0	1				(01)
11	P = 35	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**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

7.1. 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), cuya fórmula es:

$$R = a_{n-1}b_{m-1}2^{n+m-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} a_i b_j 2^{i+j} + 2^{n+m-1} +$$
$$+ 2^{n-1}\left(\overline{a_{n-1}}2^{m-1} + a_{n-1} + \sum_{i=0}^{m-2} a_{n-1}\,\overline{b_i}2^i\right) +$$
$$+ 2^{m-1}\left(\overline{b_{m-1}}2^{n-1} + b_{m-1} + \sum_{i=0}^{n-2} \overline{a_i}\,b_{m-1}2^i\right)$$

$\overline{X}$ es el complemento de X. Cambia 1 por 0 y 0 por 1.

Por ejemplo, para dos operandos A y B de 4 bits, y resultado R de 8 bits, los términos a sumar se muestran en la siguiente figura:



Realizar las multiplicaciones:
-7 * -3, -6 * 6, -2 * 5, 5 * 7
para operandos de 4 bits.

```
(-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
```

```
(-6)   1   0   1   0
(+6)   0   1   1   0
-------------------------
       1*1 0*0 1*0 0*0
   1*0 0*1 1*1 0*1
 1*0 0*1 1*1 0*1
1*0 1*0 0*0 1*0
  0           1
1 1           0
-------------------------
```

```
(-6)     1 0 1 0
(+6)     0 1 1 0
-----------------
       1 1 0 0 0
     0 0 1 0
   0 0 1 0
 0 0 0 0
 0     1
1 1    0
-----------------
1 1 0 1 1 1 0 0
```

```
(-2)      1   1   1   0
(+5)      0   1   0   1
---------------------------
          1*0 1*1 1*1 0*1
      1*1 1*0 1*0 0*0
  1*0 1*1 1*1 0*1
1*0 0*0 0*0 1*0
  0       1
1 1       0
---------------------------
```

```
(-2)      1   1   1   0
(+5)      0   1   0   1
---------------------------
          1   0   1   1   0
      1   1   0   0   0
          0   1   1   0
  0   0   0   0
      0       1
1 1       0
---------------------------
1 1 1 1 0 1 1 0
```

```
(+5)      0   1   0   1
(+7)      0   1   1   1
---------------------------
          0*0 1*1 0*1 1*1
      0*0 1*1 0*1 1*1
  0*0 1*1 0*1 1*1
0*0 0*0 1*0 0*0
  1       0
1 1       0
---------------------------
```

```
(+5)      0   1   0   1
(+7)      0   1   1   1
---------------------------
              1
          1   0   1   0   1
      1   0   1   0   1
  1   0   1   0   1
      0   1   0   1
  0   0   0   0
1 1       0
1 1       0
---------------------------
0 0 1 0 0 0 1 1
```

8.1. 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) Indicar con ejemplos por qué el cociente y el resto deben tener 4 bits.

15/1 = 15 => Qmax = 15 => Q debe ser de 4 bits

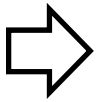
14/15 = 0, R = 14 => Rmax = 14 => R debe ser de 4 bits

b1) **13/5**

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

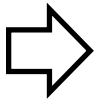
	b3	**b2**	**b1**	**b0**
Divisor = 5	0	1	0	1

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

9.1. 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	2	3	4	5	6	
P0	P1	M0	P2	M1	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) Calcular el código de Hamming para los tres bits M0M1M2. Comprobar que la DH entre las palabras es al menos 3.

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

a) Calcular el código de Hamming para los tres bits M0M1M2. Comprobar que la DH entre las palabras es al menos 3.

Código de Hamming generado (las distancias de Hamming son mayores o iguales que 3 entre cualquier palabra del código).

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) Determinar si hay algún error en los siguientes datos recibidos en formato (P0P1M0P2M1M2) y decodificar el valor correcto de (M0M1M2): (101110), (111010), (001110)

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

Cálculo de las paridades de F0

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

Cálculo de las paridades de F1

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 las paridades de F2

b) Determinar si hay algún error en los siguientes datos recibidos en formato (P0P1M0P2M1M2) y decodificar el valor correcto de (M0M1M2): (101110), (111010), (001110)

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 la columna errónea

Cálculo de los datos correctos. Valores finales de (M0M1M2)

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

10.1. 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.

$$P = \sum_i \Pr(i) \bullet (n^o bits)_i$$

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.

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

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.

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.

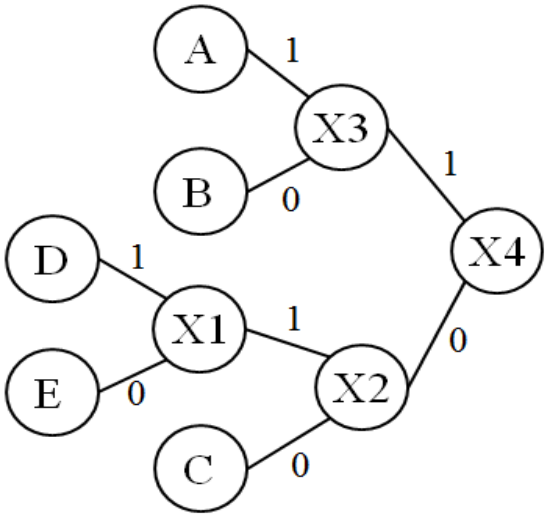
X3	X2
0.55	0.45
1	0

Paso 3d. Se sustituye X3 y X2 por X4.

X4
1.0

X4 es 1. Fin.

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.



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

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

$$P = \sum_i \Pr(i) \bullet (n^o\, bits)_i$$

P = 0.3 * 2 + 0.25 * 2 + 0.20 * 2 + 0.15 * 3 + 0.10 * 3 =

= 2.25 bits por elemento.