

Tema 4. Lógica Combinacional

Grado de robótica - USC

Prof. Juan J. Pombo García

Contenidos del tema

- Sistemas de numeración
- Introducción a las funciones aritmético-lógicas simples
 - Aspectos prácticos
- Sumadores y restadores
 - Sistemas de numeración con signo
- Comparadores
- Unidades aritmético-lógicas
- Funciones de ruta de datos
- Multiplexores y demultiplexores
- Codificadores

Sistemas de numeración

Números binarios: base 2 (*radix* 2). Cada dígito será 0 o 1.

Empezamos con la representación de números enteros SIN SIGNO:

1	0	1	0	1	0	= 42 ₁₀
32	16	8	4	2	1	Binary coefficients
2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	
MSB					LSB	

Inversamente, convertir 42₁₀ en binario: 101010₂. Calculable usando sucesivas divisiones.

Cantidad representable con n-bits: 2ⁿ-1

Sistemas de numeración. Binario fraccional

También se pueden codificar números fraccionales en modo binario (“coma binaria”)

Ej. Convertir 0.1011 a decimal

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64

En la práctica no se utiliza este método de representación. Se utiliza el modo “punto flotante” que veremos más adelante.

Sistemas de numeración. Hexadecimal

16 dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Contar en hexadecimal?

$$1010_2 = 10_{10} = A_{16}$$

$$1101_2 = 13_{10} = D_{16}$$

$$1011_2 = 11_{10} = B_{16}$$

$$1110_2 = 14_{10} = E_{16}$$

$$1100_2 = 12_{10} = C_{16}$$

$$1111_2 = 15_{10} = F_{16}$$

$$0 \quad 2 \quad A_{16} = 42_{10}$$

256	16	1	Hex
16^2	16^1	16^0	coefficients
MSB	LSB		

Para la conversión o se divide sucesivamente por 16 o se hacen grupos de 4 bit en la codificación binaria

$$31_{10} = 11111_2 = \underbrace{0001}_1 \underbrace{1111}_{15=F} = 1F_{16}$$

Sistemas de numeración. Hexadecimal (II)

Hex digit	Decimal equivalent	4-bit Binary equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Sistemas de numeración. Hexadecimal (III)

Conversión de decimal a HEX

Ej: 2000_{dec} to HEX

Sistemas de numeración. BCD

binary-coded decimal: No tiene nada que ver con lo anterior. Se ve sobre todo en sistemas antiguos, selectores, visualizadores, ...

Cada dígito se reemplaza por su equivalente binario de 4 bits.

$$951_{10} = 1001\ 0101\ 0001 = (100101010001)_{BCD}$$

Existen otros formatos de BCD.

Ejemplo de tabla de conversión

Resumen

Decimal number	Representation			
	Hexadecimal	Octal	Binary	BCD
0	0	0	0000	0000
1	1	1	0001	0001
2	2	2	0010	0010
3	3	3	0011	0011
4	4	4	0100	0100
5	5	5	0101	0101
6	6	6	0110	0110
7	7	7	0111	0111
8	8	10	1000	1000
9	9	11	1001	1001
10	A	12	1010	0001 0000
11	B	13	1011	0001 0001
12	C	14	1100	0001 0010
13	D	15	1101	0001 0011
14	E	16	1110	0001 0100
15	F	17	1111	0001 0101

Otros sistemas de representación. Gray

Existen otros sistemas de representación que no son “posicionales”: los bits no tienen un peso. Estos formatos no son adecuados para cálculos aritméticos, pero si para representación de datos.

Algunos de ellos presentan características interesantes de cara a comunicación, corrección de errores, etc.

Código gray:

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

Otros sistemas de representación.

Otros sistemas de representación de datos binarios de datos son:

- *p-out-of-n*: usa siempre “p” bits 1 y (n-p) bit en valor 0. Usado en entornos donde interesa filtrar errores, ej. algunos códigos de barras (2-out-of-5)



- *ASCII code (American standard code for information Exchange)*
- *EBCDIC*
- *ANSI*
- *Unicode (UTF-8, UTF-16, UTF-32)*

Más adelante profundizaremos algo más en estas representaciones, y en algunas otras que tienen propiedades de corrección de errores.

Representación de los números enteros con signo

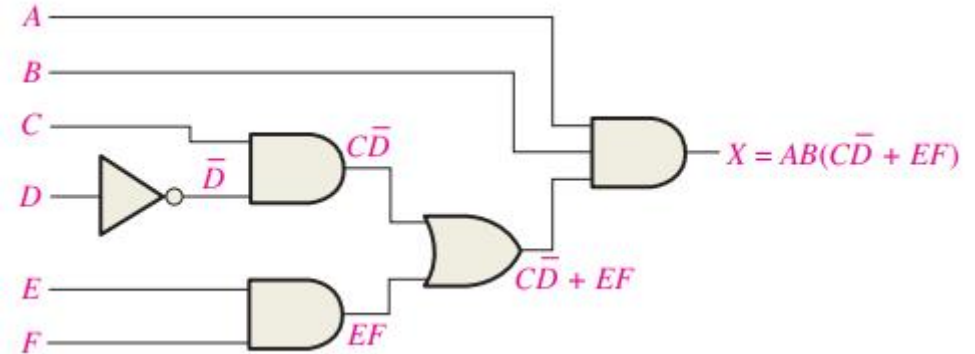
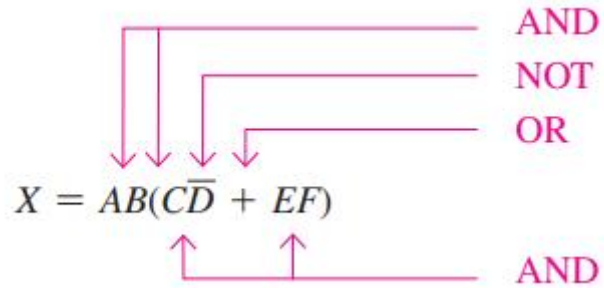
Se han adoptado distintas representaciones para los enteros con signo. Los que han alcanzado mayor difusión:

- Signo-magnitud (SM)
- Complemento a 2 (2C)
- excess-E (XSE)

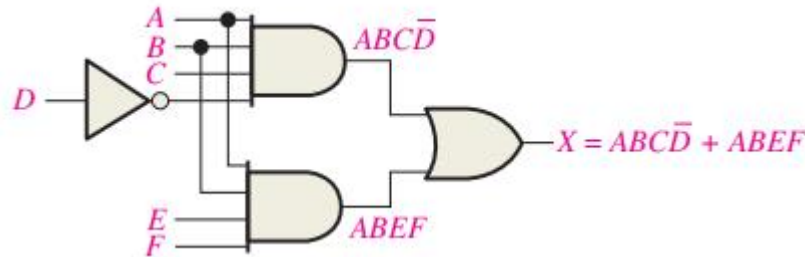
Pero antes de profundizar en estos formatos analizaremos las bases de la aritmética digital combinacional de manera que después entenderemos mejor el potencial de cada formato.

Primeros circuitos combinacionales

Pasar de una expresión booleana a un circuito es bastante directo.



Si quisiéramos disminuir el número de niveles para reducir los tiempos de propagación del circuito podríamos pasar a una forma SOP donde limitamos el número de niveles a 3:

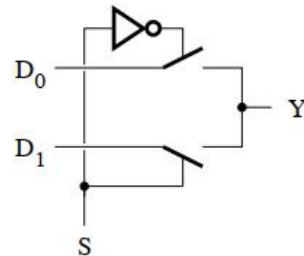
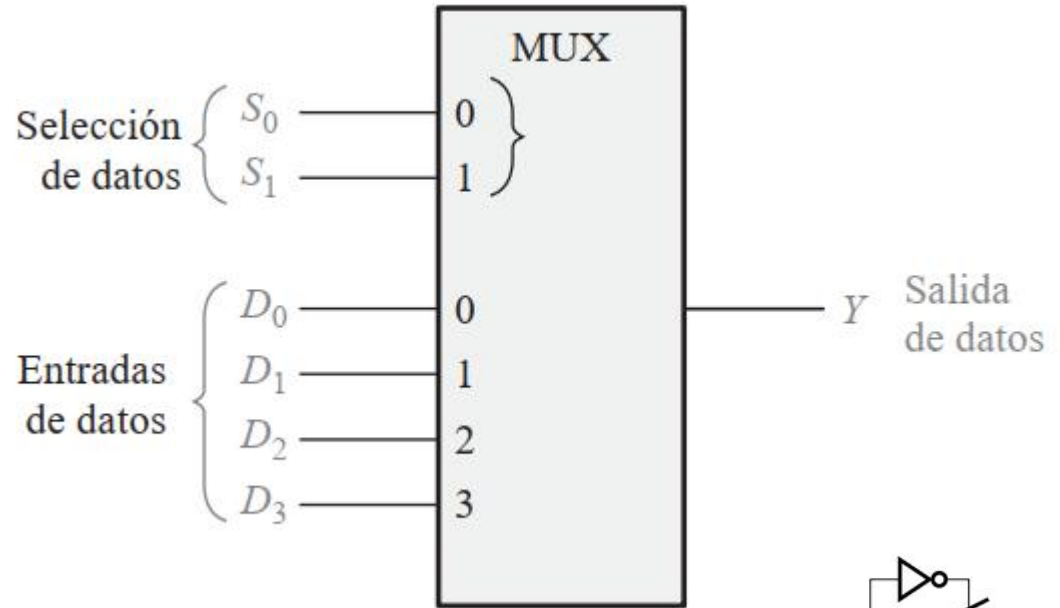
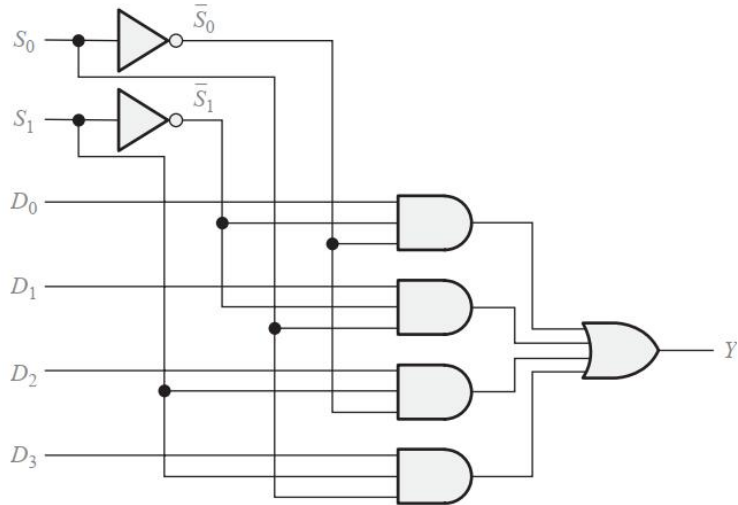


Multiplexor

• Función de “selector de datos”. Permite poner en la salida los datos procedentes de una (o varias) vías de entrada que se seleccionan mediante un control.

• Ejemplo para selector de 1 salida / 4 entradas

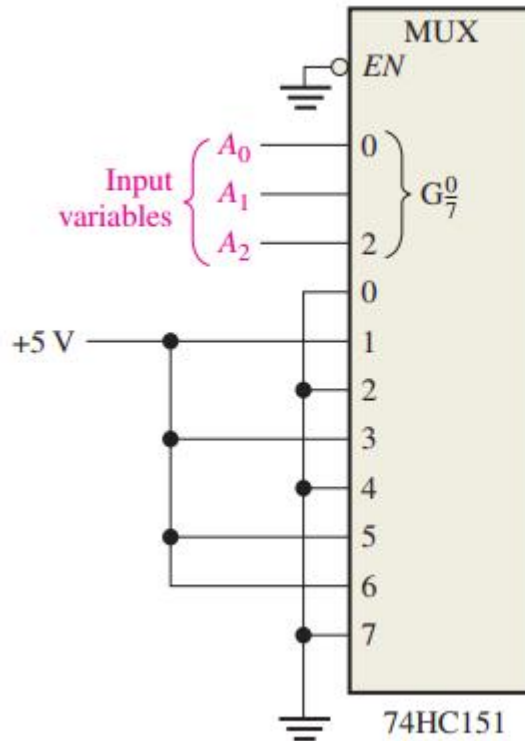
$$Y = D_0 \bar{S}_1 \bar{S}_0 + D_1 \bar{S}_1 S_0 + D_2 S_1 \bar{S}_0 + D_3 S_1 S_0$$



Aplicación de Multiplexores

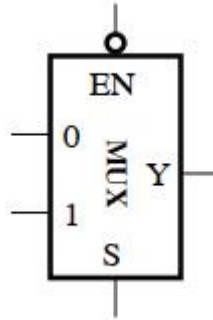
•App1 : Implementar funciones lógicas con multiplexores

Entradas			Salida
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



$$Y = \bar{A}_2 \bar{A}_1 A_0 + \bar{A}_2 A_1 A_0 + A_2 \bar{A}_1 A_0 + A_2 A_1 \bar{A}_0$$

EN	S	Y
1	x	0
0	0	D_0
0	1	D_1

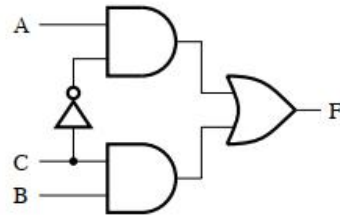


Consideraciones prácticas

- “Timing” de los circuitos combinacionales.
- Un diagrama de tiempos (cronograma) de un circuito nos describe cómo van cambiando los valores de las variables dentro de un circuito a lo largo del tiempo.
- En un cronograma podemos tomar en consideración los tiempos de subida y bajada de las señales, tiempos de propagación y demás consideraciones que nos permiten saber si algún aspecto práctico invalida nuestro diseño “idealizado”.
 - Hoy en día tienen gran importancia los simuladores.

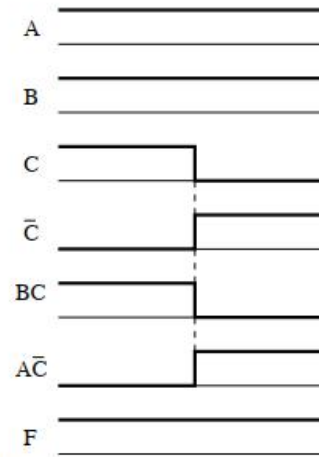
• Ejemplo de **static hazard**

$$F = A \cdot \bar{C} + B \cdot C$$



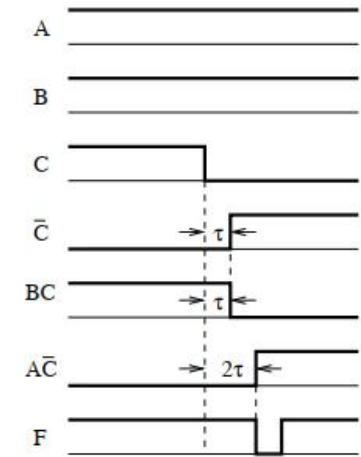
(a)

Caso ideal



(b)

Considerando retardos



(c)

Consideraciones prácticas

• Ejemplo de **static Hazard**

• ¿Qué podemos hacer? Consideremos el mapa de Karnaugh y una versión extendida:

$$F = A \cdot \overline{C} + B \cdot C$$

AB \ C		A			
		00	01	11	10
C	0	0	0	1	1
	1	0	1	1	0

$$F = A \cdot \overline{C} + B \cdot C + A \cdot B$$

El método consiste en identificar términos adyacentes en el diagrama de Karnaugh y añadir términos redundantes que vayan a garantizar que no parecen esos resultados espúreos no deseados.

Aritmética binaria. Operar. Sumar

$$0 + 0 = 0$$

$$0 + 1 = 1$$

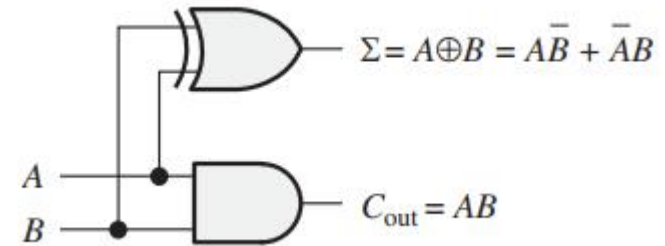
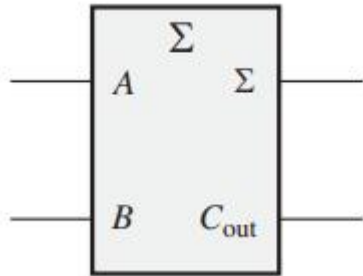
$$1 + 0 = 1$$

$$1 + 1 = 10$$

Concepto de acarreo

Aritmética binaria. Funciones. Semi-sumador (*half-adder*)

$0 + 0 = 0$	Suma 0 con acarreo 0
$0 + 1 = 1$	Suma 1 con acarreo 0
$1 + 0 = 1$	Suma 1 con acarreo 0
$1 + 1 = 10$	Suma 0 con acarreo 1

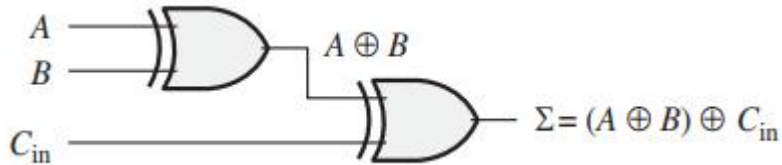
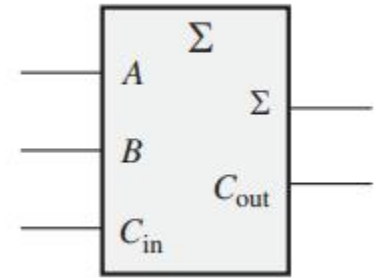


Aritmética binaria. Sumador completo (*Full-adder*)

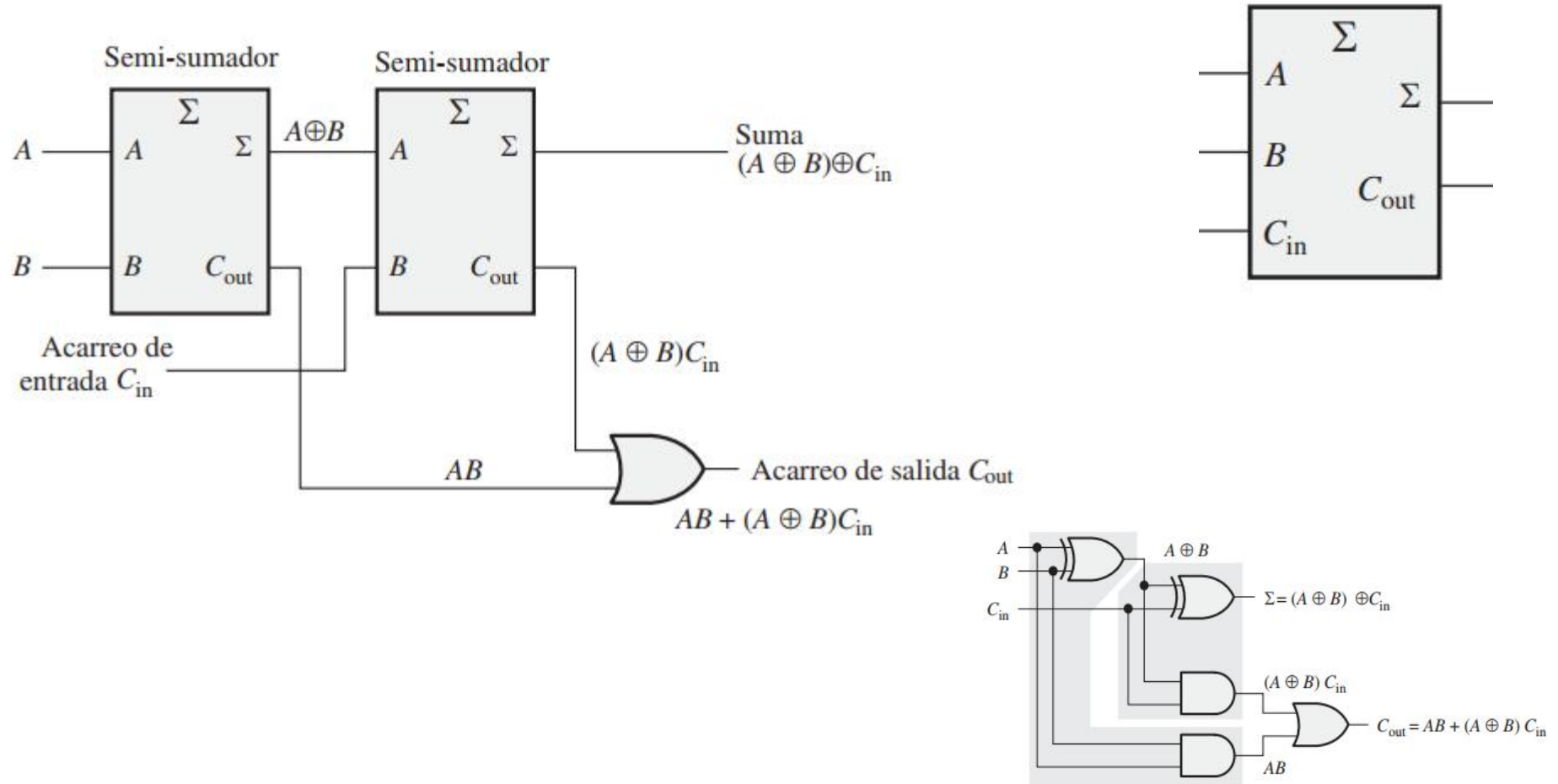
Se trata de introducir un acarreo de entrada

A	B	C_{in}	C_{OUT}	Σ
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

$$C_{out} = AB + (A \oplus B) C_{in}$$



Aritmética binaria básica. Sumador completo 1 bit (*full adder*)



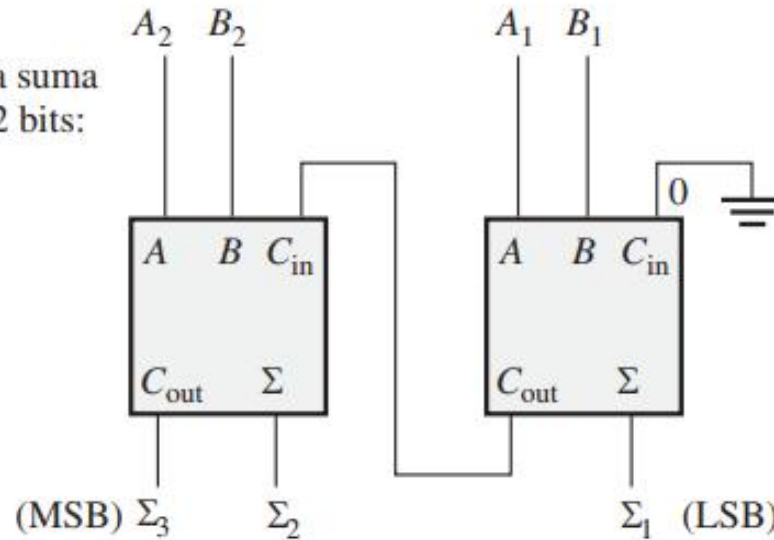
Sumador paralelo

• ¿Cómo sumar números de más bits?

$$\begin{array}{r} 1 \\ 11 \\ +01 \\ \hline 100 \end{array}$$

Formato general de la suma de dos números de 2 bits:

$$\begin{array}{r} A_2A_1 \\ + B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



La desventaja de este modo de propagar el acarreo es que no es la solución más rápida cuando tenemos múltiples niveles de sumador.

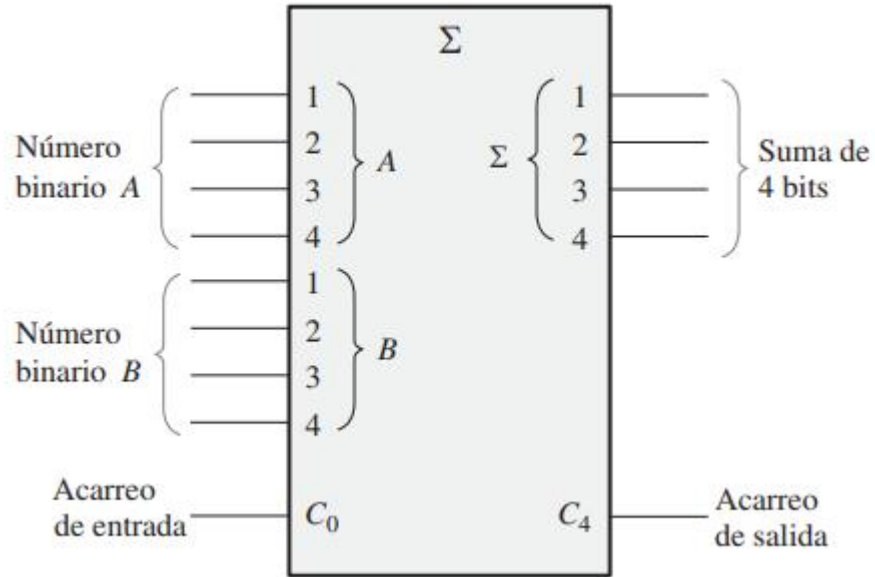
Existe una optimización a este circuito que es el **acarreo anticipado**

Sumador paralelo de 4 bits (nibble)

•LSB y MSB

•Existen chips comerciales co
esta operación

•Ej: 74HC283 (CMOS)



Ejemplo de uso de estos circuitos MSI

.Construir el conversor de BCD (2 digitos – 8bit) a binario estándar

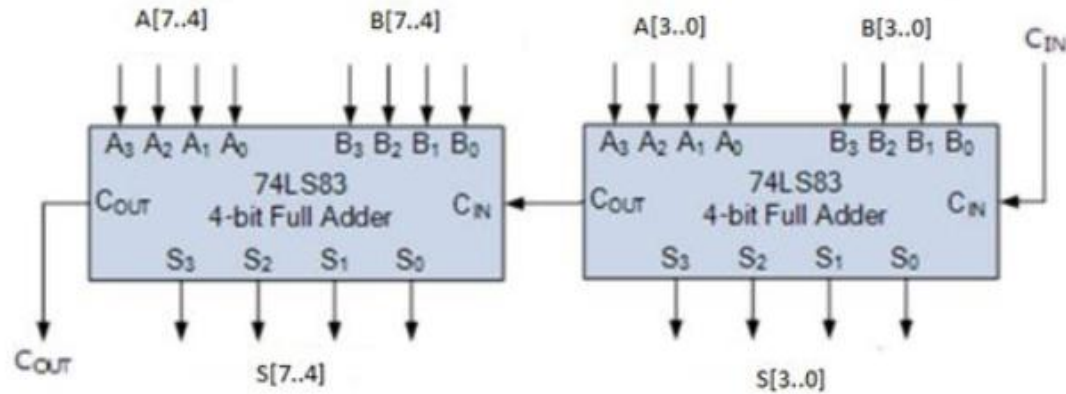
BCD= *binary coded decimal* (decimal codificado en binario)

Conversor BCD a binario

Bit BCD	Peso BCD	(MSB)	Representación binaria					(LSB)
		64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

.Cada bit de la entrada por tanto tiene una aportación al nro final que tengo que sumar

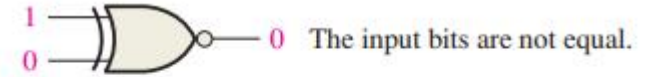
Conversor BCD a binario



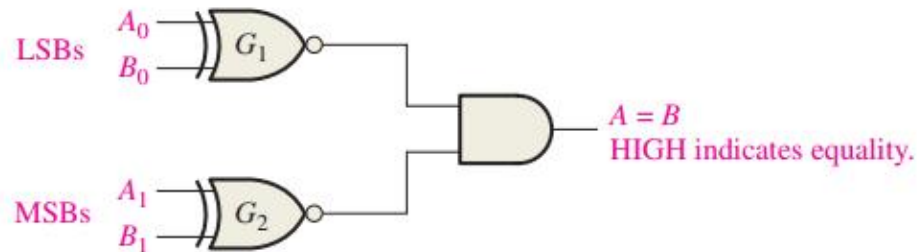
Por tanto la conversión puede realizarse por sucesivas sumas con un circuito como el que se muestra aquí, y utilizando la tabla de pesos de la diapositiva anterior.

Más bloques funcionales: COMPARADOR de magnitud

• Comparador básico de 1-bit es un X-NOR para obtener **igualdad/desigualdad**

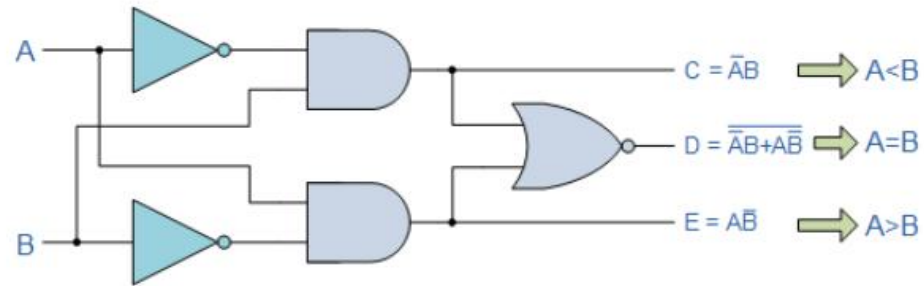


• Para comparar 2 bits



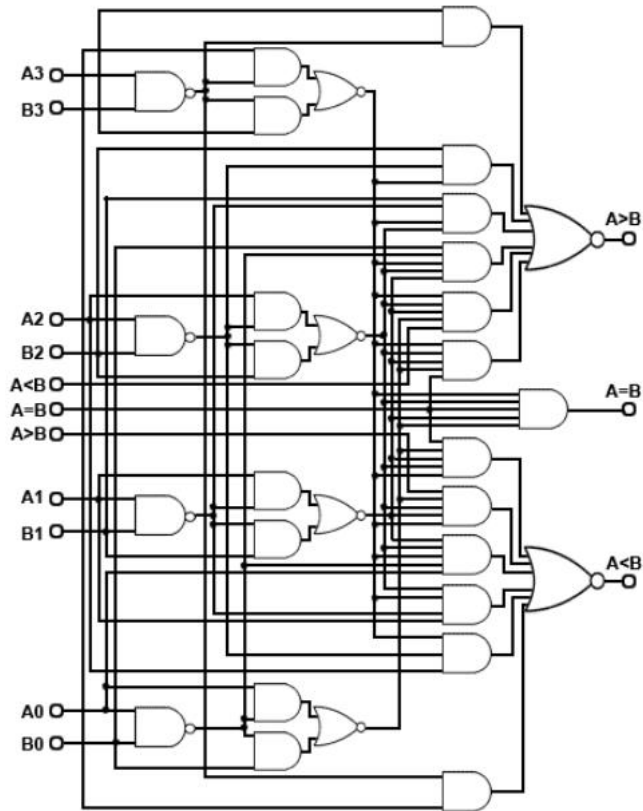
Más bloques funcionales: COMPARADOR de magnitud

Si quiero comparar también la magnitud:



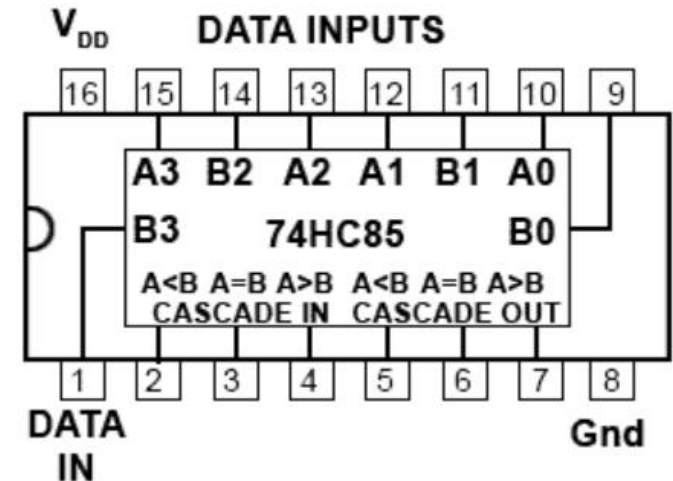
Más bloques funcionales: COMPARADOR de magnitud

• Para comparar nros 3 bits

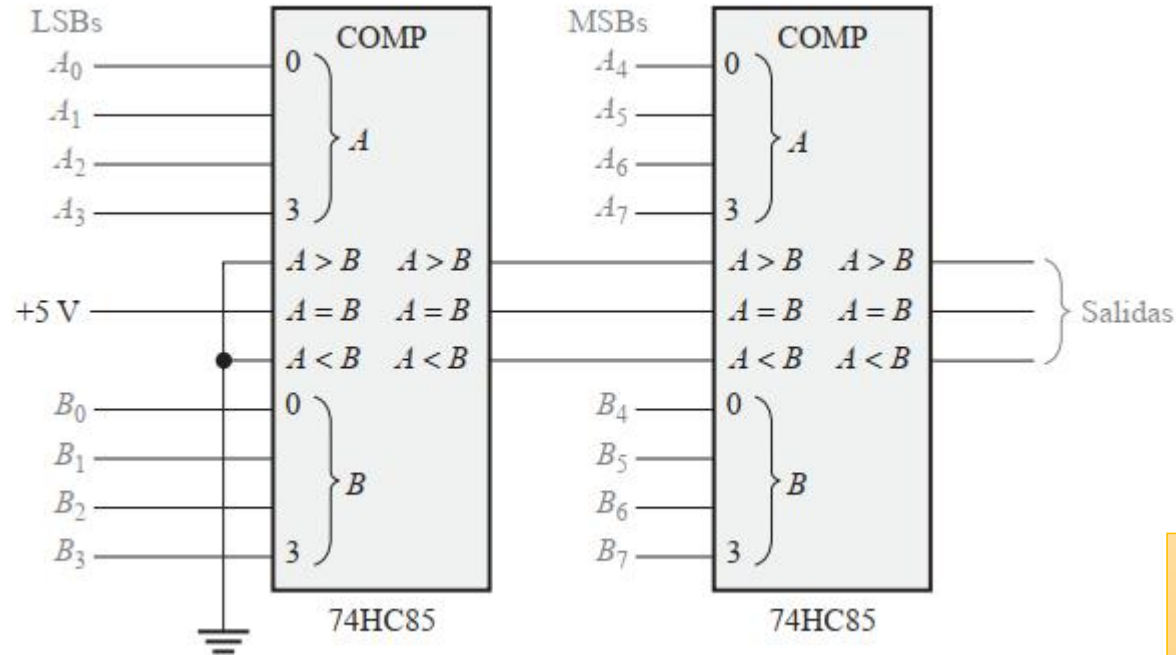


• Para comparar nros 4 bits

MSI: Medium scale integrated devices



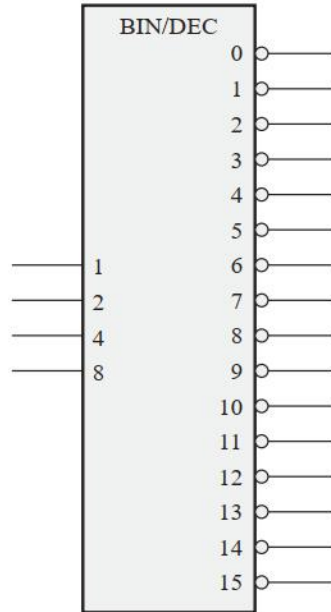
COMPARADOR : conexión en cascada



Los comparadores se usan intensivamente, por ejemplo, en el acceso a las memorias *cache*

Decodificadores

- Ante la presencia de una combinación de bits a la entrada señalar en alguna de las líneas de salida la presencia de un cierto código.
- En el caso general n-bits a la entrada requieren 2^n líneas para identificar todos los códigos posibles.
- Ejemplo: decodificador 1 de 16 (**74HC154**)



INPUTS					OUTPUTS															
\overline{E}	A3	A2	A1	A0	O0	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15
1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

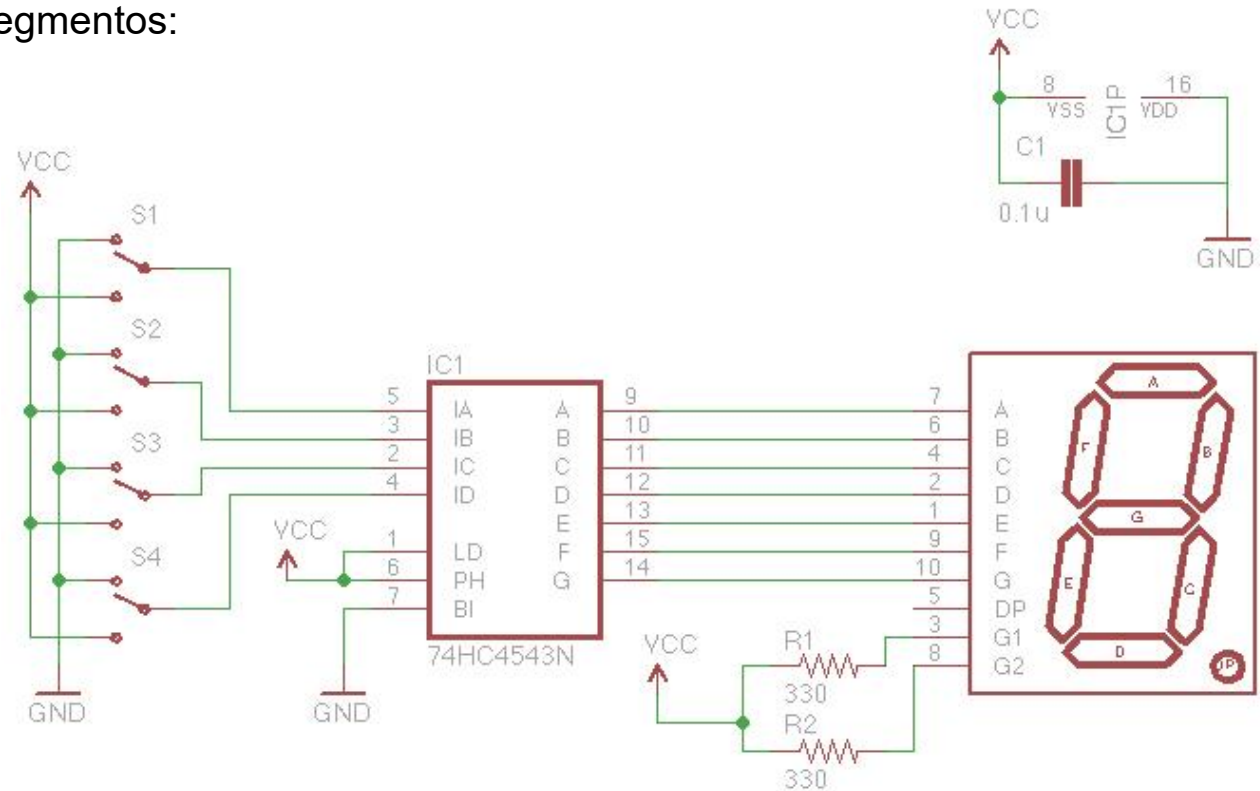
Cada línea es un circuito de las 4 variables de entrada

Decodificadores (II)

.Usos:

-Monitores de datos, displays, impresoras, decodificadores de dirección en las memorias, etc.

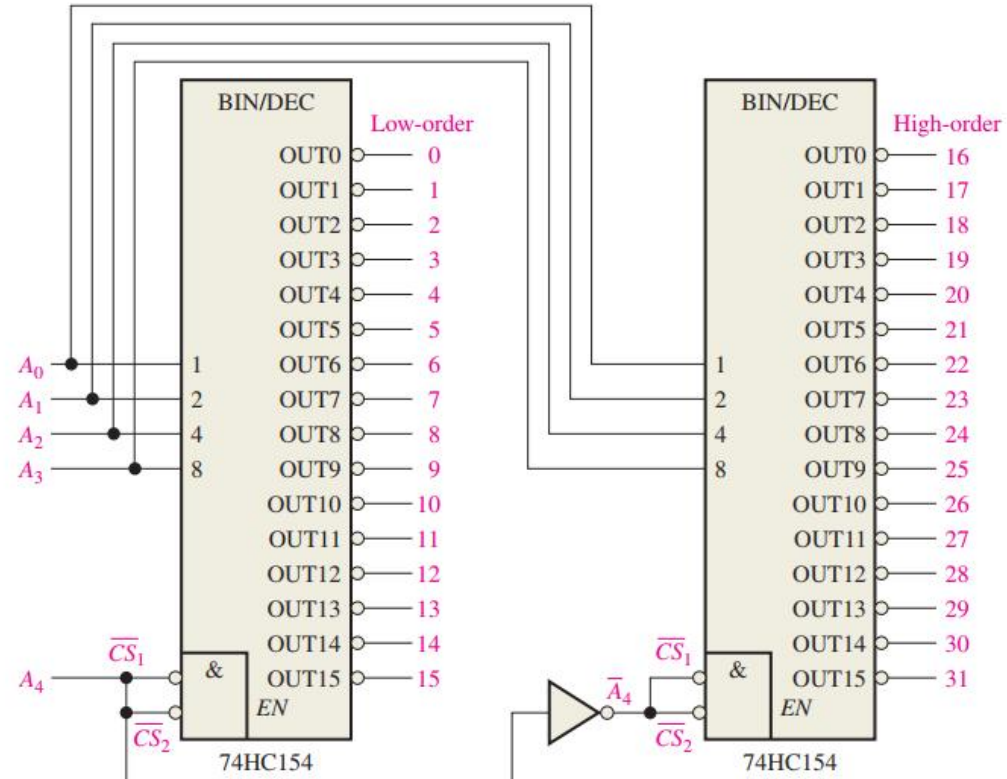
.Otro ejemplo, decodificador 7 segmentos:



Decodificadores en cascada

- Ejemplo: decodificador de 5 bits
- Cuando el número es >15 el $A_4 = \text{ON}$
- Se desactiva en este caso el primer deco y se activa el segundo.

En este decodificador el *Enable* se consigue con CS1 y CS2 a low



Después de la suma : ¿ RESTAR ?

Hemos visto las bases de la aritmética y algunos bloques de circuito útiles para implementar diversas funciones.

Necesitamos resolver un problema más:

Ahora necesitamos extender la aritmética para poder manejar números negativos.

Sistemas de representacion de Negativos en binario (I)

-Signo – magnitud

- El MSB es el signo (0 si es positivo o 1 si es negativo)
- El resto de los bits son la magnitud del número a representar, en binario natural

$$+25_{10} = 011001_{SM}$$

$$-25_{10} = 111001_{SM}$$

-Complemento a 1

- Si el número es positivo:
 - .El MSB es un 0 (signo)
 - .El resto de los bits son la magnitud en binario natural
- Si el número es negativo:
 - .El MSB es un 1 (signo)
 - .El resto de los bits son el complemento (a 1) de la magnitud

$$+25_{10} = 011001_{Ca1}$$

$$-25_{10} = 100110_{Ca1}$$

Ca1 es como restar de “todo 1’s”:

$$\text{Ej, } Ca1(0110) = 1001$$

$$1111-0110 = 1001$$

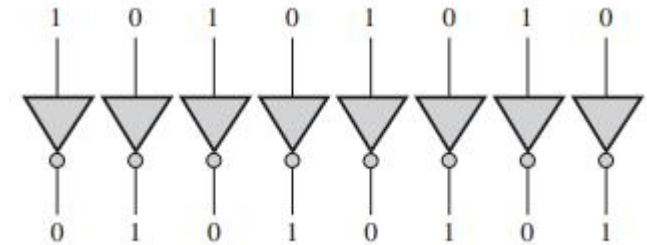
Sistemas de representacion de Negativos en binario (II)

•El más común: complemento a 2.

-complementar a 1 todos los bits y añadir 1.

•Ejemplo 1:

10110010	Número binario
01001101	Complemento a 1
+ 1	Sumar 1
<hr/> 01001110	Complemento a 2



$$+25_{10} = 011001_{Ca2}$$

$$-25_{10} = 100111_{Ca2}$$

•Método rápido:

Complemento a 1 de los bits originales →	10111000	Número binario
	01001000	Complemento a 2

↑ Estos bits no varían.

$$Ca2(A) = 2^n - A$$

$$(2^n - A = 2^n - 1 + 1 - A = 11\dots 11 - A + 1 = \overline{A} + 1)$$

Negativos: Complemento a 2

•Al introducir el signo reducimos el rango representable:

-8 bits sin signo: $2^8 = 256$

-16 bits sin signo: $2^{16} = 65.536$

-8 bits CON signo: -128 a + 127

-16 bits CON signo: -32.768 a +32.767

Complemento a 2. Comparación

- No confundir la operación de complementar a 2 con la representación de un nro. en complemento a 2!!
- Ojo al extender el nro de bits: los negativos extienden '1's
- Comparamos con las otras representaciones:

Codificación			SM	Ca1	Ca2
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	2	2	2
0	1	1	3	3	3
1	0	0	-0	-3	-4
1	0	1	-1	-2	-3
1	1	0	-2	-1	-2
1	1	1	-3	-0	-1

Complemento a 2. Propiedades.

$$\text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = A_{\text{Ca2}}$$

$$\text{Dem: } \text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = 2^n - (\text{Ca2}(A_{\text{Ca2}})) = 2^n - (2^n - A_{\text{Ca2}}) = A_{\text{Ca2}}$$

$$-A_{\text{Ca2}} = \text{Ca2}(A_{\text{Ca2}})$$

Dem: Si A_{Ca2} es positivo, entonces por la propia definición de representación en Ca2

Si A_{Ca2} es negativo, entonces $-A_{\text{Ca2}}$ se obtiene haciendo la operación inversa del Ca2, pero como $\text{Ca2}(\text{Ca2}(A_{\text{Ca2}})) = A_{\text{Ca2}}$ se tiene que $-A_{\text{Ca2}} = \text{Ca2}(A_{\text{Ca2}})$

Partiendo de un número positivo: 00001001 (+9)

Realizando la operación de Ca2: 11110111 (-9)

Realizando la operación de Ca2: 00001001 (+9)

Suma binaria

- Las sumas de números naturales en binario se realizan igual que en decimal:

$$\begin{array}{r} 1001 \text{ (9)} \\ + 1101 \text{ (13)} \\ \hline 10110 \text{ (22)} \end{array}$$

- Utilizando la representación de los números en Ca_2 , el método es válido también para números con signo, si se siguen las siguientes reglas.
 - Operandos con el mismo número de bits
 - Se descarta el acarreo final
 - Si los dos operandos tienen el mismo signo, y el resultado de la operación tiene signo diferente el resultado no es válido. Se dice que hay desbordamiento ("**overflow**"):
 - Esto sucede porque haría falta un bit adicional para poder representar el resultado.

No hacen
falta
restadores

Conversión Ca2 a decimal

• Sea el binario $b_{n-1}b_{n-2}\dots b_1b_0$

Equivalente decimal:

$$-b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i$$

• Ejemplo:

1 1 1 1 0 0 1 0

$$\begin{aligned} &= -b_7 \times 2^7 + \sum_{i=0}^6 b_i \times 2^i \\ &= -1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 \\ &= -128 + 64 + 32 + 16 + 2 = -14 \end{aligned}$$

• Alternativamente:

- Hacer el complemento a 2: “pasar a positivo”

- Después poner el signo

•

Suma binaria

- Dos números positivos:
(+9) + (+4)

$$\begin{array}{r} 0\ 1001_{\text{Ca2}}\ (+9) \\ +\ 0\ 0100_{\text{Ca2}}\ (+4) \\ \hline 0\ 1101_{\text{Ca2}}\ (+13) \end{array}$$

- Número positivo grande y número negativo pequeño:
(+9) + (-4)

$$\begin{array}{r} 0\ 1001_{\text{Ca2}}\ (+9) \\ +\ 1\ 1100_{\text{Ca2}}\ (-4) \\ \hline 1\ 0\ 0101_{\text{Ca2}}\ (+5) \end{array}$$

- Números iguales de signo contrario:
(+9) + (-9)

$$\begin{array}{r} 0\ 1001_{\text{Ca2}}\ (+9) \\ +\ 1\ 0111_{\text{Ca2}}\ (-9) \\ \hline 1\ 0\ 0000_{\text{Ca2}}\ (0) \end{array}$$

- Dos números negativos:
(-9) + (-4)

$$\begin{array}{r} 1\ 0111_{\text{Ca2}}\ (-9) \\ +\ 1\ 1100_{\text{Ca2}}\ (-4) \\ \hline 1\ 1\ 0011_{\text{Ca2}}\ (-13) \end{array}$$

- Número positivo pequeño y número negativo grande:
(-9) + (+4)

$$\begin{array}{r} 1\ 0111_{\text{Ca2}}\ (-9) \\ +\ 0\ 0100_{\text{Ca2}}\ (+4) \\ \hline 1\ 1011_{\text{Ca2}}\ (-5) \end{array}$$

- Overflow:
(+1) + (+1)

$$\begin{array}{r} 0\ 1_{\text{Ca2}}\ (+1) \\ +\ 0\ 1_{\text{Ca2}}\ (+1) \\ \hline 1\ 0_{\text{Ca2}}\ (-2) \end{array}$$

Control del desbordamiento (*overflow*)

.En binario natural sin signo simplemente tenía desbordamiento tras una operación si el *carry* del bit MSB (más significativo) era '1'

.En Ca2 es diferente, en la regla de "resta" NO se considera el último acarreo explícitamente.

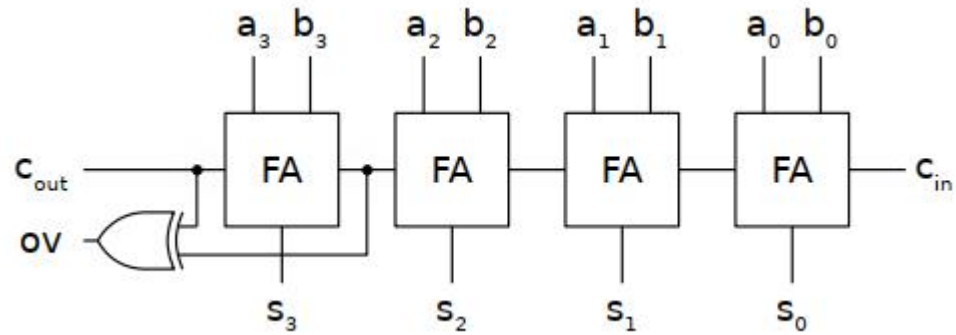
$$\begin{array}{r}
 \textcolor{red}{0} \\
 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 15 \\
 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ +15 \\
 \hline
 (0) \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ \underline{30} \ \text{OK} \\
 \textcolor{red}{1} \\
 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 127 \\
 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ +1 \\
 \hline
 (0) \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ \underline{-128} \ \text{overflow}
 \end{array}$$

$$\begin{array}{r}
 \textcolor{red}{1} \\
 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ -15 \\
 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ + -15 \\
 \hline
 (1) \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ \underline{-30} \ \text{OK} \\
 \textcolor{red}{0} \\
 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ -127 \\
 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ + -2 \\
 \hline
 (1) \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ \underline{127} \ \text{overflow}
 \end{array}$$

Se coteja el acarreo de entrada del MSB con el de salida : si son diferentes estoy en situación de desbordamiento (EXOR de los 2 acarreos)

Control del desbordamiento (*overflow*)

$$\begin{array}{r} \textcolor{red}{0} \text{ } \textcolor{red}{1} \\ 0 \dots \\ + 0 \dots \\ \hline 1 \dots \end{array}$$
$$\begin{array}{r} \textcolor{red}{1} \text{ } \textcolor{red}{0} \\ 1 \dots \\ + 1 \dots \\ \hline 0 \dots \end{array}$$



Números en coma flotante (nros reales)

•Opción 1 (representar números con coma decimal): Punto fijo

-Ej : 20 bits parte entera “.” 12 bits decimales

-Fácil de usar para las operaciones sencillas pero poco práctico

-Ej : $3.25_{10} = 11.01_2$

•Opción 2 (representar nros. enteros grandes o con parte fraccionaria) : Punto flotante

-***mantisa*** + ***exponente***. $N = M \times b^E$

-Ejemplos de coma flotante:

$$2448,35_{10} = 2,44835 * 10^3$$

$$0,0035_{10} = 3,5 * 10^{-3}$$

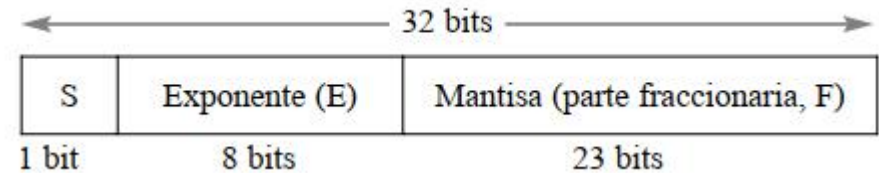
$$111,0110_2 = 1,11011_2 * 2^2$$

$$0,001101_2 = 1,101_2 * 2^{-3}$$

-Se representarán con un nro. fijo bits ***mantisa***, ***exponente*** y bit de ***signo***

Números en coma flotante (II)

.Standard IEEE 754 precisión simple (32 bits) :



.Normalización:

- E toma valores de 1 a 254
- E=255 es infinito (*overflow*) 0's en la mantisa
- 0 es E=0 M=0

$$N = (-1)^s * 2^{E-127} * 1.M$$

Números en coma flotante (nros reales)

$$N = (-1)^S * 2^{E-127} * 1.M$$

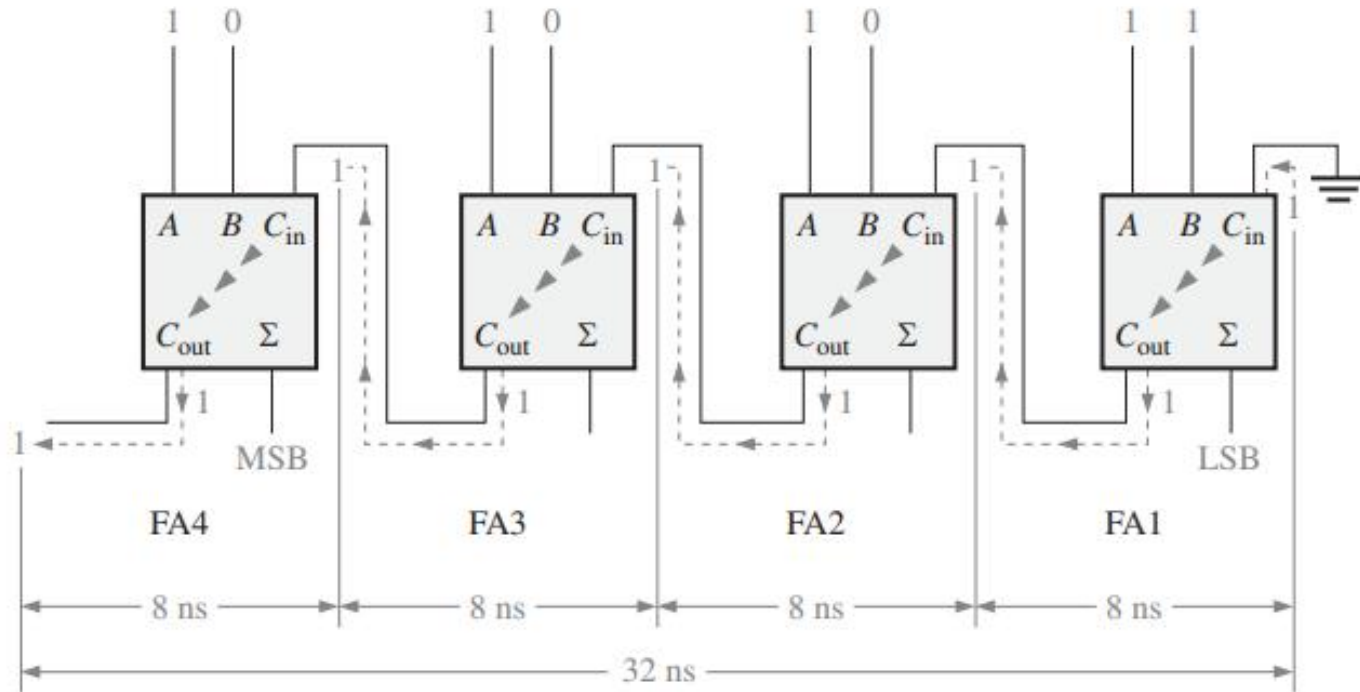
Distribución de los bits : 1 | 8 | 23 (bit de signo | Exponente | Mantisa)

Ejemplo: $3,248 \times 10^4$

Más sobre aritmética

Sumador paralelo. Propagación del acarreo.

• Problema de propagación: acarreo serie



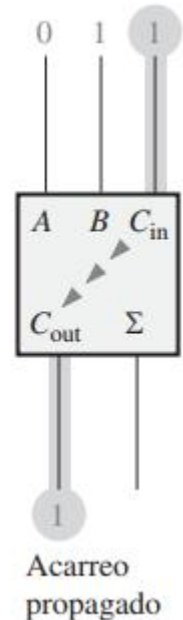
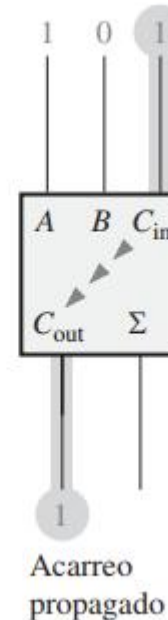
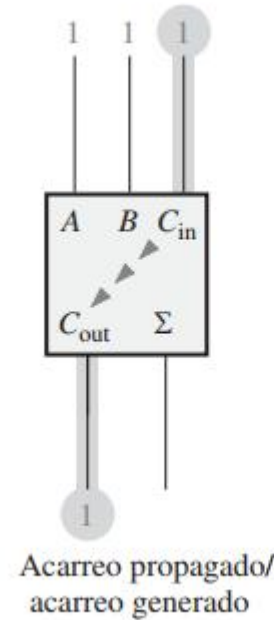
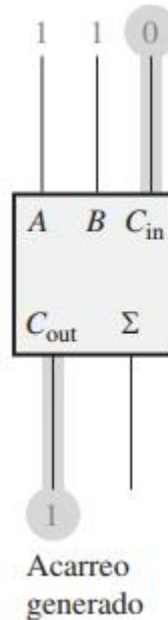
Sumador paralelo. Acarreo adelantado (look-ahead)

- Se trata de mejorar el rendimiento del sumador visto anteriormente
- Procedimiento: calcular “en paralelo” los bits de acarreo y el resultado de la suma.
- Se usan 2 ideas: generación de acarreo y propagación de acarreo

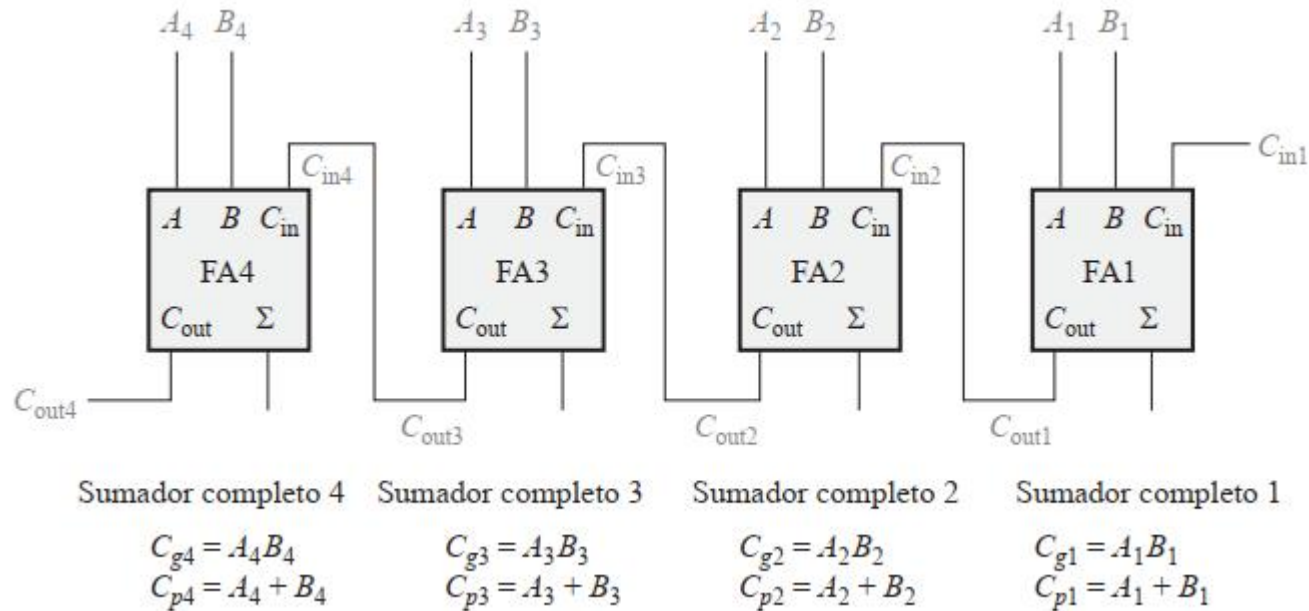
$$C_g = A.B$$

$$C_p = A+B$$

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Sumador paralelo. Acarreo adelantado (look-ahead)



Sumador completo 1:

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

Sumador completo 2:

$$C_{in2} = C_{out1}$$

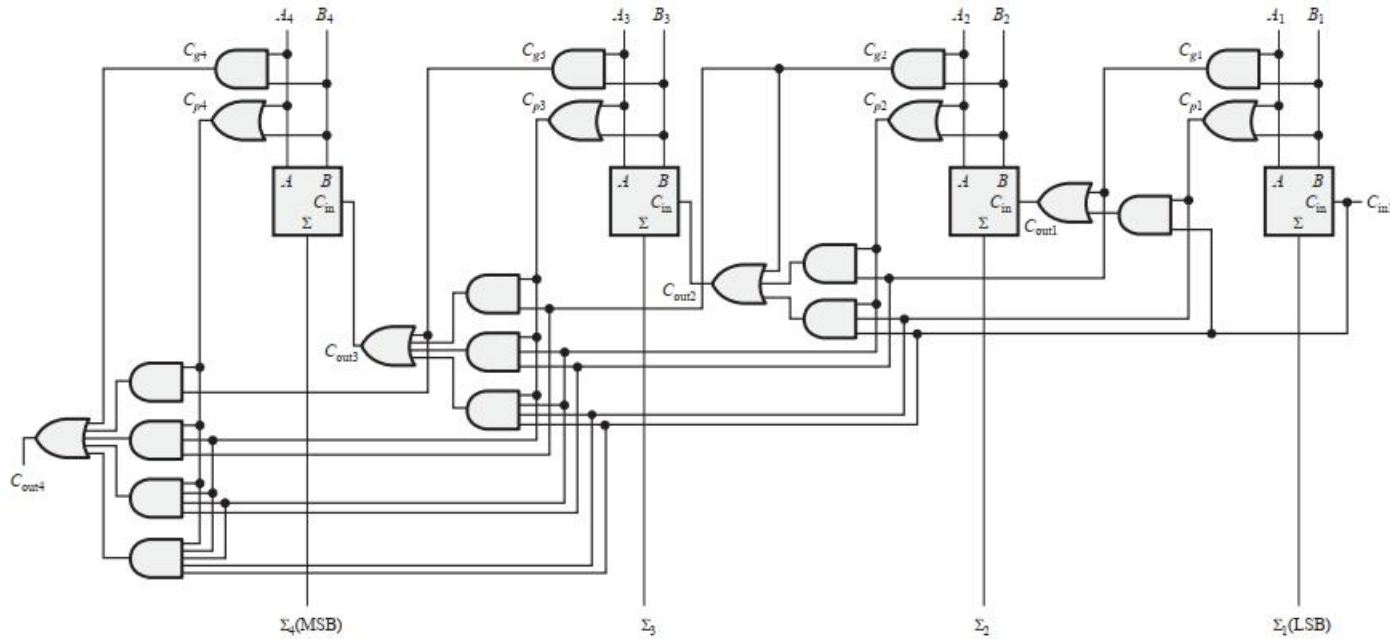
$$\begin{aligned}C_{out2} &= C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1} = C_{g2} + C_{p2}(C_{g1} + C_{p1}C_{in1}) \\&= C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}\end{aligned}$$

Sumador completo 3:

$$C_{in3} = C_{out2}$$

$$\begin{aligned}C_{out3} &= C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2} = C_{g3} + C_{p3}(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}) \\&= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}\end{aligned}$$

Sumador carry look-ahead 4 bit



•En ocasiones se combinarán sumadores multibit de acarreo adelantado y el resultado será una cosa “híbrida”.