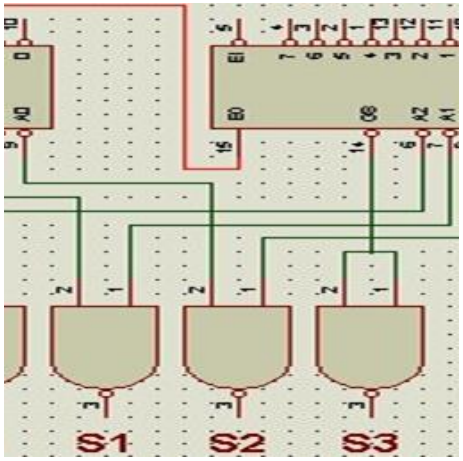
Tema 06

ARITMÉTICA BINARIA



EQUIPOS MICROPROGRAMABLES

Salus Nievas

1. INTRODUCCIÓN

La aritmética binaria es la que se utiliza en los sistemas digitales, y el mundo de los ordenadores.

En este tema estudiaremos entre otros la suma, resta, multiplicación y división binaria.

Además veremos como representar números positivos y negativos y como representar los números en valor absoluto y signo, complemento a 1 y en complemento a 2 C2.

2.1 SUMA BINARIA

Las reglas para efectuar la suma binaria son:

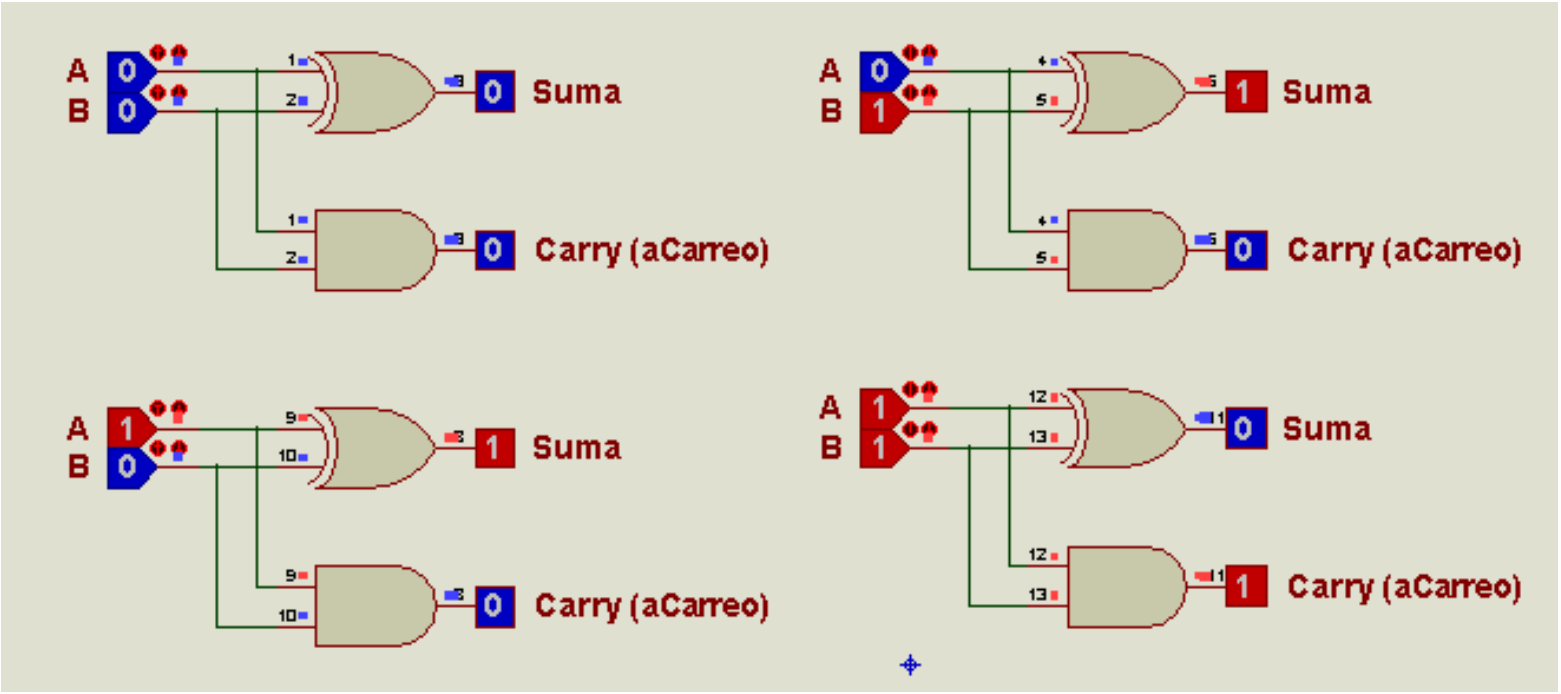
Operación; ;Resultado
0 + 0 = 0 ; Suma = 0 y me llevo 0; S=0 , Carry= 0
1 + 0 = 1 ;Suma = 1 y me llevo 0; S=1 , Carry= 0
0 + 1 = 1 ;Suma = 1 y me llevo 0; S=1 , Carry= 0
1 + 1 = 0 ;Suma = 0 y me llevo 1; S=0 , Carry= 1

Lo anterior se puede expresar mediante una tabla de verdad:

Operandos		resultados	
a	b	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.1 SUMA BINARIA

Si obtenemos las ecuaciones de la tabla de verdad e implementamos el circuito obtenemos:



Este circuito es conocido como sumador de 2 bits o semisumador

2.2 SUMADOR TOTAL

En la práctica, normalmente siempre sumaremos 2 bits + el acarreo procedente de la etapa anterior, y se debe generar la suma y el acarreo para la siguiente etapa. La tabla de verdad para este caso es:

C_{in}	A	B	S	C_{out}
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

Donde:
Cin (Carry Input) es el acarreo procedente de una etapa anterior.
A y B son los bits a sumar
S es la suma generada.
Cout es el acarreo generado para la siguiente etapa.

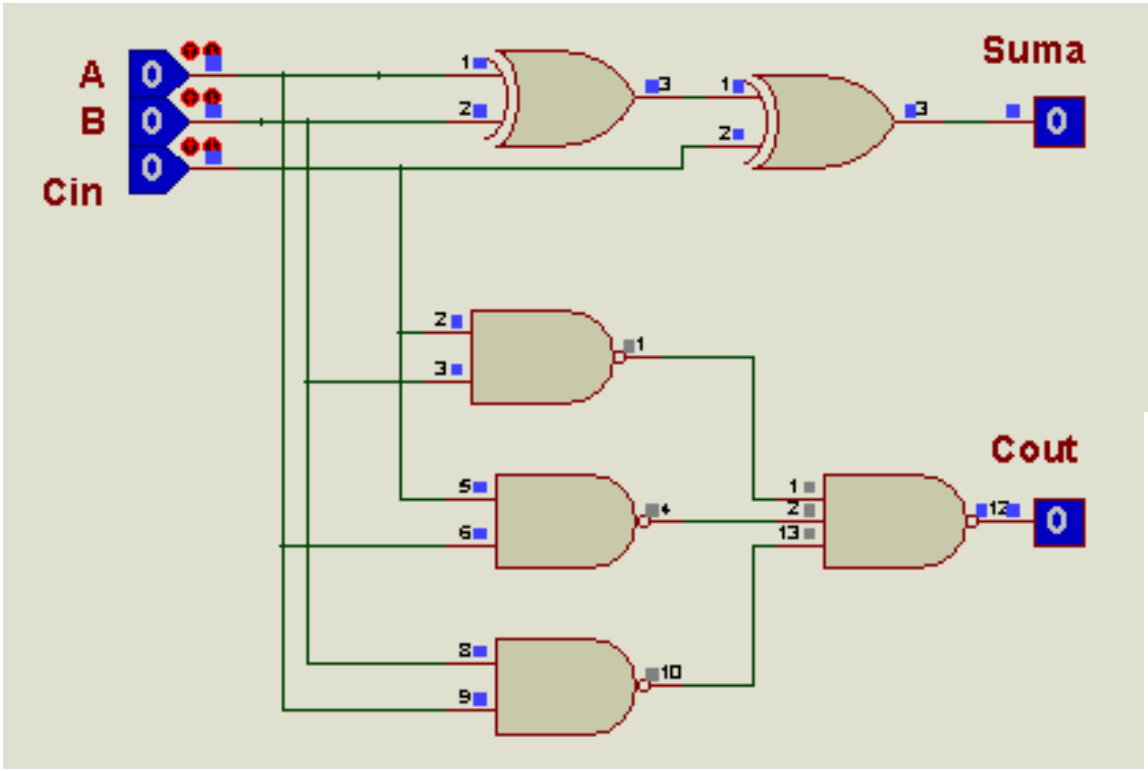
Simplificando las funciones S y Cout obtenemos:

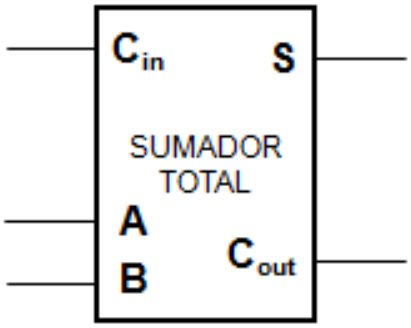
$$S = (\overline{A}\,\overline{B} + AB)C_{in} + (\overline{A}B + A\overline{B})\overline{C_{in}}$$

$$C_{out} = C_{in}A + C_{in}B + AB$$

2.2 SUMADOR TOTAL

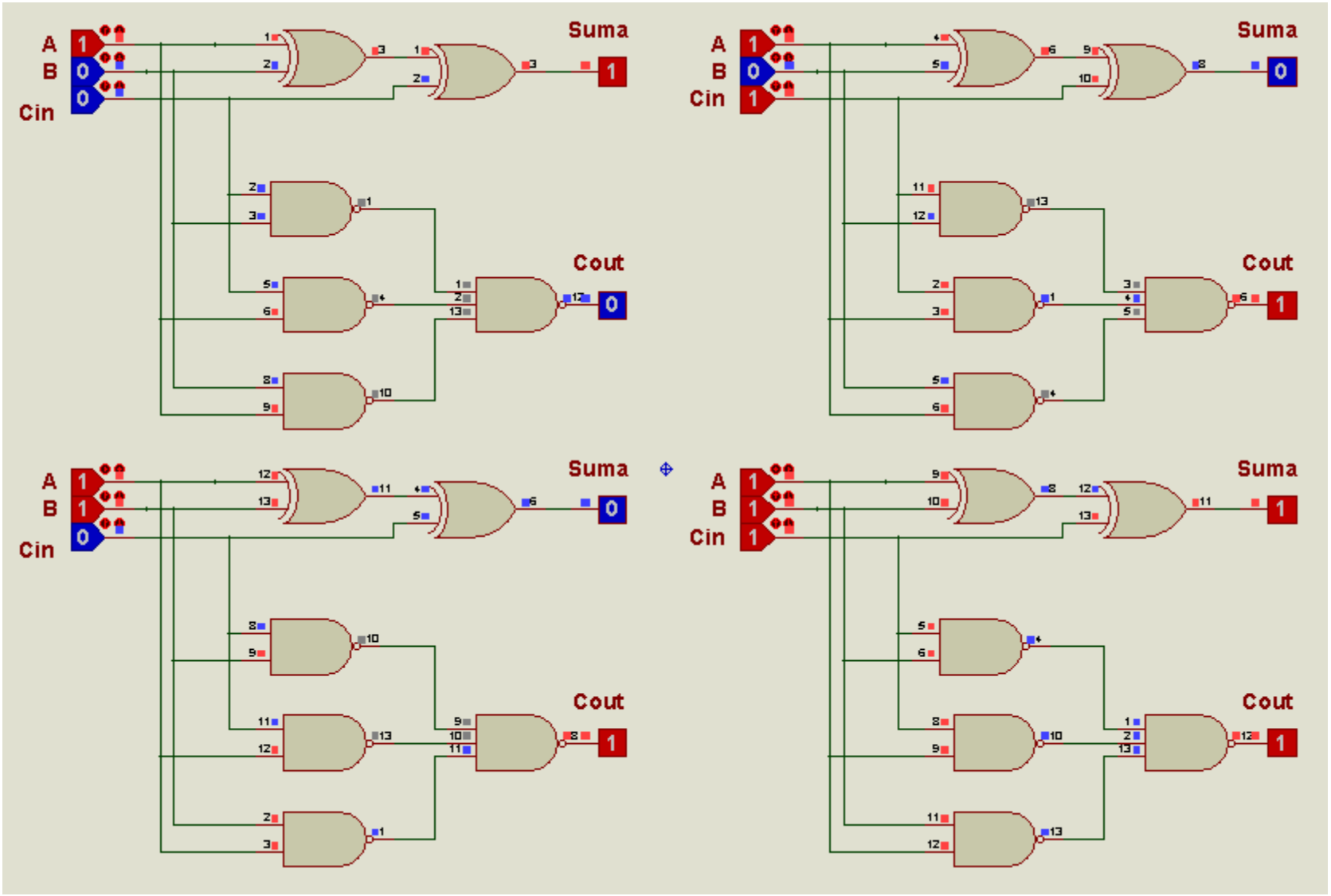
Implementando las ecuaciones anteriores obtenemos el siguiente circuito:





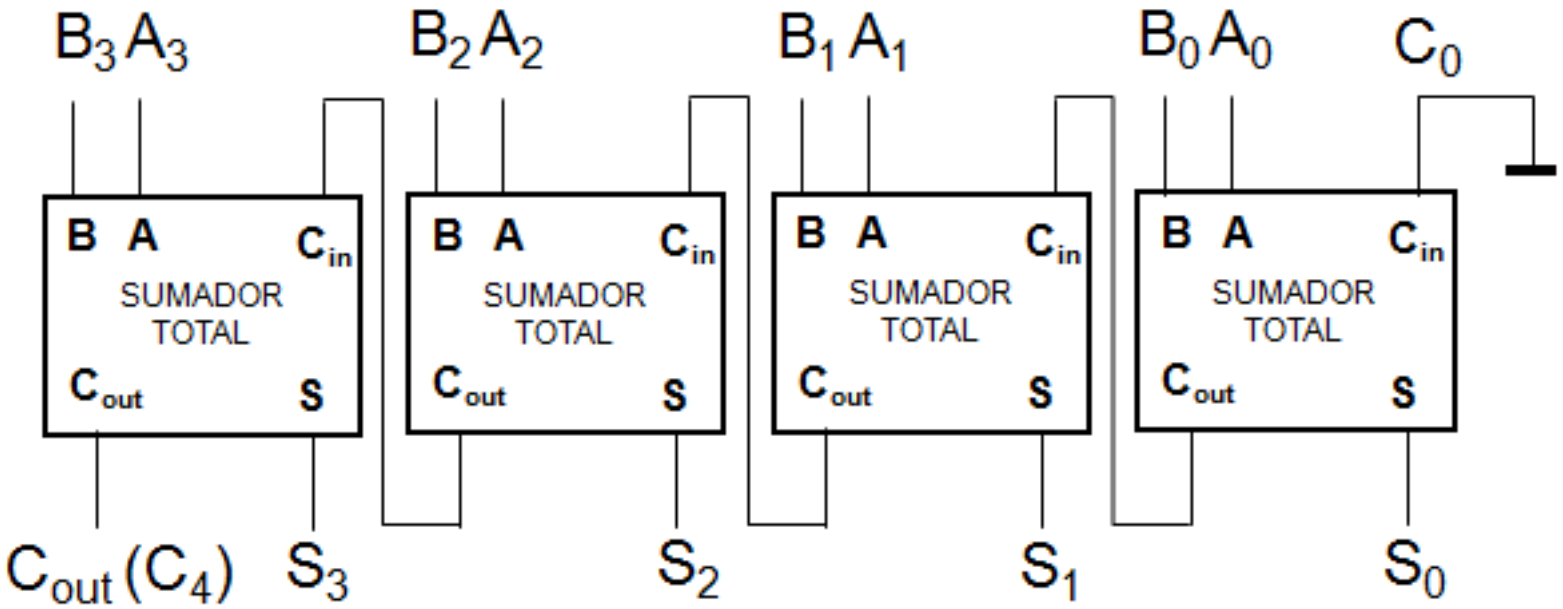
Este circuito recibe el nombre de sumador total

2.3 SUMADOR TOTAL

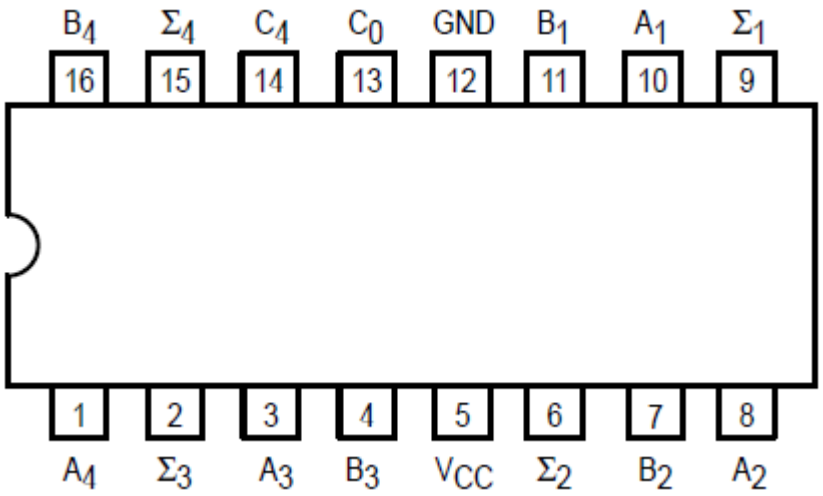


2.3 SUMA BINARIA DE 4 BITS

Enlazando bloques sumadores podemos obtener un sumador del número de bits deseado, observe que el carry final es el bit de mayor peso de la suma:



2.4 SUMADOR BINARI0 IC 74LS83

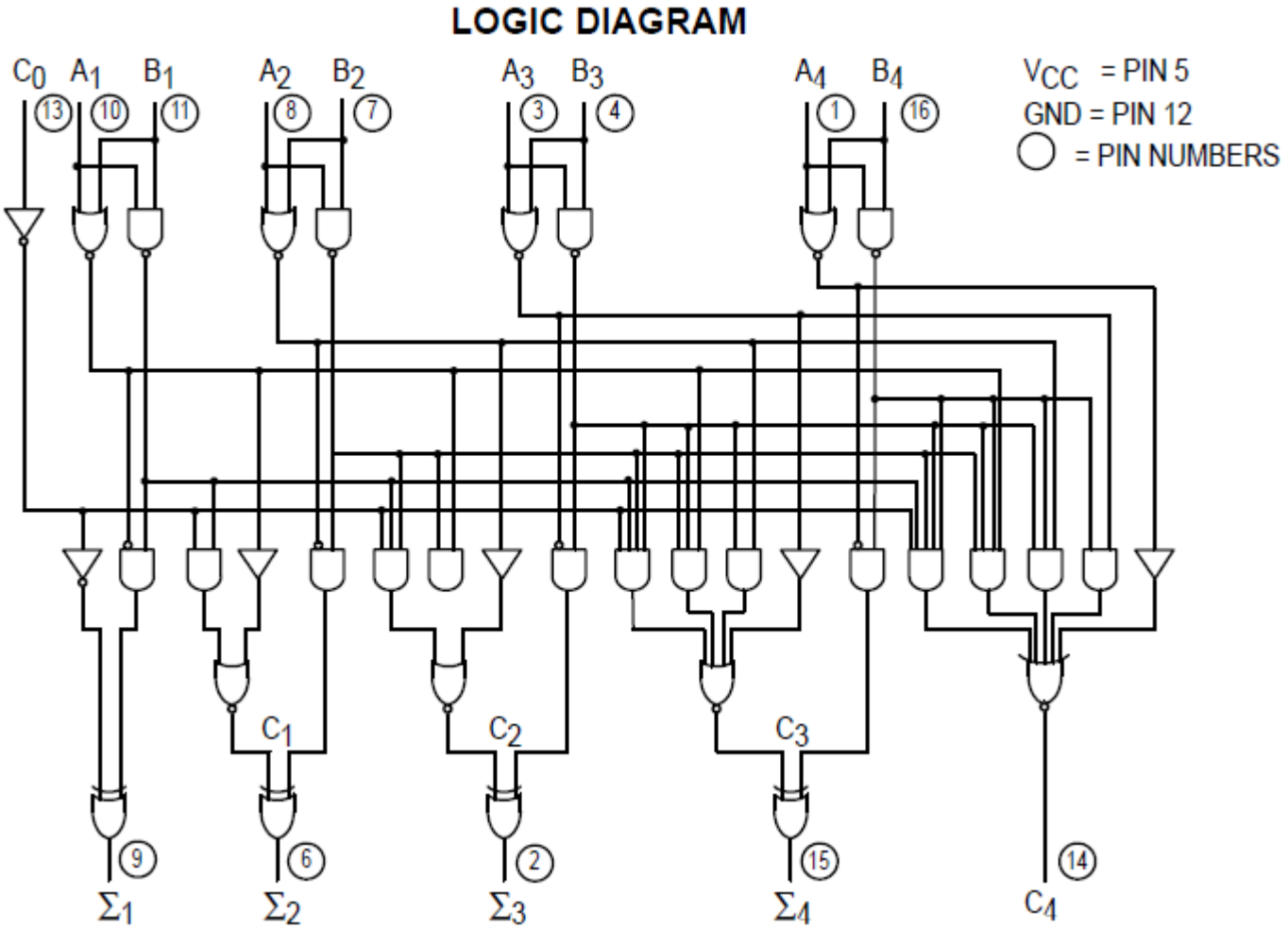


LOGIC SYMBOL

PIN NAMES

$A_1 - A_4$	Operand A Inputs
$B_1 - B_4$	Operand B Inputs
C_0	Carry Input
$\Sigma_1 - \Sigma_4$	Sum Outputs (Note b)
C_4	Carry Output (Note b)

2.4 SUMADOR BINARI0 IC 74LS83



** El acarreo se genera anticipadamente

2.4 SUMADOR BINARI0 DE 4 BITS IC 74LS83

Seguidamente se da la tabla de verdad de cada uno de los sumadores totales internos y un ejemplo.

FUNCTIONAL TRUTH TABLE

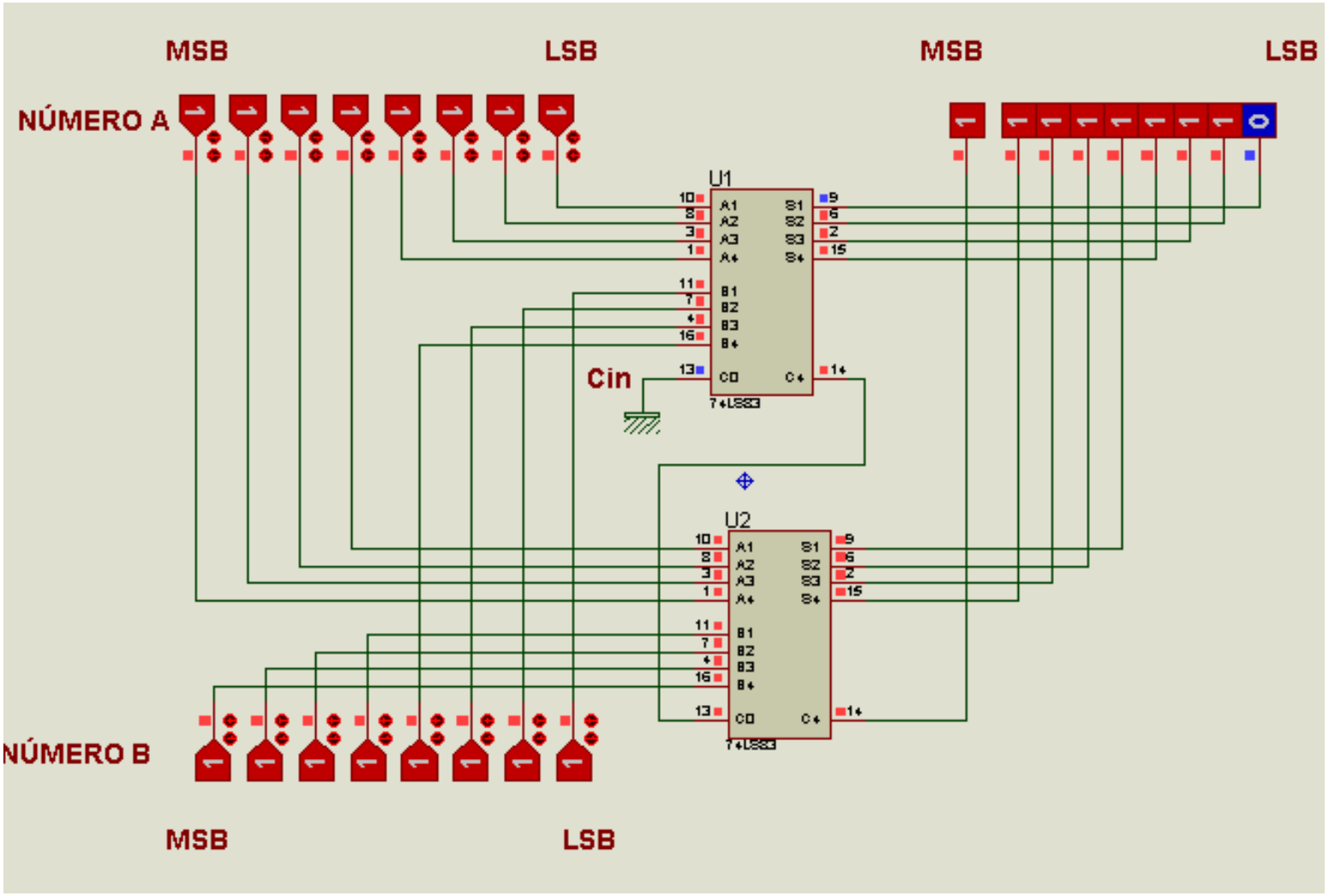
$C_{(n-1)}$	A_n	B_n	Σ_n	C_n
L	L	L	L	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	L	H
H	H	H	H	H

C_1 — C_3 are generated internally
C_0 — is an external input
C_4 — is an output generated internally

Example:

	C_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4	Σ_1	Σ_2	Σ_3	Σ_4	C_4	
Logic Levels	L	L	H	L	H	H	L	L	H	H	H	L	L	H	
Active HIGH	0	0	1	0	1	1	0	0	1	1	1	0	0	1	(10+9 = 19)

2.5 SUMA BINARIA DE 8 BITS CON 74LS83



3. RESTA BINARIA

Las reglas para efectuar la resta binaria son:

Operación; ;Resultado
0 - 0 = 0 ; Resta = 0 y debo 0; R=0 , Carry= 0
1 - 1 = 0 ; Resta = 0 y debo 0; R=0 , Carry= 0
1 - 0 = 1 ; Resta = 1 y debo 0; R=0 , Carry= 0
10 - 1 = 01 ; Resta = 1 y debo 1; R=1 , Carry= 1

En el último caso vemos como se ha generado un acarreo negativo que se traslada al siguiente bit.

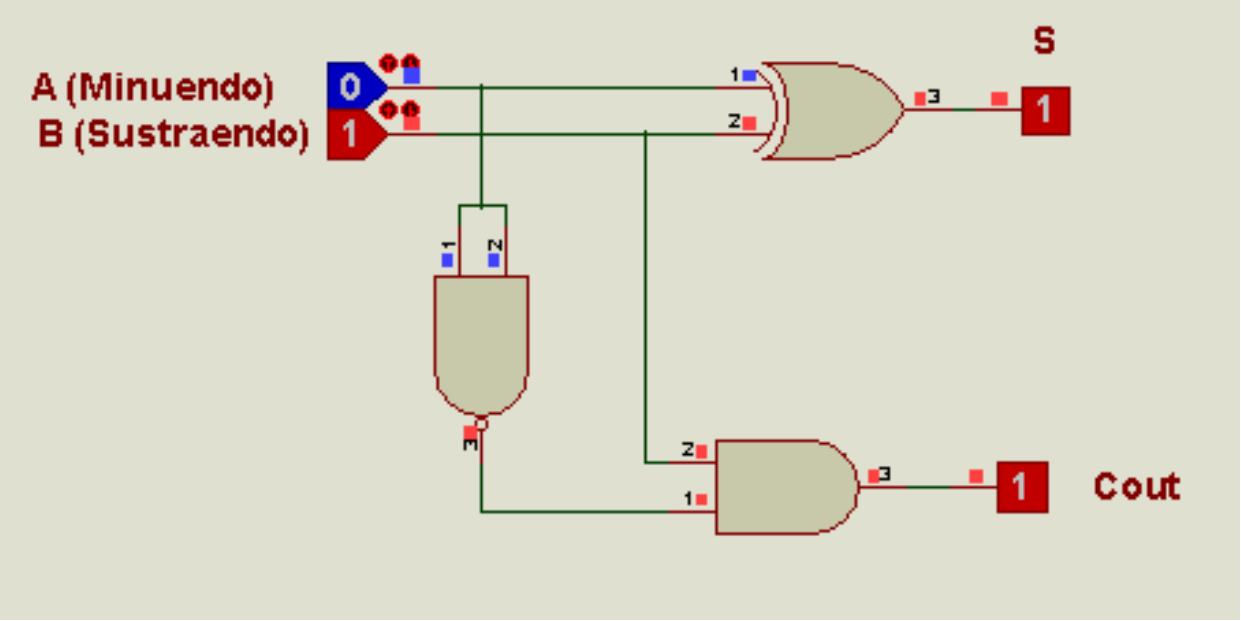
Tabla de verdad

A (minuendo)	B (sustraendo)	S	Cout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

3. RESTA BINARIA

Las ecuaciones que se obtienen son:

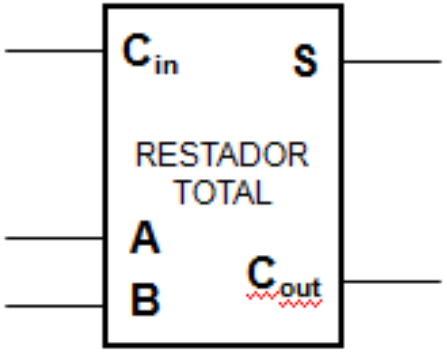
$$S = (\overline{A}B + A\overline{B}) \qquad Cout = \overline{A}B$$



3.1 RESTA BINARIA CON ACARREO DE ENTRADA

Tabla de verdad de un restador total:

A (M)	B (S)	C_{in}	B + C_{in}	S	C_{out}
0	0	0	0	0	0
0	1	0	1	1	1
1	0	0	0	1	0
1	1	0	1	0	0
0	0	1	1	1	1
0	1	1	0	0	1
1	0	1	1	0	0
1	1	1	0	1	1



M = Minuendo; S = sustraendo; Cin = debo de la etapa anterior
S = resta Cout = debo que se traslada a la etapa siguiente

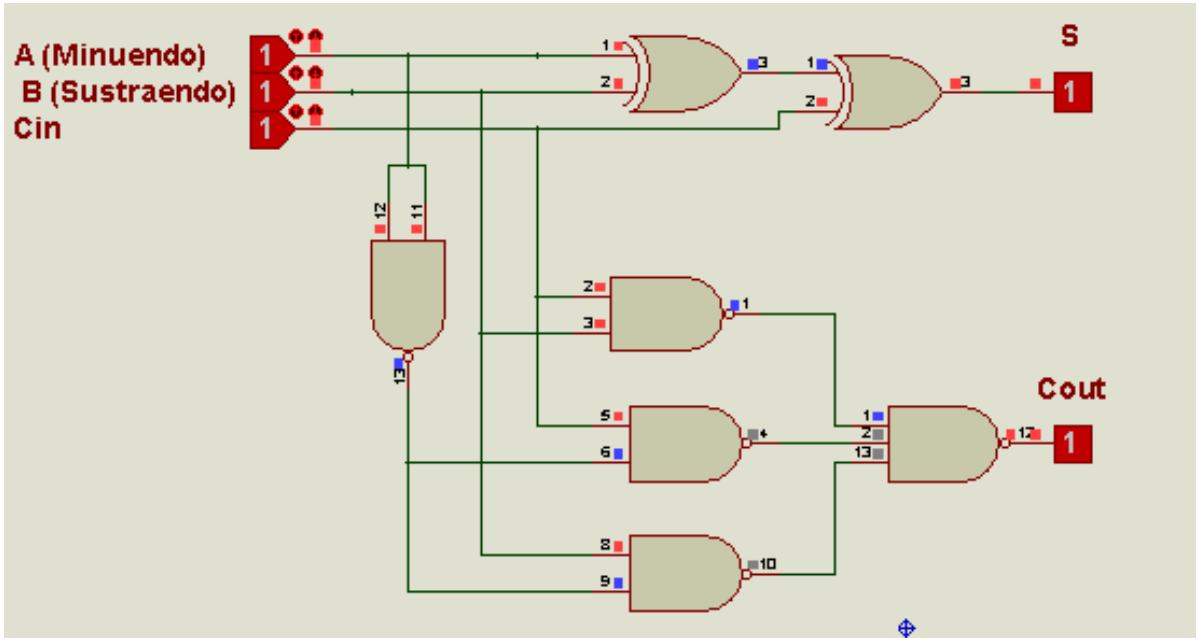
3.1 . RESTADOR TOTAL

Las ecuaciones que obtenemos al simplificar ambas funciones son:

$$S = (\overline{A}\overline{B} + AB)C_{in} + (\overline{A}B + A\overline{B})\overline{C_{in}}$$

$$C_{out} = C_{in}\overline{A} + C_{in}B + \overline{A}B$$

Y el circuito resultante es:



*** Este circuito no existe comercialmente.

4. REPRESENTACION DE LOS NÚMEROS CON SIGNO

En el caso de las restas, el caso más sencillo es aquel en el que el Minuendo es mayor o igual que el sustraendo, con lo que el resultado es un número positivo o cero.

Ahora bien puede suceder que el sustraendo, sea mayor que el minuendo, con lo que el resultado es un número negativo.

Ya que hasta el momento no hemos previsto como representar números negativos en binario, deberemos comenzar a estudiar como representar dicho números.

Existen tres formas de expresar los con signo: *signo-magnitud; complemento a 1 y complemento a 2*

4. REPRESENTACION DE LOS NÚMEROS CON SIGNO

En todos los números con signo el bit más a la izquierda es el signo, siguiéndose el siguiente criterio:

Todos los números que comienzan por 1 son negativos, mientras que los positivos comienzan todos por 0.

4.1. NÚMEROS CON SIGNO SIGNO Y MAGNITUD.

Cuando un número se expresa en este sistema, el primer bit es el signo, mientras que el resto de los bits son la magnitud (valor).

Ejemplos:

el número binario 0111 equivale a +7 en decimal

el número binario 1111 equivale a - 7 en decimal

Para transformar un número expresado en signo y magnitud en decimal se procede como sigue.

4.1. NÚMEROS CON SIGNO SIGNO Y MAGNITUD.

Se suman los pesos de las posiciones que tienen a 1 el bit correspondiente del número binario a convertir, excluyendo el bit de mayor peso (signo).

Al valor obtenido se le pone signo – si el MSB es 1, y + si el MSB es cero.

Ejemplo:

El número $10010101_{vas} = -(16+4+1) = -21$

El número $01010101_{vas} = +(64+16+4+1) = +85$

4.2. NÚMEROS EN COMPLEMENTO A 1.

Números en complemento a 1, la representación de un número positivo en complemento a 1, es exactamente igual a como lo hacíamos en valor absoluto y signo.

Para representar un número negativo en complemento a 1, se parte del número positivo y se le complementan todos los bits. O lo que es lo mismo se cambian todos los unos por ceros y los ceros por unos:

ejemplo:

$+7 = 0111_{c1;}$

$-7 = 1000_{c1}$

4.2. NÚMEROS EN COMPLEMENTO A 1.

Conversión de números en complemento a 1 a decimal.

Números positivos (comienzan con 0) se suman los pesos de las posiciones que tienen a 1 el bit correspondiente del número binario a convertir, es igual que en binario natural.

Al valor obtenido se le pone signo + (ya que comienza por 0)

Números negativos se suman los pesos de las posiciones que tienen a 0 el bit correspondiente del número binario a convertir .

Al valor obtenido se le pone signo - (ya que comienza por 1)

4.2. NÚMEROS EN COMPLEMENTO A 1.

Ejemplos:

Número binario: 01100111_{C1} = + (64+ 32+4+2+1) = +103

Número binario: 10011000_{C1} = - (64+32+4+2+1) = -103

Ejercicios:

Convertir 01100000_{C1} a decimal:

Convertir 11100100_{C1} a decimal:

4.3. NÚMEROS EN COMPLEMENTO A 2.

La representación de un *número positivo* en complemento a 2, es exactamente igual a como lo hacíamos en valor absoluto y signo y en C1.

Para representar un *número negativo* en complemento a 2, se parte del número en complemento a 1 y se le suma 1. Evidentemente el bit de mayor peso debe seguir valiendo 1 (es un número negativo).

ejemplo:

$+7 = 0111_{MS}\ \ 0111_{c1}\ \ 0111_{c2}$

$-7 = 1000_{c1}$; $1000_{c1} + 1 = 1001 = -7$ en complemento a 2

4.3. NÚMEROS EN COMPLEMENTO A 2.

ejemplo representar los números +25 y -25 en complemento a 2 (C2) de 8 bits :

+25= 000100011_{c1} = 000100011_{c2} = 000100011_{sm}

Para obtener -25, partimos del número +25 expresado en C1, lo complementamos bit a bit y le sumamos 1

000100011_{c1} lo complementamos obteniendo 111011100

por último le sumamos 1 obteniendo 111011101; por tanto

-25 expresado en C2 es = 111011101.

Un método más rápido para calcular cualquier número negativo en complemento 2 es la siguiente:

4.3. NÚMEROS EN COMPLEMENTO A 2.

Partimos del número positivo, (recuerde que el bit mas a la izquierda debe ser 0), con ese número y empezando por el bit mas a la izquierda (LSB) empezamos a leer bits, hasta que encontremos un 1, ese primer uno lo dejamos como está y el resto de bits los complementamos cambiando ceros por unos y unos por ceros.

Ejemplo representar los números +6 y -6 en complemento a 2 (C2) de 8 bits :

$$+6 = 00000110_{c2}$$

$$+ 6= 11111010_{c2}$$

4.3. TABLA DE NÚMEROS CON SIGNO.

Como habrá deducido al representar los números con signo, el rango de valores sufre modificaciones como ejemplo se da una tabla con los límites de estos números y para 3 bits , recuerde que estos límites dependen del número de bits con que representemos los valores.

DEC	SM	C1	C2	DEC	SM	C1	C2
0	000	000	000	-0	100	111	------
1	001	001	001	-1	101	110	111
2	010	010	010	-2	110	101	110
3	011	011	011	-3	111	100	101
				-4	------	--------	1100

4.3.TABLA DE NÚMEROS EN Ca2.

Tabla de equivalencias entre números en decimal y complemento a 2 de 4 bits .

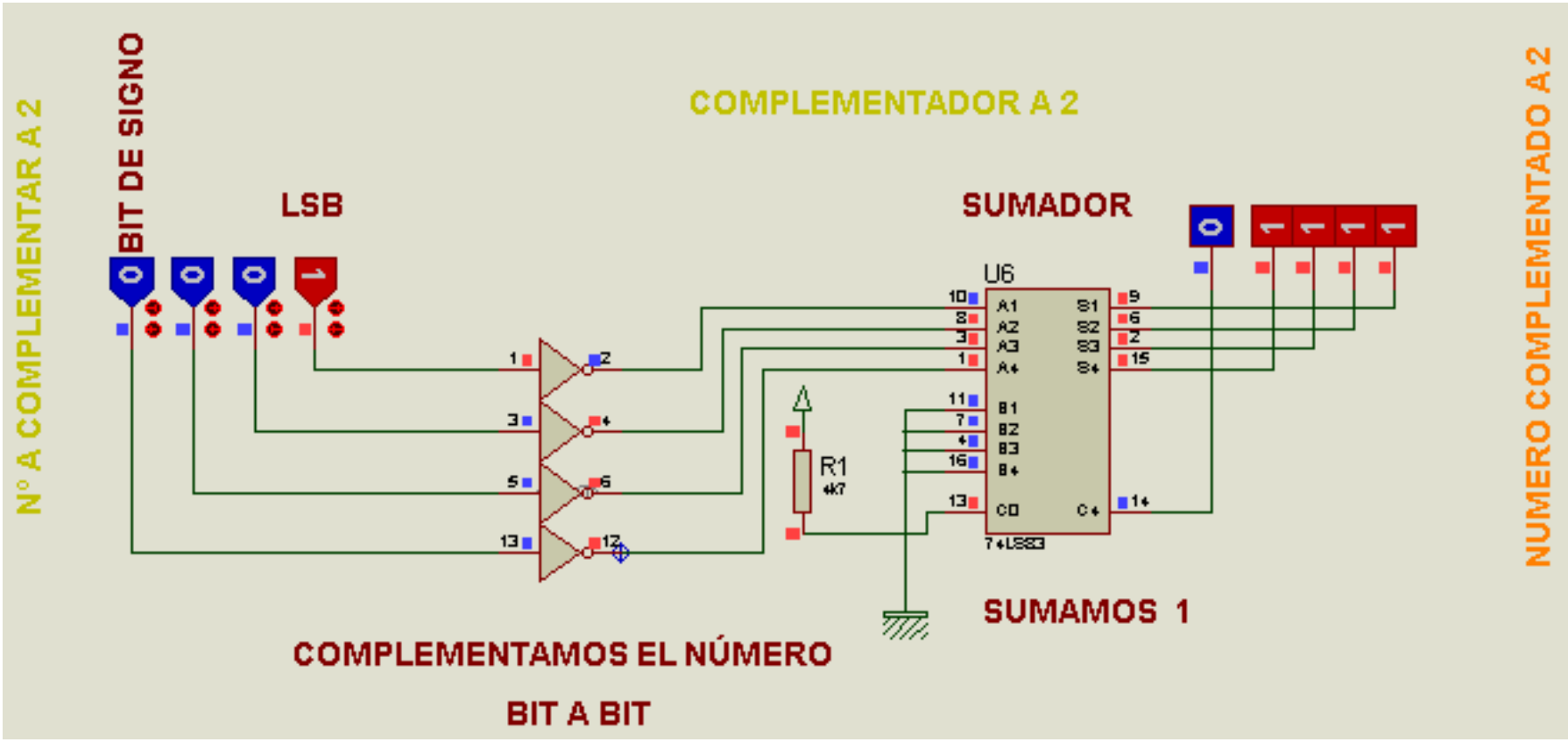
DEC	C2	DEC	C2
0	0000	-----	-------
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001
		-8	1000

4.4. RANGO DE LOS NÚMEROS EN Ca2.

Los números en Ca2 son el sistema habitual con que se representan los números en el mundo de la informática (lo marcado en negrita), por ello damos a continuación los rangos que poseen en función del número de bits empleados.

Nº de bits	Número + negativo	Número + positivo
4 (nibble)	-8	+7
5	-16	+15
6	-32	+31
7	-64	+63
8 (byte)	**-128**	**+127**
16 (int)	**- 32768**	**+32767**
32 (long int)	**- 2.147.483.648**	**+ 2.147.483.647**

4.4. CIRCUITO PARA COMPLEMENTAR A 2 UN NÚMERO EN Ca2.



5 . CONVERSIÓN DE Ca2 A DECIMAL.

Para convertir un número positivo de Ca2 a decimal se procede como en complemento a 1 o en binario, mediante los pesos.

Para los números negativos los más sencillo es volver a realizar al número un complemento a 2 con lo convertimos en positivo, lo evaluamos pero teniendo la precaución de ponerle el signo – ya que hemos partido de un número negativo.

6. OPERACIONES ARITMÉTICAS DE NÚMEROS CON SIGNO.

Las operaciones aritméticas que realizan las computadoras son siempre en Ca2, por lo que las operaciones que realizaremos serán siempre en este sistema.

SUMA:

Al sumar dos números con signo se pueden producir cuatro casos:

1. Uno positivo y otro negativo, y además el positivo es mayor en valor absoluto que el negativo (+32 y -18)
2. Uno positivo y el otro negativo, pero el negativo es mayor en valor absoluto que el positivo (+45 y -98)
3. Ambos números son positivos
4. Ambos números son negativos

6.1 SUMA DE 2 NÚMEROS UNO POSITIVO Y OTRO NEGATIVO.

La suma de un número positivo y negativo siempre genera un resultado correcto.

1. Suma de un número positivo y otro negativo, y además el positivo es mayor en valor absoluto que el negativo.

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	00010111	(23 en decimal)
sumando B =	11111001	(-7 en decimal)
	---------	
suma	100010000	(23-7 = 16 en decimal)

El resultado es correcto, se debe despreciar el acarreo.

6.1 SUMA DE 2 NÚMEROS UNO POSITIVO Y OTRO NEGATIVO.

2. Suma de un número positivo y otro negativo, y además el negativo es mayor en valor absoluto que el negativo.

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	00000011	(3 en decimal)
sumando B =	11111001	(-7 en decimal)
	---------	
suma	11111100	(3-7 = -4 en decimal)

El resultado es correcto.

6.2 SUMA DE 2 NÚMEROS POSITIVOS.

3. Suma de 2 números positivos . La suma de dos números positivos es siempre positiva

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	00000111	(7 en decimal)
sumando B =	00001001	(9 en decimal)
	---------	
suma	00010000	(7+9 = 16 en decimal)

El problema que puede surgir es que se produzca un desbordamiento (overflow) que es cuando el número resultante no se puede representar con los bits de que disponemos.

6.2 SUMA DE 2 NÚMEROS POSITIVOS.

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	01111111	(127 en decimal)
sumando B =	00000011	(3 en decimal)
	---------	
suma	10000010	(127+3 = 130 en decimal)

El resultado no es correcto ya que el bit mas a la izquierda es 1, indicando que es un número negativo, cuando el resultado debería ser un número positivo.

6.3 SUMA DE 2 NÚMEROS NEGATIVOS.

La suma de dos números negativos es siempre NEGATIVA

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	11100111	(-25 en decimal)
sumando B =	11001001	(-55 en decimal)
	---------	
suma	110110000	(-25 -55 = -80 en decimal)

El acarreo final se desprecia y el resultado es correcto.

6.3 SUMA DE 2 NÚMEROS NEGATIVOS.

El problema surge cuando la suma de ambos operandos excede del valor -128.

Ejemplo tenemos que sumar 2 números en Ca2 de 8 bits

sumando A =	11100111	(-25 en decimal)
sumando B =	10001001	(-119 en decimal)
	---------	
suma	101110000	(-25 -119 = -144 en decimal)

El resultado es incorrecto ya que hemos obtenido un bit de signo positivo. Se ha producido un overflow

6.3 SUMA DE 2 NÚMEROS NEGATIVOS.

Teniendo en cuenta todo lo descrito anteriormente pasamos a diseñar un sumador de 4 bits en complemento a 2.

Recuerde que se puede dar cualquiera de los casos descritos anteriormente y que debemos generar una señal de overflow si el resultado no se puede representar en 4 bits.

Para generar esta señal de overflow nos fijamos en los signos de los sumandos y en signo de la suma.

Desarrollamos la tabla de verdad donde Sa = signo del sumando A, Sb= signo del sumando B, Ss signo de la suma; O= Indicador de Overflow ; O = 1 operación correcta O = 0 se ha producido un overflow .

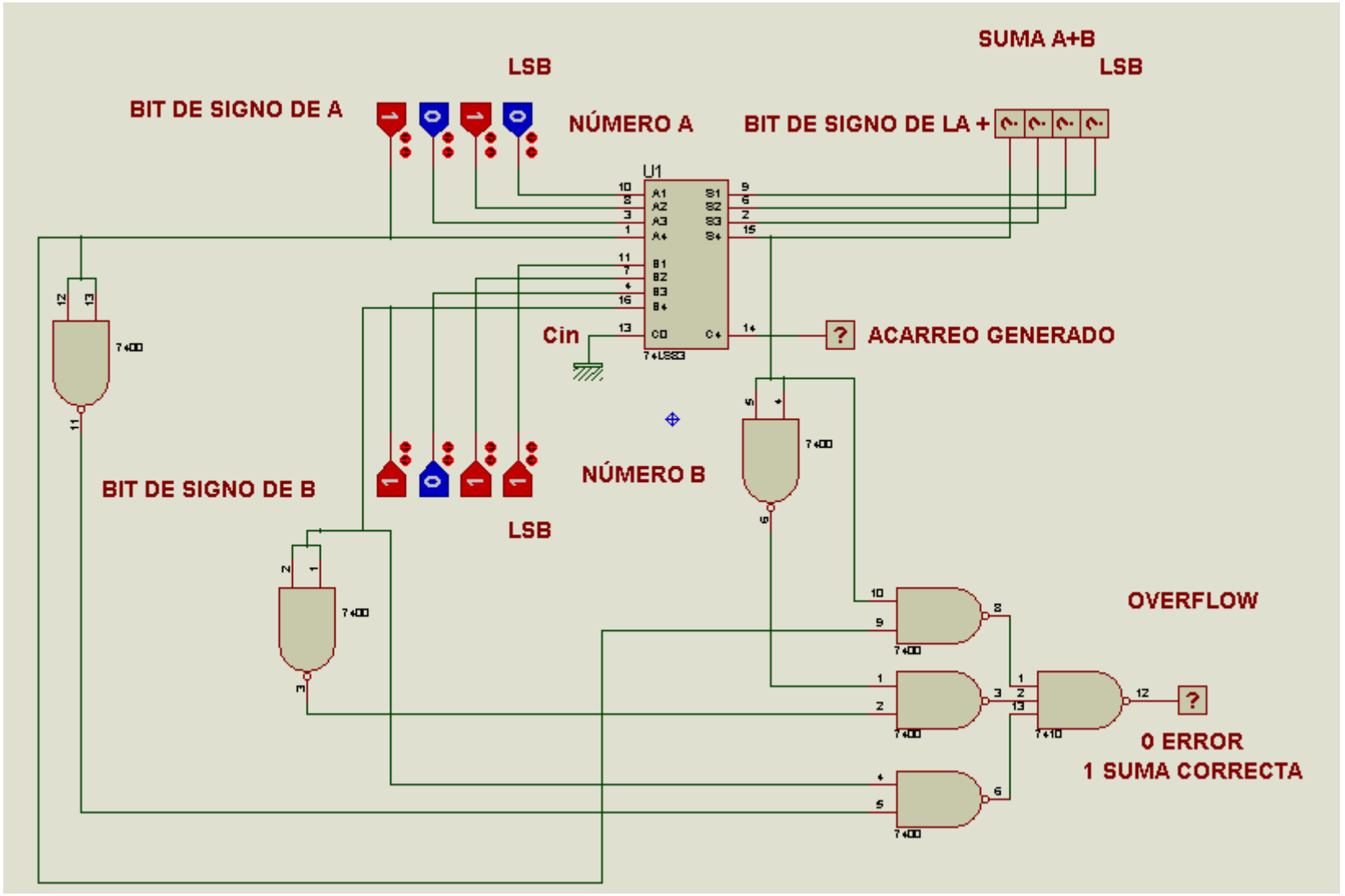
6.3 SUMA DE 2 NÚMEROS EN CA2.

SsSaSb	Overflow	Aclaración
0 0 0	1	Los 2 sumandos + ; suma + CORRECTO
0 0 1	1	1 sumando + y el otro –; suma + CORRECTO
0 1 0	1	1 sumando - y el otro +; suma + CORRECTO
0 1 1	0	Los 2 sumandos - ; suma + INCORRECTO
1 0 0	0	Los 2 sumandos + ; suma - INCORRECTO
1 0 1	1	1 sumando + y el otro –; suma - CORRECTO
1 1 0	1	1 sumando + y el otro -; suma - CORRECTO
1 1 1	1	Los 2 sumandos - ; suma - CORRECTO

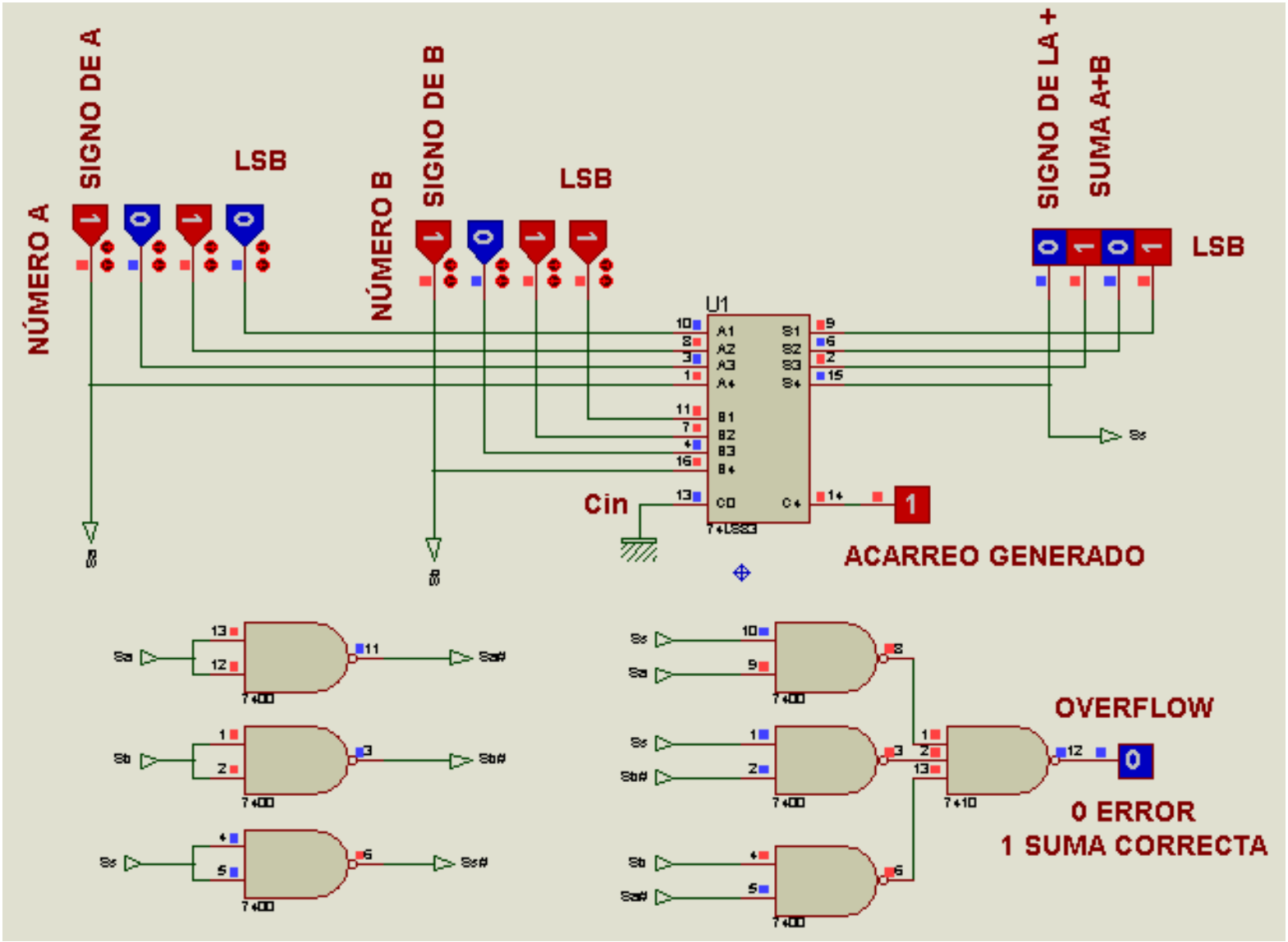
Si simplificamos por Karnaug obtenemos :

$$O = S_S\, S_A + \overline{S}_S\, \overline{S}_B + S_B \overline{S}_A$$

6.3 SUMA DE DOS NÚMEROS EN Ca2.



6.3 SUMA DE DOS NÚMEROS EN Ca2. (el mismo circuito)



6.3 RESTA DE DOS NÚMEROS EN Ca2.

La resta es un caso especial de la suma.

Para restar dos números, lo que se hace es sumar al minuendo el complemento a dos del sustraendo.

Lo que hacemos al hacer el complemento a dos del sustraendo es cambiarle de signo.

Para la resta tendremos en cuenta los mismos considerandos que hemos hecho para la suma.

Seguidamente diseñamos un circuito sumador/restador de 4 bits, nótese que debemos añadir una señal de control para determinar si es una suma o una resta.

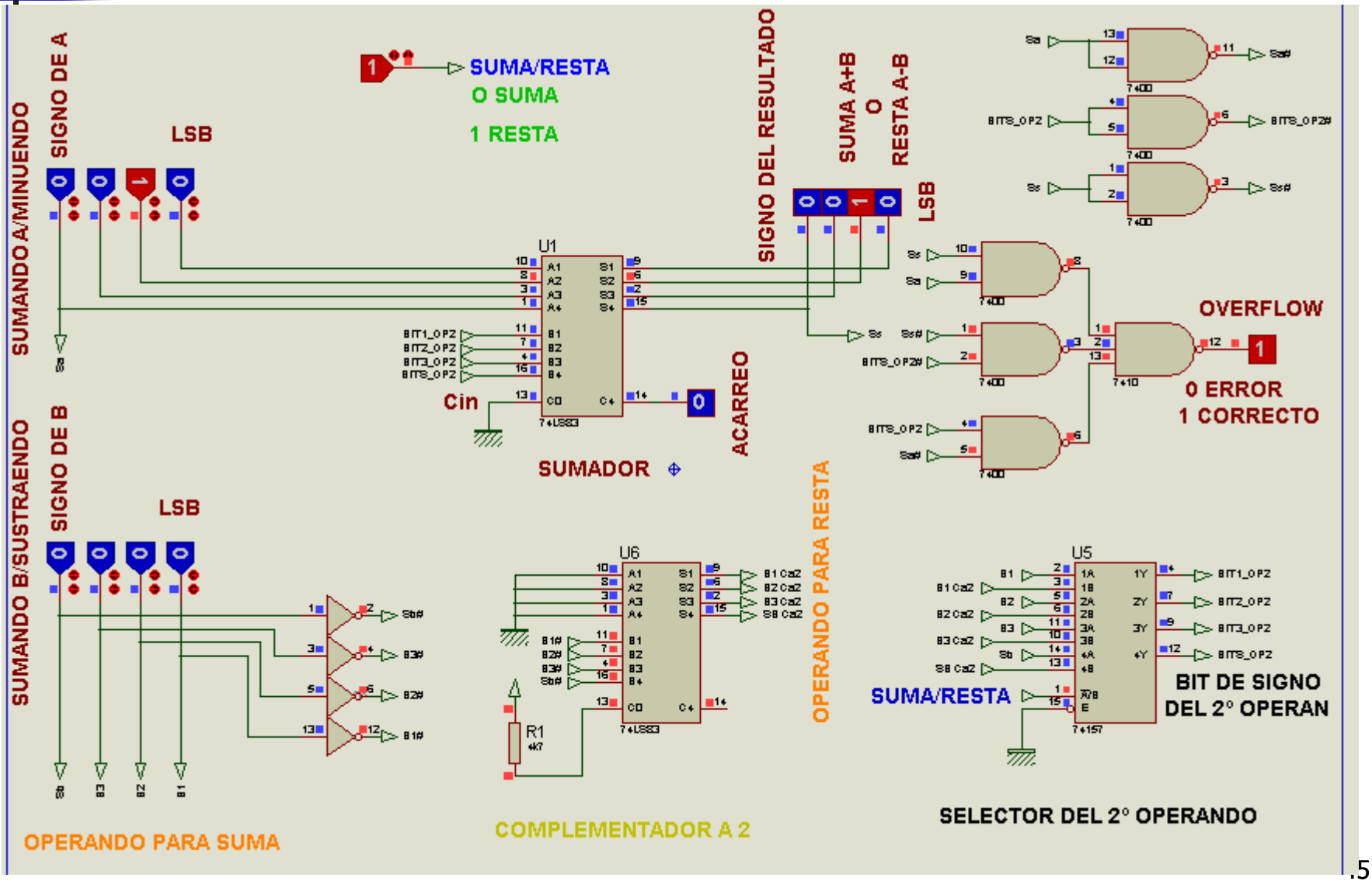
6.3 RESTA DE DOS NÚMEROS EN Ca2.

El primer operando (número A) es el primer sumando o minuendo, mientras que el segundo operando puede ser o 2º sumando o sustraendo.

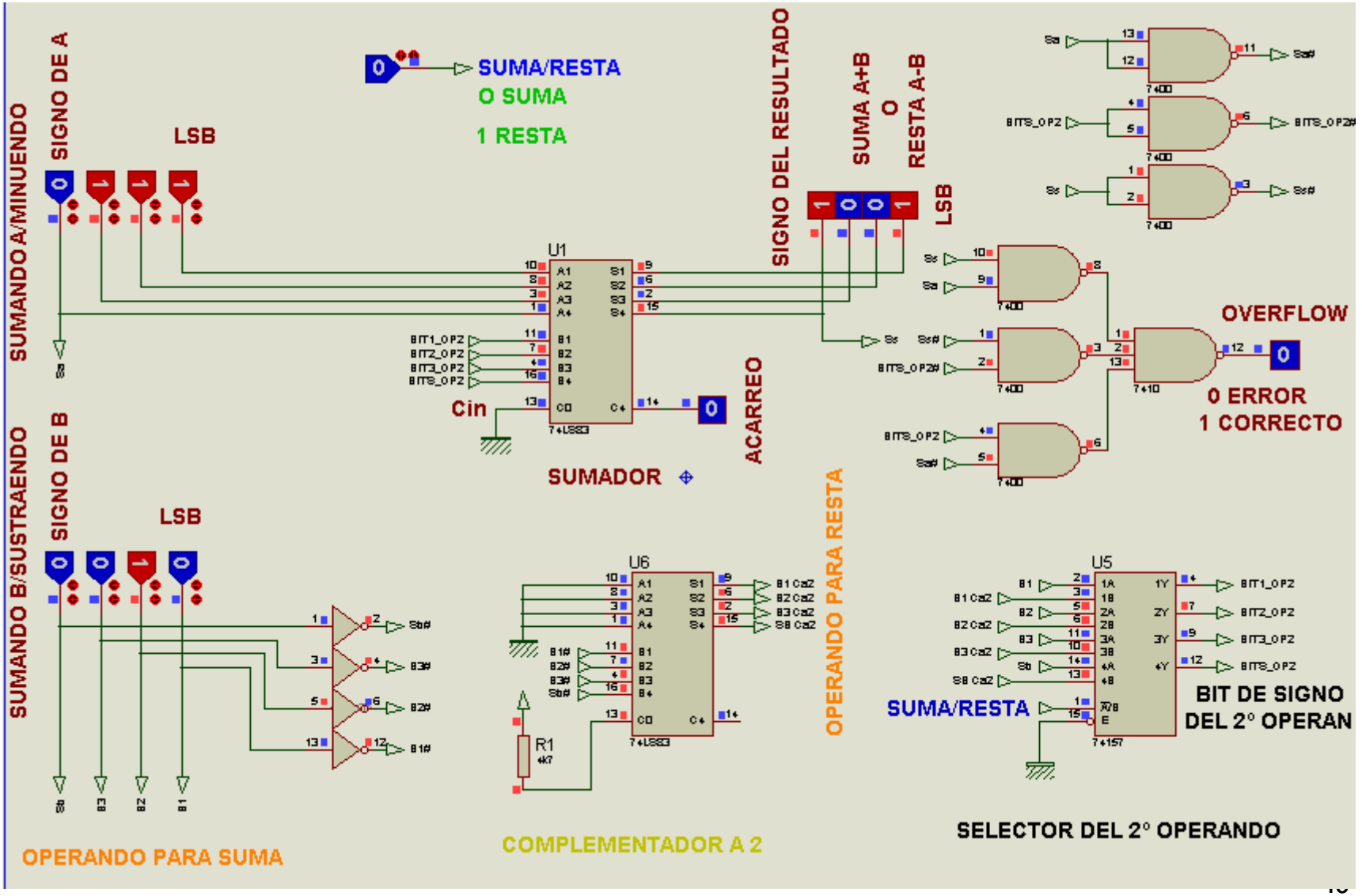
La detección de overflow es la misma que en el caso del sumador en Ca2.

Tenga en cuenta que para determinar el error de overflow, tenemos que utilizar el bit del signo del 2º operando

6.3 SUMADOR/RESTA DOR DE DOS NÚMEROS EN Ca2.



6.3 SUMADOR/RESTA DOR DE DOS NÚMEROS EN Ca2.



6.3 SUMADOR/RESTA DOR DE DOS NÚMEROS EN Ca2.