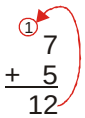
1. ADICION DE NUMEROS BINARIOS

Antes de analizar la suma de números binarios, recordemos como es el procedimiento para sumar dos números decimales. Por ejemplo, la operación 7+2 sería:

```
  7
+ 2
  9
```

En este caso el resultado es de un dígito, al igual que los operandos. Pero veamos ahora el caso 7+5:



```
  7
+ 5
 12
```

Se dice “siete mas cinco igual a doce, me llevo uno”. Es decir, existe un acarreo de “uno” desde las cifras de las unidades hacia las decenas.
Para el caso de la suma de números binarios, el procedimiento es similar, con la diferencia de que pueden darse cuatro casos:

0 + 0 = 0	no hay acarreo
0 + 1 = 1	no hay acarreo
1 + 1 = 10	1+1 es 0, con un acarreo de 1 que se suma a la columna siguiente
1 + 1 + 1 = 11	la suma de los dos primeros 1 da 10 (hay acarreo), este 10 + 1 da 11 (sin acarreo)

Veamos algunos ejemplos de sumas de binarios:

```
   011 (3)        1001  (9)
+  110 (6)     +  1111 (15)
  1001 (9)       11000 (24)
```

2. NUMEROS NEGATIVOS

Para representar que un número binario es negativo, se utiliza un bit extra para indicar el signo (tradicionalmente el MSB – More Significative Bit), tal que este es **0** para los números positivos, y **1** para los números negativos.

2.1. Magnitudes Signadas

Así como en base 10 expresamos +5 y -5, en binario, operando en 8 bits y reservando el MSB para el signo, podemos escribir:

+5 = 00000101 y -5 = 10000101

Habrá dos representaciones para el 0:

+0 = 00000000 y -0 = 10000000

Y con los 8 bits se pueden representar los números en el rango:

+127 = 01111111 a -127 = 11111111

Para poder, por ejemplo, sumar dos números según este esquema, se deben comparar los signos: si son iguales se suman las cantidades y se coloca el signo común; si son distintos se comparan los números, se resta el menor del mayor, y al resultado se le coloca el signo del mayor.
Pero todas estas operaciones (**si**, **comparar**, etc.) complican la lógica, además de que hay dos representaciones para el 0. Esto hace que esta forma de representación no sea la más adecuada, por lo que para representar números negativos se utiliza la representación siguiente:

2.2. Complemento a la base:
El complemento a la base (o **complemento a 2** para el caso de un número binario) se utiliza para representar cantidades negativas. Dado un número **N**, el complemento a la base es el número que hay que adicionarle a **N** para llegar a la base. Para la base, siempre debe tenerse en cuenta la cantidad de cifras con las que se está trabajando. Por ejemplo, en el sistema decimal, el complemento a la base del número 76_{10} es:

- Si se trabaja con 3 cifras (base 100), el complemento es **24**, ya que 76 + 24 = 100
- Si se trabaja con 3 cifras (base 1000), el complemento es **924**, ya que 76 + 924 = 1000
- Si se trabaja con 4 cifras (base 10000), el complemento es **9924**, ya que 76 + 9924 = 10000

Entonces el complemento de un número siempre dependerá de la cantidad de cifras con las que se trabaje.

En el caso de números binarios, se acostumbra trabajar con cantidades de 8 cifras (8 bits, o *byte*), de 16 cifras (16 bits, o *word*), o de 32 cifras (32 bits, o *double word*). En cada caso, el complemento a 2 de un mismo número será distinto, ya que la base es diferente. Por ejemplo, para el caso del número 76_{10}, su equivalente binario es 01001100_2 en representación de 8 bits. El complemento a 2 será **10110100**, ya que 10110100 + 01001100 = 100000000.

Entonces, generalizando, se dice que si **N** es un número binario positivo de **n** cifras, su correspondiente negativo **-N** se define como:

$\mathbf{-N = 2^n - N}$ o sea $\mathbf{N + (-N) = 2^n}$

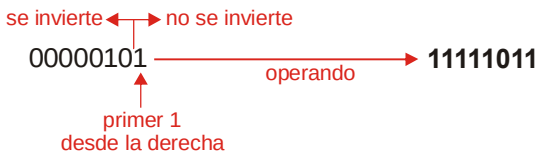
Para obtener el complemento a 2 de un número binario, hay dos formas rápidas. Tomemos de ejemplo el +5 decimal, cuyo equivalente binario es 00000101_2. Para obtener el -5:

- 1º forma: se niega el numero positivo (se cambian 0 por 1 y viceversa, a esto también se lo conoce como complemento a 1), y luego se suma 1. Para el ejemplo 00000101, primero se niega, con lo cual queda 11111010, y luego se suma 1:

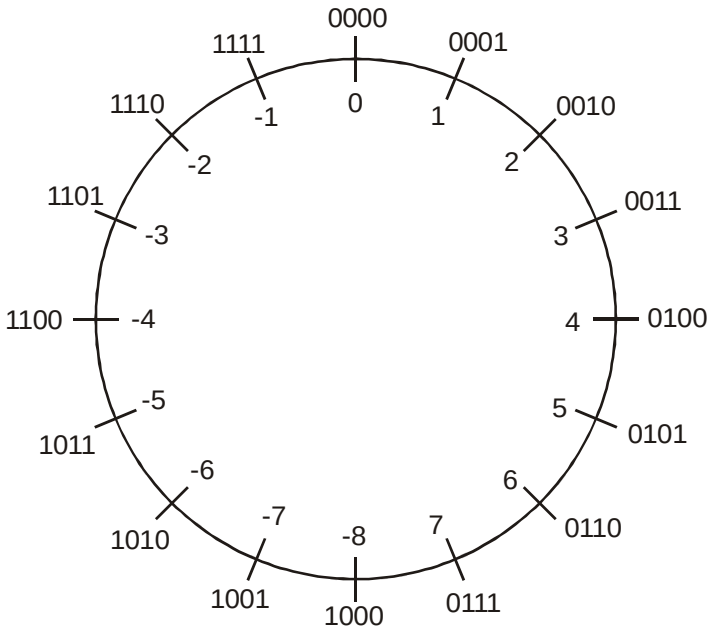
$$\begin{array}{r} 11111010 \\ + \qquad\quad 1 \\ \hline 11111011 \end{array}$$

  Por lo tanto **11111011** es la representación de 8 bits del decimal **- 5**. Notar que el bit de más a la derecha (el MSB) se encuentra en **1**, indicando que se trata de un número negativo.

- 2º forma: leyendo el número positivo de derecha a izquierda, se dejan los bits tal cual estén hasta el primer 1 inclusive. Luego de este primer uno, los bits restantes hacia la izquierda se niegan. Para el caso del ejemplo:



Resumiendo, el complemento a 2 de un número binario:
- Se utiliza para transformar en suma una operación de resta, ya que la operación **N - M** se transforma en la operación **N + (complemento a 2 de M)**. El hecho poder reducir una operación de resta a una operación de suma hace que se simplifique la complejidad de los circuitos que realizan operaciones aritméticas.
- Elimina la inconsistencia de tener dos representaciones para el cero. Utilizando complemento a dos, con **n** bits se puede representar números en el rango **$-(2^{n-1})$** al **$(2^{n-1} - 1)$**. Por ejemplo, con 8 bits (n = 8), se pueden representar números en el rango $\mathbf{11111111_2 = -128_{10}}$ al $\mathbf{01111111_2 = 127_{10}}$, y hay una sola representación para el $\mathbf{0_{10} = 00000000_2}$.
- Ejemplo de representación para cuatro bits:



2.3. Posibles casos

Veamos algunos casos posibles que pueden darse. Consideremos números de 8 bits, por lo cual tendremos un rango de -128_{10} a 127_{10}.

Suma de dos números positivos

Supongamos que sumamos dos números, el 18_{10} y el 27_{10}. Según lo ya visto, la operación sería:

$18_{10} = 00010010_2$ $\quad 27_{10} = 00011011_2$

$$\begin{array}{r} 00010010 \\ + \quad 00011011 \\ \hline 00101101 \end{array}$$

El resultado es 00101101_2, que es 45 en decimal, lo que es correcto. Notemos que el MSB es **0**, lo que está indicando un resultado positivo.

Veamos ahora el caso de la suma de 101_{10} mas 48_{10}. El resultado debería ser 149_{10}:

$101_{10} = 01100101_2$ $\quad 48_{10} = 00110000_2$

$$\begin{array}{r} 01100101 \\ + \quad 00110000 \\ \hline 10010101 \end{array}$$

El resultado es 10010101_2, que es -107_{10} y no 149_{10}. El MSB es 1, indicando que se trata de un número negativo ¿Cómo puede ser que sumando dos números positivos se obtenga un negativo? Lo que ocurrió aquí es que se excedió la capacidad de representación posible con 8 bits, es decir el resultado correcto que es 149_{10} no puede representarse con 8 bits, razón por la cual el resultado de la suma anterior no es el esperado. En forma general, cuando el signo de ambos sumandos es el mismo (ambos positivos o ambos negativos), el resultado debe tener el mismo signo. Si esto no ocurre, como en este caso, significa que se excedió el rango representable.

Resta de dos números positivos

Veamos dos casos, en el primero el minuendo mayor que el sustraendo, y en el segundo el minuendo menor que el sustraendo.

Primer caso, se quiere realizar la operación 76_{10} menos 16_{10}. Esta resta la puedo realizar como una suma, haciendo 76_{10} + (compl. a dos de 16_{10}).

$76_{10} = 01001100_2$
$16_{10} = 00010000_2$ $\quad$ compl. a 2 de $16_{10} = 11110000_2$

Entonces 76_{10} + (compl. a dos de 16_{10}) queda:

$$\begin{array}{r} 01001100 \\ + \quad 11110000 \\ \hline 1\ 00111100 \end{array}$$

Notar que aquí hay un noveno bit de acarreo, pero como estoy trabajando en formato de 8 bits, a este noveno bit lo descarto, por lo que el resultado sería **00111100**. El MSB del resultado está en **0**, lo que indica un signo positivo, y el equivalente decimal del resultado es 60_{10}, que es el esperado.

Veamos ahora el segundo caso, la operación 76_{10} menos 79_{10}. Igual que en el caso anterior, se resuelve haciendo 76_{10} + (compl. a dos de 79_{10}):

$76_{10} = 01001100_2$
$79_{10} = 01001111_2$ compl. a 2 de $79_{10} = 10110001_2$

$$\begin{array}{r} 01001100 \\ + \quad 10110001 \\ \hline 11111101 \end{array}$$

El resultado es **11111101**. El MSB en **1** está indicando que se trata de un resultado negativo. Para hallar su equivalente decimal, debemos hacer el complemento a 2 del mismo, es decir:

Compl. a 2 de $11111101_2 = 00000011_2 = 3_{10}$

Es decir, el resultado entonces es:

$11111101_2 \equiv -3_{10}$

2.4. Cambio del rango de representación

Consiste en pasar de un formato de representación a otro, digamos por ejemplo pasar de una representación de 8 bits a 16 bits. Esto es útil cuando el resultado de una operación excedería el rango de representación. Un ejemplo sería si se quiere sumar en 8 bits los números 101_{10} y 48_{10}, como vimos anteriormente el resultado excede la representación en 8 bits. La solución es entonces pasar a una representación mayor, por ejemplo 16 bits. La pregunta es, dado un número signado de 8 bits, ¿cómo obtengo su representación en 16 bits? Bien, la forma más rápida y sencilla es la siguiente: se completan los bits faltantes con bits de signo. Por ejemplo:

$101_{10} = 01100101_2$ en formato de 8 bits. Para pasar a formato de 16 bits, debo agregar a la derecha los 8 bits faltantes, iguales al bit de signo original. Es decir:

$101_{10} = 01100101_2$ en formato de 8 bits = 0000000001100101_2 en formato de 16 bits.

Lo mismo ocurre con magnitudes negativas: si por ejemplo tengo la representación en 8 bits del número decimal -16_{10} y quiero obtener su representación en 16 bits, solo tengo que agregar bits de signo hasta completar los 16 bits:

$-16_{10} = 11110000_2$ en formato de 8 bits = 1111111111110000_2 en formato de 16 bits.

La operación inversa (es decir, pasar de 16 bits a 8) también es posible, sólo si el número a convertir no excede el rango de representación de 8 bits. En este caso, sólo deben eliminarse los bits de signo sobrantes. Ejemplo:

$-3_{10} = 1111111111111101_2$ en formato de 16 bits = 11111101_2 en formato de 8 bits.

$220_{10} = 0000000011011100_2$ en este caso no se puede realizar la transformación a 8 bits, ya que 220_{10} excede el rango representable con 8 bits.

3. Circuitos lógicos de suma

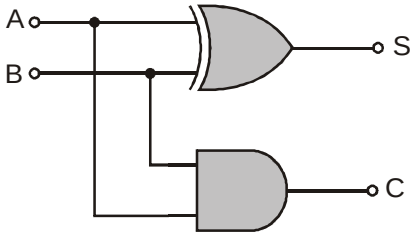
3.1. Semisumador binario (half adder)

El semisumador es un circuito digital que permite sumar dos números binarios **A** y **B** de 1 bit. La salida es la suma de los dos bits **S** y el acarreo **C**.

La tabla de verdad del semisumador es:

Entradas		**Salidas**	
A	**B**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

El mismo puede implementarse mediante dos compuertas, según el siguiente circuito:



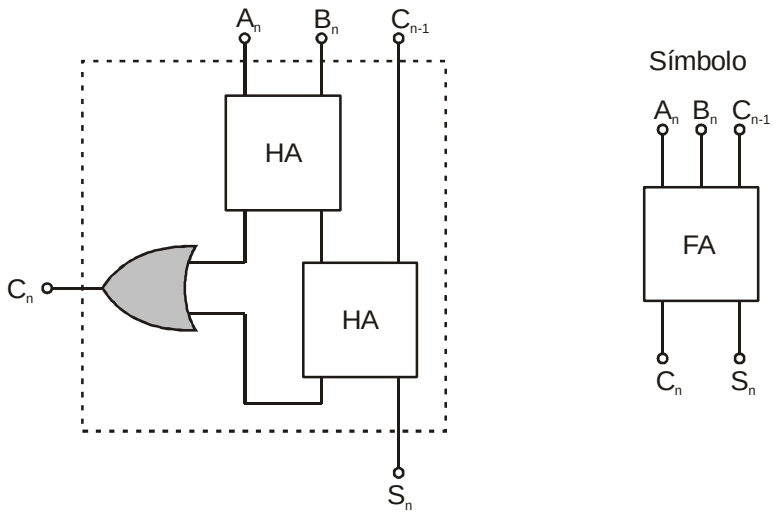
Símbolo

3.2. Sumador Completo (full adder)

Con el semisumador anterior, se pueden sumar dos números binarios de un solo bit. Cuando se tienen que sumar dos números de **n** bits, se debe también sumar el acarreo de los bits anteriores. Es decir que el sumador completo requiere tres entradas, y entrega dos salidas. La tabla de verdad del sumador completo es:

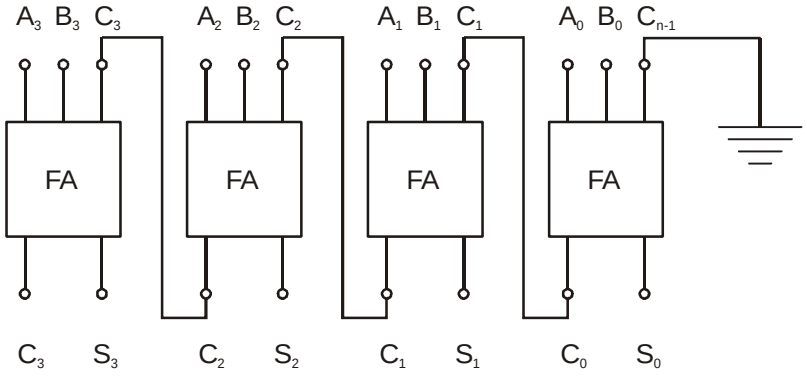
Entradas			**Salidas**	
C_{n-1}	A_n	B_n	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

El sumador completo puede sintetizarse a partir de dos semisumadores, según el siguiente circuito:



3.3. Sumador completo para números de 4 bits

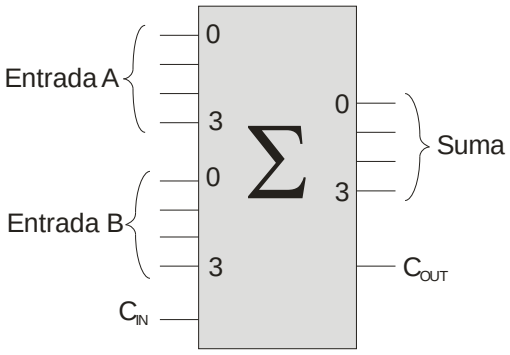
Para realizar la suma de dos números binarios de 4 bits, se pueden utilizar 4 sumadores completos de un bit. En general, para sumar números de **n** bits, se podrían utilizar **n** sumadores completos de un bit. Volviendo al caso de 2 binario de 4 bits, el circuito sería:



Donde podemos observar que el acarreo de salida de un semisumador se conecta al acarreo de entrada del siguiente. En el caso del sumador de los bits menos significativos, el acarreo de entrada se deja a masa (0 lógico) ya que no hay acarreo de entrada.

La configuración anterior ya viene integrada en un chip disponible comercialmente. Uno de los más conocidos es el 74LS283, siendo de tecnología TTL, pero también hay disponibles en versiones CMOS.

El símbolo gráfico que suele utilizarse para representar un sumador completo de 4 bits es el siguiente:



3.4. Unidad verdad / complemento

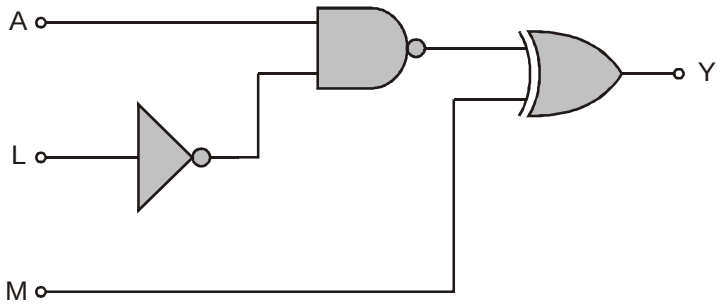
Consiste en un circuito que posee tres entradas. Una de ellas es un bit **A**, y las dos restantes son entradas de control (llamémosles **L** y **M**). Según los estados de las señales de control, la salida del circuito puede ser:

- 0 lógico.
- 1 lógico.
- La entrada **A**.
- El complemento de la entrada **A** (o sea $\overline{\mathbf{A}}$).

Según lo anterior, la tabla de verdad de este circuito sería:

Entradas de Control		**Salida**
L	**M**	**Y**
0	0	$\overline{\mathbf{A}}$
0	1	**A**
1	0	1
1	1	0

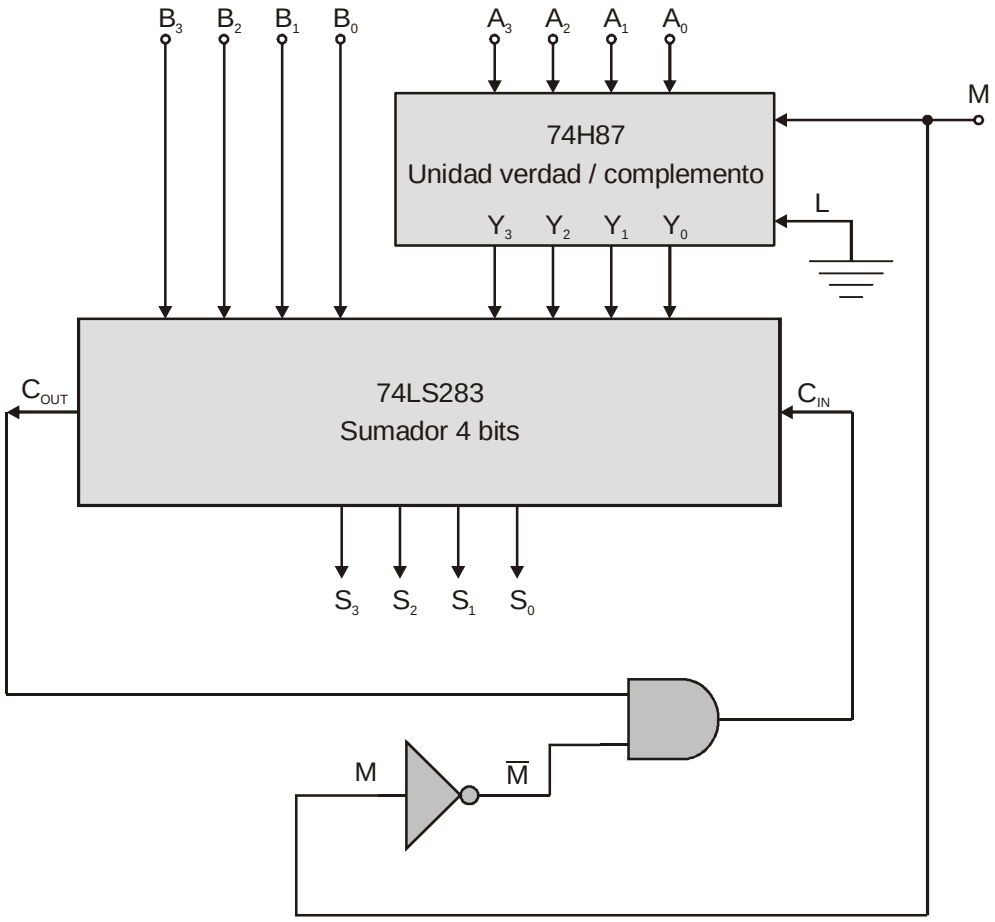
Y la síntesis del mismo es:



La configuración anterior también se encuentra disponible en circuitos integrados comerciales, como por ejemplo el 74H87, que contiene cuatro unidades verdad / complemento.

3.5. Sumador – Restador de 4 bits

Combinando un sumador completo de 4 bits con una unidad verdad / complemento de 4 bits, podemos obtener un bloque que según ciertas entradas de control, realice la suma o resta de dos números binarios de 4 bits. El diagrama circuital sería el siguiente:



Si en el circuito anterior se hace **M = 1**, en la salida de la unidad verdad / complemento tendremos el operando **A** sin complementar, con lo cual el sistema realiza la suma de los operandos **A** y **B**, obteniéndose a la salida la suma **S**.

Si ahora hacemos **M = 0**, en la salida de la unidad verdad / complemento tendremos el complemento a uno de **A**, es decir $\overline{\mathbf{A}}$ (no el complemento a 2, que es el necesario). El resultado de esta operación hace que se obtenga un acarreo en el acarreo de salida del sumador, y este acarreo se introduce en el acarreo de entrada a través de las compuertas. En definitiva, en este caso se hace la operación **B +** $\overline{\mathbf{A}}$ **+ 1**, o lo que es lo mismo, **B - A**. Este sistema realiza la resta entre **B** y **A** siempre que **B** sea mayor que **A**, caso contrario no se produciría acarreo en la salida del sumador y no se realimentaría a la entrada del mismo, por lo que el resultado ya no sería válido.

El esquema anterior, con algunas pequeñas modificaciones, forma parte de lo que se denomina **ALU** o Unidad Aritmético Lógica, que es el corazón de cualquier sistema microprocesador (tales como computadoras, microcontroladores, PLC, etc.).