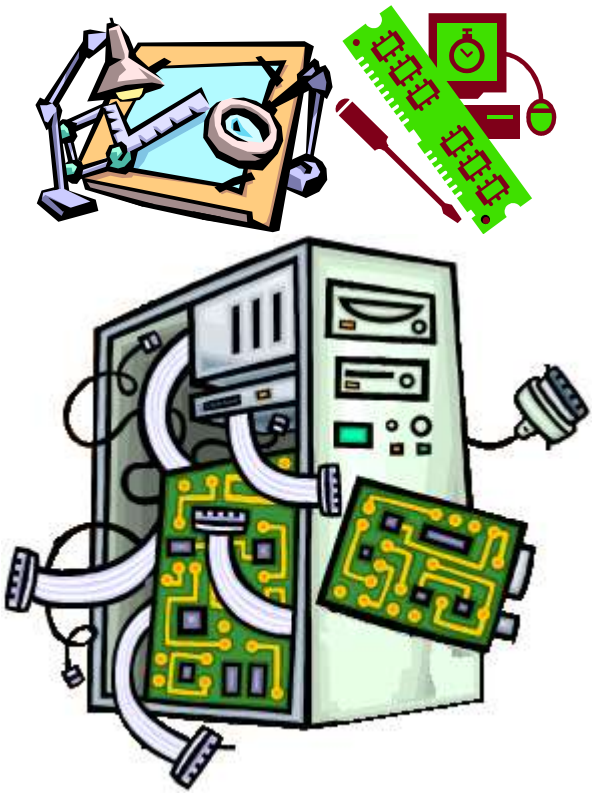
Ejercicios de Arquitectura de Computadoras



José Garzía

2009

✎ Explique qué es el rebose cuando se suman dos números sin signo y cómo puede ser detectado.

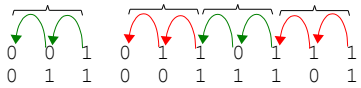
Hay rebose cuando pretendemos almacenar el resultado en un registro de la misma longitud que los operandos, pero no cabe.
El arrastre de la última etapa sirve como indicador de rebose.

✎ Explique qué es el rebose cuando se suman dos números de signos opuestos y cómo puede ser detectado.

Cuando se suman dos números de distintos signos ¡nunca hay rebose!.
El módulo del resultado siempre será mayor o igual que el mayor de los módulos de los operandos.

✎ La siguiente pareja de números binarios *x* e *y* se suma en un sumador binario paralelo con propagación de arrastres. Analice el número de secuencias de arrastre que comienzan simultáneamente.

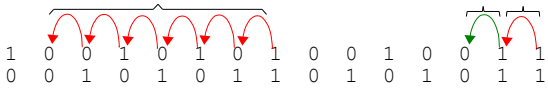
x = 0 0 1 0 1 1 0 1 1 1
y = 0 1 1 0 0 1 1 1 0 1



Hay 4 secuencias, todas ellas de longitud 2

✎ La siguiente pareja de números binarios *x* e *y* se suma en un sumador binario paralelo con propagación de arrastres. Analice el número de secuencias de arrastre que comienzan simultáneamente.

x = 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1
y = 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1



Hay 3 secuencias, la mayor de longitud 6

✎ Demuestre que el código BCD no es autocomplementario.

Lo haremos con un contraejemplo:
Sea *x* = 0000:

$C_1(x_b) = 1111_b$.
$C_9(x_{10}) = 9_{10} = 1001_b$.

Como $C_1(x_b) \neq C_9(x_{10})$, no es autocomplementario.

✎ Sean dos números de 12 bits representados en *binario puro*: *x* = 000011000010, *y* = 000101110001. Calcule su suma y exprésela en BCD.

Los números están dados en binario, por tanto tenemos libertad para realizar la suma en binario (y luego convertirlos a BCD).

```
  0 0 0 0     1 1 0 0     0 0 1 0
+ 0 0 0 1     0 1 1 1     0 0 0 1
-----------------------------------
  0 0 1 0     0 0 1 1     0 0 1 1
```

$= 563_d = 0101\ \ 0110\ \ 0011_{BCD}$

✎ Sean dos números binarios de 8 bits representados en código BCD: *x* = 01010100, *y* = 00101000. Realice su suma en BCD.

```
  0 1 0 1     0 1 0 0
+ 0 0 1 0   + 1 0 0 0
-------------------------
  0 1 1 1     1 1 0 0   ←
```

Este dígito es mayor que 1001.
Es necesaria una corrección que consiste en sumarle 0110 y traspasar un acarreo al dígito BCD siguiente

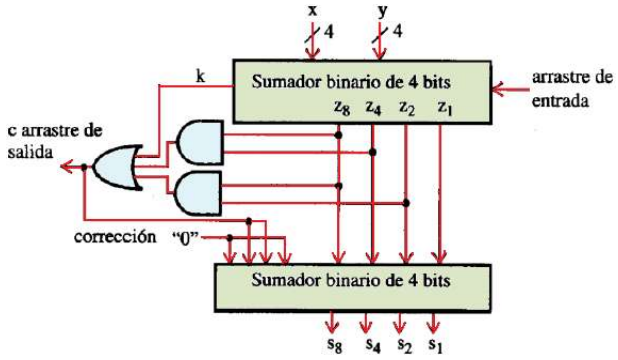
1º. Suma de 0110

```
  1 1 0 0
+ 0 1 1 0
---------
  0 0 1 0
```

2º.Traspaso del acarreo

```
  0 1 1 1
+ 0 0 0 1
---------
  1 0 0 0
```

Resultado tras la corrección:
1 0 0 0 0 0 1 0



Sean dos números binarios de 8 bits representados en código BCD: x = 010110001001, y = 001101000111. Realice su suma en BCD.

```
  0 1 0 1     1 0 0 0     1 0 0 1
+ 0 0 1 1     0 1 0 0     0 1 1 1
-----------------------------------
  1 0 0 0     1 1 0 0     0 0 0 0
```

Sumar 0110 y traspasar un acarreo al dígito BCD siguiente

```
                  + 1     0 1 1 0
                -------   -------
                1 1 0 1   0 1 1 0
```

Sumar 0110 y traspasar un acarreo al dígito BCD siguiente

```
      + 1     0 1 1 0
    -------   -------
    1 0 0 1   0 0 1 1     0 1 1 0
```

Sean dos números binarios de 8 bits representados en código BCD: x = 0011100101010100, y =0011100100000110. Realice su suma en BCD.

```
  0 0 1 1     1 0 0 1     0 1 0 1     0 1 0 0
+ 0 0 1 1     1 0 0 1     0 0 0 0     0 1 1 0
-----------------------------------------------
  0 0 1 1     0 0 1 0     0 1 0 1     1 0 1 0
```

```
      + 1     0 1 1 0           + 1     0 1 1 0
    -------   -------         -------   -------
    0 1 1 1   1 0 0 0         0 1 1 0   0 0 0 0
```

Sumar 0110 y traspasar un acarreo al dígito BCD siguiente

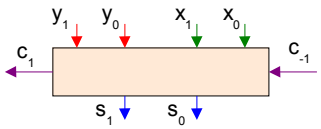
A partir de las expresiones de S (resultado de la suma) y C (acarreo de la suma) de un sumador binario completo SBC, desarrolle las expresiones del resultado de la suma (s_1 y s_0) y del acarreo (c_1) de la suma de los números binarios de dos bits x_1x_0 e y_1y_0.

$s_0 = x_0 \oplus y_0 \oplus c_{-1}$

$s_1 = x_1 \oplus y_1 \oplus c_0 = x_1 \oplus y_1 \oplus [\, x_0 \cdot y_0 + (x_0 \oplus y_0) \cdot c_{-1}]$

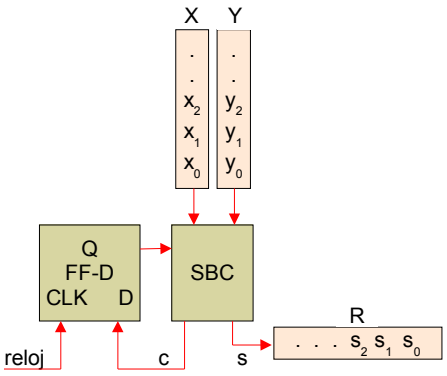
$c_0 = x_0 \cdot y_0 + (x_0 \oplus y_0) \cdot c_{-1}$

$c_1 = x_1 \cdot y_1 + (x_1 \oplus y_1) \cdot c_0 = x_1 \cdot y_1 + (x_1 \oplus y_1) \cdot [x_0 \cdot y_0 + (x_0 \oplus y_0) \cdot c_{-1}]$

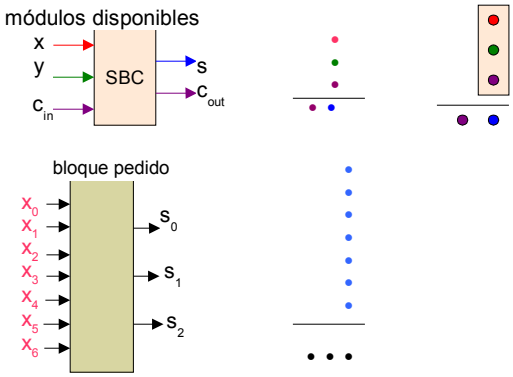


¿Cuántos SBCs de 1 bit harían falta para construir un sumador binario serie capaz de sumar dos números binarios de n bits?.

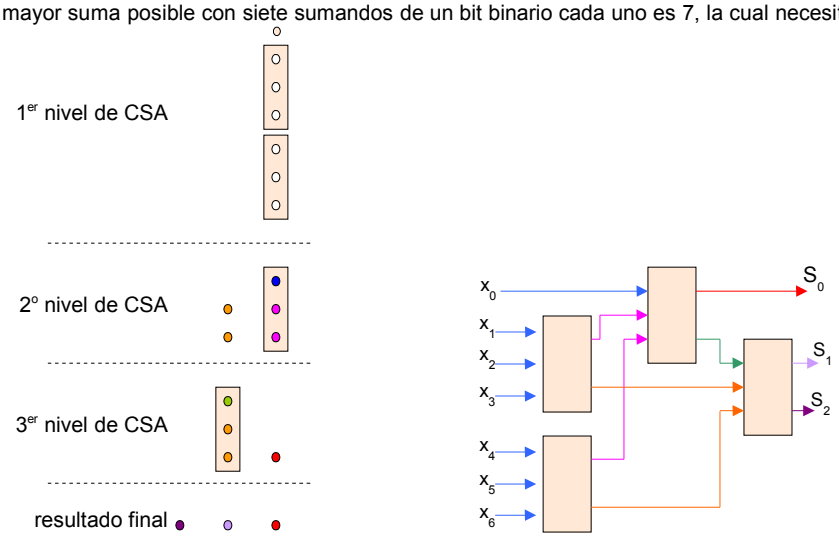
Si es un sumador serie, basta con un único SBC.



Se desea diseñar un sumador combinacional de 7 números de 1 bit utilizando únicamente módulos sumadores binarios completos, SBC. ¿Cuántos módulos de este tipo serían necesarios?.

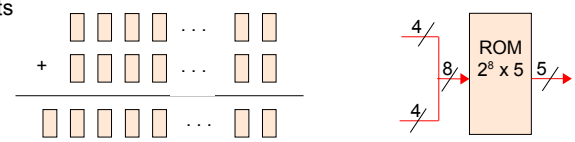


La mayor suma posible con siete sumandos de un bit binario cada uno es 7, la cual necesita tres dígitos binarios.



Se pretende construir un sumador binario de dos números de 4 bits cada uno, con una memoria ROM. Indique la capacidad mínima de dicha memoria ROM:

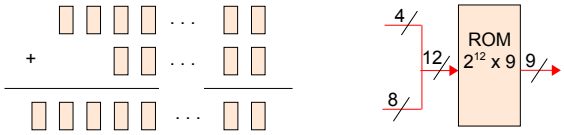
Al sumar dos palabras con signo de longitud *n* bits (el máximo número representable es $2^{n-1}-1$), el resultado ocupará una longitud menor o igual a *n+1* bits. En el peor de los casos posibles, X + Y= $(2^{n-1}-1)+(2^{n-1}-1) = 2\cdot 2^{n-1} - 2 = 2^n - 1 - 1 < 2^n - 1$, número con signo representable con *n+1* bits



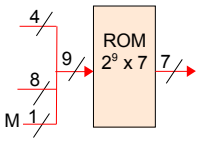
Se pretende construir un sumador binario de dos números, uno de 8 bits y el otro de 4 bits, con una memoria ROM. Indique la capacidad mínima de dicha memoria ROM:

Al sumar dos palabras con signo de longitud *n* y *m* bits (los máximos números representables serán $2^{n-1}-1$ y $2^{m-1}-1$, respectivamente),, el resultado ocupará una longitud menor o igual a *max(n,m)+1* bits.

Sea *m=max(m,n)*. En el peor de los casos posibles, X + Y= $(2^{n-1}-1)+(2^{m-1}-1) < 2^{m-1}+2^{m-1} - 2 = 2\cdot 2^{m-1} - 2 = 2^m - 2 < 2^m - 1$, número con signo representable con *m+1* bits.



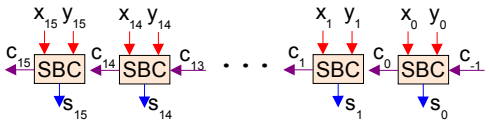
Se pretende construir con una memoria ROM un sistema sumador-restador de dos números de cuatro bits cada uno, $x_3x_2x_1x_0$ e $y_3y_2y_1y_0$ con una señal de control *M* adicional para indicar la operación a realizar. Indique la capacidad mínima de dicha memoria ROM:



✎ Compare, para el peor caso, la aceleración que se consigue con un sumador con aceleración de arastre a dos niveles para cuatro grupos de cuatro bits (longitud de palabra: dieciséis bits) con respecto al sumador binario paralelo correspondiente.

Para analizar los tiempos de ejecución contaremos los niveles de puertas lógicas que debe atravesar la señal en el peor caso posible en cada uno de ambos diseños: sumador paralelo y sumador con dos niveles de aceleración del arrastre. Entendiendo como pero caso cuando existe un arrastre que debe ser propagado a través de todas las etapas.

Sumador paralelo:

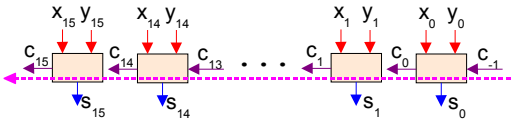


En cada SCB las expresiones de las salidas son:

$s_i = x_i \oplus y_i \oplus c_{i-1}$

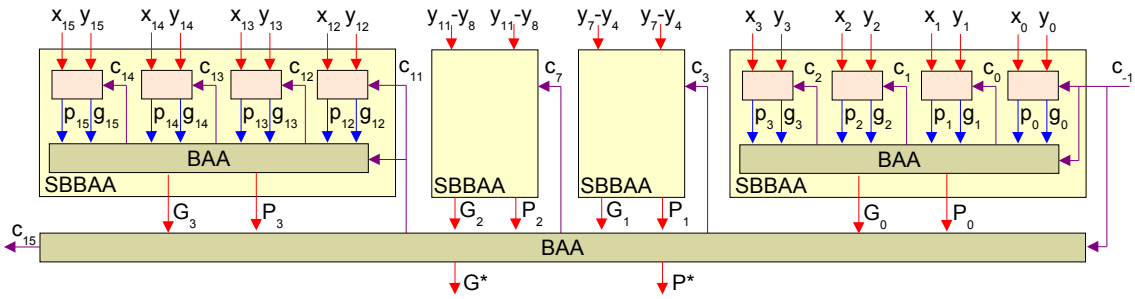
$c_i = x_i \cdot y_i + x_i \cdot c_{i-1} + y_i \cdot c_{i-1}$

Cada SBC tiene dos niveles de puertas: primero el nivel AND para generar los términos producto y después el nivel OR para sumar dichos productos. La suma puede darse por concluida cuando se ha esperado por un posible acarreo propagado desde c_1 hasta c_{15}:



Así pues, en el sumador paralelo, el retardo en el peor caso posible es $\tau_{paralelo} = 2 \cdot 16 = 32$

Sumador con dos niveles de aceleración de arrastre:

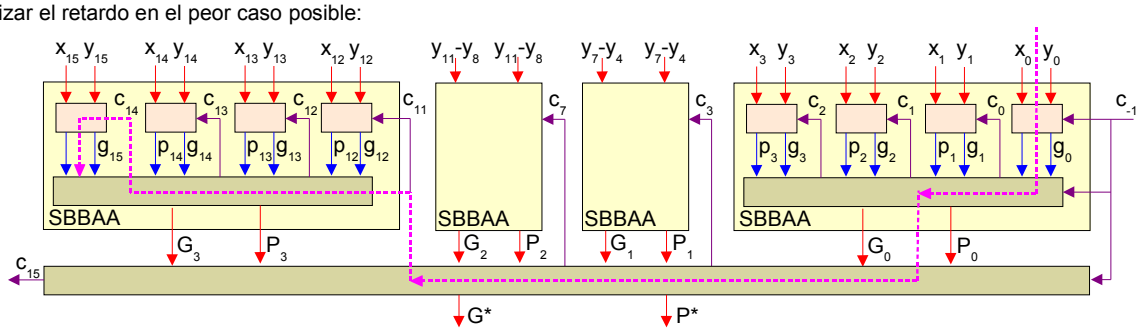


Para nuestro análisis no son necesarias las señales de los bits de suma, basta con representar las de los acarreos p_i y g_i.

El nivel más interno de aceleración lo forman cuatro sumadores binarios completos modificados (SBCMs, los bloques más pequeños de los representados), conectados a un bloque de aceleración de arrastres BAA para constituir un sumador binario con bloque de aceleración de arrastre (SBBAA) de cuatro bits. En el primer SBAA, los bits p_3, g_3, p_2, g_2, p_1, g_1, p_0 y g_0 son calculados simultáneamente. Ídem para los otros tres SBAAs.

El nivel más externo de aceleración lo forman cuatro SBBAAs conectados mediante un BAA. Los bits P_3, G_3, P_2, G_2, P_1, G_1, P_0 y G_0 son calculados simultáneamente (por supuesto, después que p_3, g_3, p_2, g_2, p_1, g_1, p_0 y g_0).

Para analizar el retardo en el peor caso posible:



La suma puede darse por concluida cuando se ha esperado hasta el cálculo de s_{15} hasta c_{15}.

Expresión	Retardos
$g_i = x_i \cdot y_i$, $p_i = x_i \oplus y_i$	1
$G_0 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3$	2
$c_{11} = G_2 + G_1 P_2 + G_0 P_1 P_2 + c_{-1} P_0 P_1 P_2$	2
$c_{14} = g_{14} + g_{13} p_{14} + g_{12} p_{13} p_{14} + c_{11} p_{12} p_{13} p_{14}$	2
$s_{15} = x_{15} \oplus y_{15} \oplus c_{14}$	2

No se tienen en cuenta P_3, G_3, P_2, G_2, P_1, ni G_1; puesto que se obtienen simultáneamente a G_0.
Tampoco se tiene en cuenta

$c_{15} = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_{-1} P_0 P_1 P_2 P_3$,

puesto se obtiene simultáneamente a c_{11} (y antes que s_{15}).

Por tanto, la suma total de los retardos de este sumador con dos niveles de aceleración es:

$$\tau_{dos\ niveles\ de\ aceleración} = 9$$

Comparando un caso con otro, la mejora relativa es:

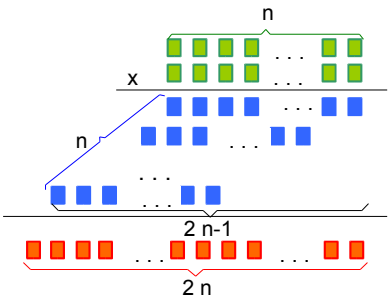
$$\frac{\tau_{paralelo}}{\tau_{dos\ niveles\ de\ aceleración}} = \frac{32}{9} = 3.56$$

✎ Analice cuál es el máximo número de dígitos que puede tener el producto de dos números, *x* e *y*, de *n* dígitos cada uno en base B.

$(X \cdot Y)_{max} = X_{max} \cdot Y_{max} = (B^n-1) \cdot (B^n-1) = B^n \cdot B^n + 1 - 2 \cdot B^n = B^{2n} + 1 - B^{n+1} \leq B^{2n} + 1 - 2 \cdot 1 = B^{2n} - 1$, número sin signo representable con $2 \cdot n$ bits.

✎ Se pretende construir un multiplicador binario para dos números de 8 bits, utilizando puertas lógicas tipo AND y sumadores binarios completos (SBC). ¿Cuál es el número de puertas AND que hay que utilizar?.

Al multiplicar dos palabras sin signo de longitud *n* bits (el máximo número representable es 2^n-1), el resultado ocupará una longitud menor o igual a $2 \cdot n$ bits:

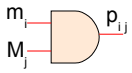


En el peor de los casos posibles, $X \cdot Y = (2^n-1) \cdot (2^n-1) = 2^n \cdot 2^n + 1 - 2 \cdot 2^n = 2^{2n} + 1 - 2^{n+1} \leq 2^{2n} + 1 - 2 = 2^{2n} - 1$, número sin signo representable con $2 \cdot n$ bits.

$$
\begin{array}{rcccc}
 & M_7 & M_6 & \cdots & M_0 \\
\times & m_7 & m_6 & \cdots & m_0 \\
\hline
 & p_{07} & p_{06} & \cdots & p_{00} \\
 & p_{17} & p_{16} & \cdots & p_{10} \\
 & \cdots & & & \\
+ & p_{77} & p_{76} & \cdots & p_{70} \\
\hline
P_{63} & P_{62} & \cdots & P_{04} & P_{03}\ P_{02}\ P_{01}\ P_{00}
\end{array}
$$

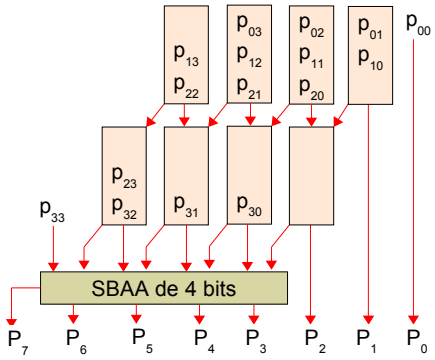
Hay 8 productos parciales. Cada producto parcial tiene 8 bits. Por tanto, la matriz es de 64 p_{ij}.

El i-ésimo producto parcial se calcula de la siguiente forma:
. Si m_i es '0', el producto parcial son 8 ceros.
. Si m_i es '1', el producto parcial es una copia del multiplicando M.
Dicho de otra forma, es como si m_i actuara de filtro. Este comportamiento se sintetiza con puertas AND. En conclusión, cada m_i construye con una puerta AND ($p_{ij} = m_i$ AND M_j)

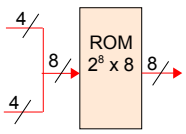


✎ Se pretende construir un multiplicador binario que multiplique dos números de cuatro bits cada uno. Haga el diseño utilizando únicamente ocho SBC (sumador binario completo), un SBAA (sumador binario con aceleración de arrastre) de cuatro bits y dieciséis puertas AND.

$$
\begin{array}{rcccc}
 & M_3 & M_2 & M_1 & M_0 \\
\times & m_3 & m_2 & m_1 & m_0 \\
\hline
 & p_{03} & p_{02} & p_{01} & p_{00} \\
 & p_{13} & p_{12} & p_{11} & p_{10} \\
 & p_{23} & p_{22} & p_{21} & p_{20} \\
+ & p_{33} & p_{32} & p_{31} & p_{30} \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3\ P_2\ P_1\ P_0
\end{array}
$$



✎ Se pretende construir un multiplicador binario de dos números de 4 bits cada uno con una memoria ROM. Indique la capacidad mínima de dicha memoria ROM:

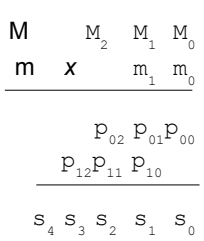


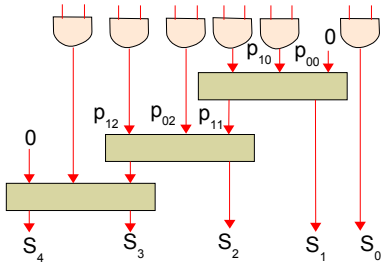
Se pretende construir un multiplicador binario que multiplique dos números sin signo de 2 y 3 bits. Realice el diseño con:

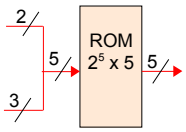
a) Puertas AND y sumadores binarios completos (SBC).

b) Una memoria ROM.

Hay 6 p_{ij}, cada uno de los cuales se construye con una puerta AND ($p_{ij} = m_i$ AND M_j)





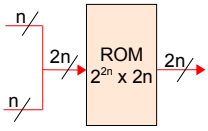


Se pretende construir un multiplicador binario de dos números de *n* bits cada uno con una memoria ROM. ¿Cuántos módulos de memoria ROM de *2n* palabras con *n* bits por palabra serían necesarios?.

Si queremos implementarlo con ROMs de $2^n x n$:

Número de filas: $\frac{2^{2^n}}{2^n} = 2^n \ filas$

Número de columnas: $\left\lceil \frac{2n}{n} \right\rceil = 2 \ columnas$



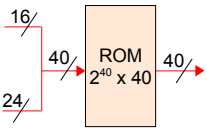
Por tanto, se necesitan $2^n \ x n$ bloques.

Se pretende construir un multiplicador binario de dos números de 16 y 24 bits, utilizando módulos memoria ROM con 256 palabras de 16 bits por palabra. Calcule cuántos módulos ROM serán necesarios.

Si queremos implementarlo con ROMs de $2^8 x 16$:

Número de filas: $\frac{2^{40}}{2^8} = 2^{32} \ filas$

Número de columnas: $\left\lceil \frac{40}{16} \right\rceil = 3 \ columnas$



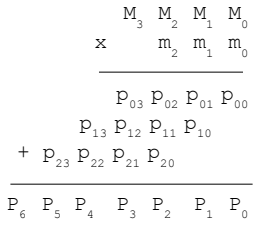
Por tanto, se necesitan 2^{32}x3 bloques.

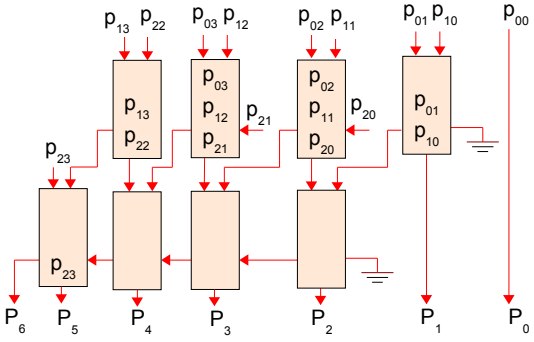
Diseñe un multiplicador de números binarios sin signo de 3 y 4 bits, con estas dos alternativas:

a) Sumadores binarios completos (SBC) y puertas AND.

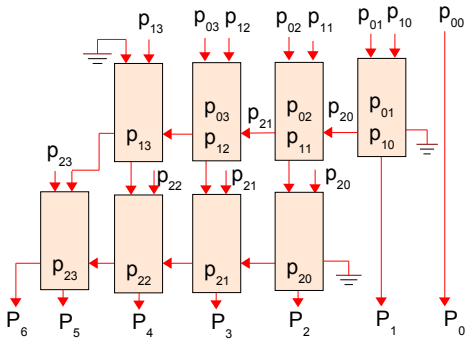
b) Sólo con memoria ROM.

a)

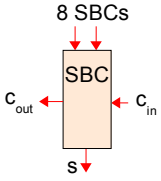




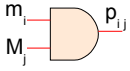
Otra forma de establecer las conexiones:



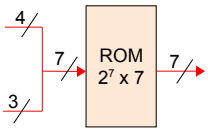
8 SBCs



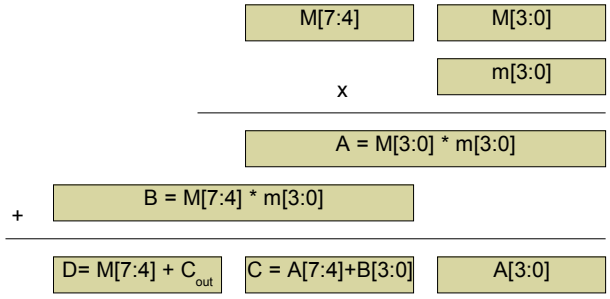
12 puertas AND

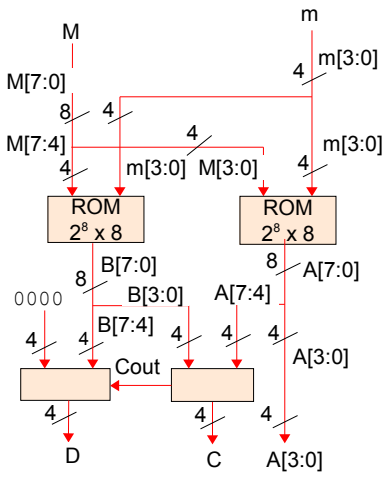


b)

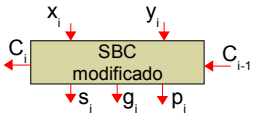


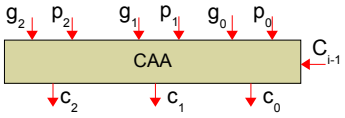
Diseñe un multiplicador binario que multiplique dos números de 4 y 8 bits, respectivamente. Utilice para ello exclusivamente dos módulos ROM de 256 palabras con 8 bits por palabra y sumadores binarios paralelos de 4 bits.

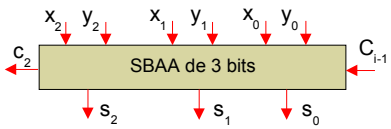




Se pretende construir un sumador binario con aceleración de arrastres (SBAA) para 2 números binarios X e Y de 3 bits cada uno. La figura de la izquierda muestra un sumador binario completo modificado (SBC modificado) con dos entradas de datos x_i e y_i y un acarreo de entrada c_{i-1}. La figura central muestra un circuito de aceleración de arrastres (CAA) de 3 bits con tres entradas de generación de acarreo g_i, tres entradas de propagación de acarreo p_i y un acarreo de entrada c_{-1}.



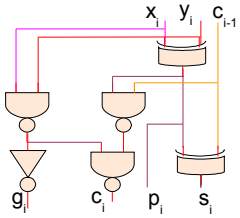




El diseño se realizará siguiendo los siguientes pasos:

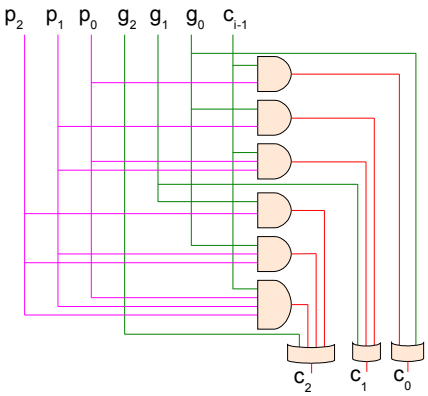
a) Exprese la función lógica de sus cuatro salidas: s_i (suma), c_i acarreo de salida), g_i (generación de acarreo) y p_i (propagación de acarreo). A partir de estas funciones lógicas, y empleando únicamente puertas lógicas, diseñe este SBC modificado.

$$s_i = x_i \oplus y_i \oplus c_{i-1} = p_i \oplus c_{i-1}$$
$$c_i = x_i \cdot y_i + (x_i \oplus y_i) \cdot c_{i-1} = g_i + p_i \cdot c_{i-1}$$

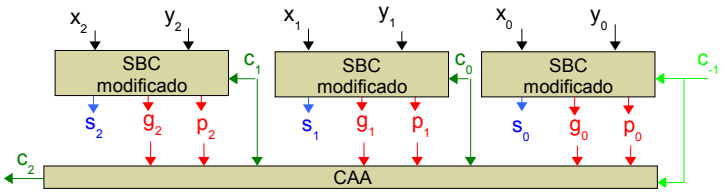


b) Exprese las funciones lógicas de los acarreos c_2, c_1 y c_0 generados por este módulo. A partir de estas funciones lógicas, y empleando únicamente puertas lógicas, diseñe este CAA.

$$c_0 = g_0 + c_{-1} \cdot p_0$$
$$c_1 = g_1 + g_0 \cdot p_1 + c_{-1} \cdot p_0 \cdot p_1$$
$$c_2 = g_2 + g_1 \cdot p_2 + g_0 \cdot p_1 \cdot p_2 + c_{-1} \cdot p_0 \cdot p_1 \cdot p_2$$



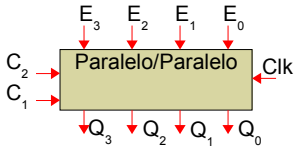
c) Utilizando únicamente 3 SBC modificados como los diseñados en a) y el CAA diseñado en b), construya razonadamente el sumador binario con aceleración de arrastre de 3 bits (SBAA) solicitado en el enunciado y mostrado en la figura de la derecha.



Diseñar un registro de desplazamiento de cuatro bits con multiplexores y elementos de memoria D.

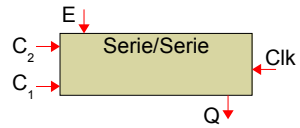
a) **Entrada en paralelo y salida en paralelo, desplazamientos lógicos** E_0, E_1, E_2, E_3 representan la entrada en paralelo; S_0, S_1, S_2, S_3 la salida paralelo; C_1, C_2 las líneas de control; y Clk la señal de reloj. El registro debe ser capaz de realizar los dos desplazamientos siguientes: LICS (Lógico-Izquierda-Cerrado-Simple) y LDCS (Lógico-Derecha-Cerrado-Simple). Además, el circuito debe permitir la carga en paralelo de la entrada y mantener la información almacenada sin modificar (no operación: NOP). La codificación de las entradas de control se muestra en la tabla adjunta.

Operación	C_2 C_1
NOP	0 0
LICS	0 1
LDCS	1 0
CARGA	1 1



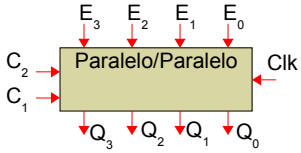
b) **Entrada en serie y salida en serie, desplazamientos lógicos**. *E* es la entrada en serie y *S* es la salida en serie.

Operación	C_2 C_1
NOP	0 0
LIAS	0 1
LDAS	1 0

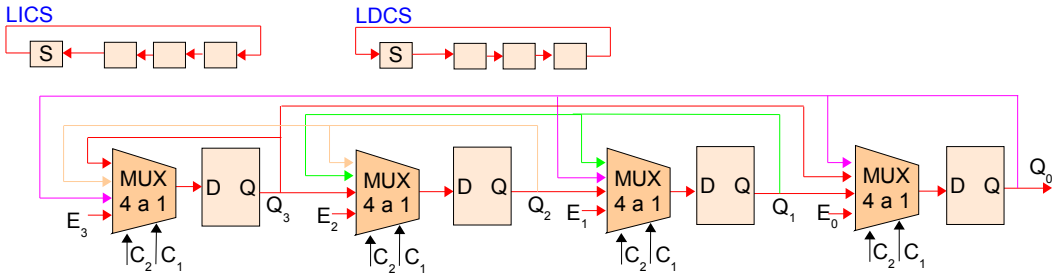


c) **Entrada en paralelo y salida en paralelo, desplazamientos aritméticos** E_0, E_1, E_2, E_3 representan la entrada en paralelo; S_0, S_1, S_2, S_3 la salida paralelo; C_1, C_2 las líneas de control; y Clk la señal de reloj. El registro debe ser capaz de realizar los dos desplazamientos siguientes: AICS (Aritmético-Izquierda-Cerrado-Simple) y ADCS (Aritmético-Derecha-Cerrado-Simple). Además, el circuito debe permitir la carga en paralelo de la entrada y mantener la información almacenada sin modificar (no operación: NOP). La codificación de las entradas de control se muestra en la tabla adjunta.

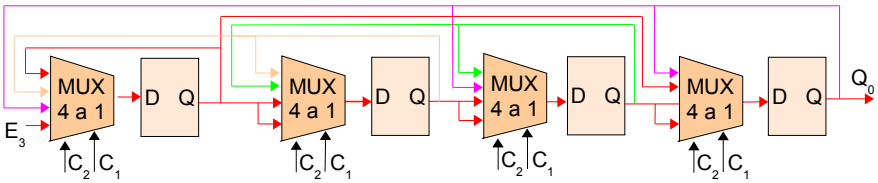
Operación	C_2 C_1
NOP	0 0
AICS	0 1
ADCS	1 0
CARGA	1 1



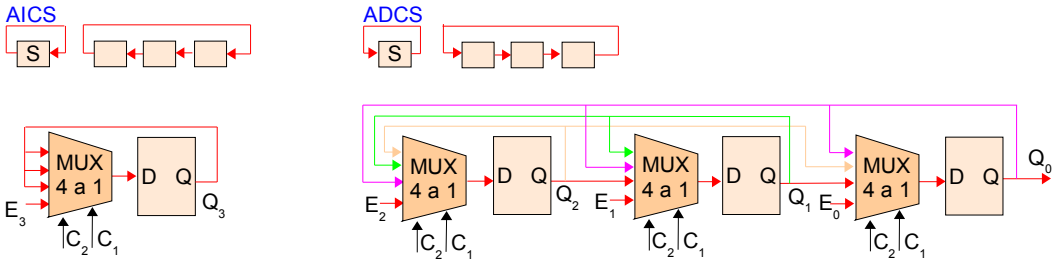
a)



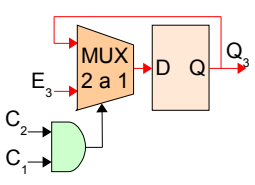
b) Como antes, pero sin E_2, E_1, E_0, Q_3, Q_2, ni Q_1.



c)



El bit de signo permanece constante, excepto en las cargas, cuando C_2=1, C_2=1. En este bit sólo hay dos operaciones diferentes. Por tanto, para economizar material, es posible utilizar un multiplexor 2 a 1 en la entrada del biestable del bit de signo:

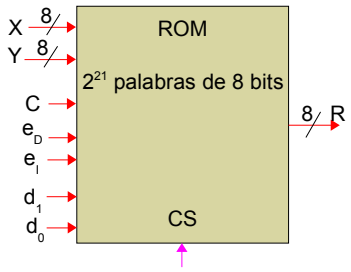


Se pretende diseñar un circuito con dos entradas X (X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0) e Y (Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0) y una salida *R* (r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0), todas de ocho bits, que sea capaz de:

- ✔ Comparar X e Y generando tres señales de salida r_7 (1 si X>Y, 0 en caso contrario), r_6 (1 si X=Y, 0 en caso contrario) y r_5 (1 si X<Y 0 en caso contrario).
- ✔ Realizar cuatro operaciones de desplazamiento sobre X empleando dos entradas adicionales llamadas $\mathbf{e_d}$ (entrada por la derecha) y $\mathbf{e_I}$ (entrada por la izquierda). El resultado de la operación es el número *R* de ocho bits.

Para ello se debe usar una señal de control **C** que seleccione el tipo de operación (comparación o desplazamiento) y dos señales más $\mathbf{d_1}$ $\mathbf{d_o}$ para indicar el tipo de operación de desplazamiento. Realice el diseño usando dos módulos de memoria ROM de 64 K palabras x 8 bits.

Contamos el número de entradas y salidas:



Por ello, en principio, el tamaño de la ROM debería ser 2^{21} palabras de 8 bits cada una. Pero podemos hacer dos simplificaciones, una por cada tipo de operación:

1ª. Cuando la operación es de comparación, habrá muchas palabras repetidas. Habrá que almacenar el mismo resultado de comparación para cada combinación de valores de e_D, e_I, d_1 y d_0. Dichas entradas no se utilizan en esta operación.
En este caso, la ROM tiene 16 entradas y 3 salidas

2ª. Cuando la operación es de desplazamiento, como sólo se pide desplazar los valores de X (8 bits), las 8 entradas Y a la ROM no serían necesarias y por lo tanto sólo necesitaríamos 8+2+2 entradas.
En este caso, la ROM tiene 12 entradas y 8 salidas

Todo este estudio ha sido para conocer el tamaño necesario de la ROM.

Nº de entradas = max{16, 12}+1 = 17. Se añade una entrada adicional para distinguir las zonas de memoria donde se almacenan las respectivas operaciones.

Nº de salidas = max{3, 8} = 8

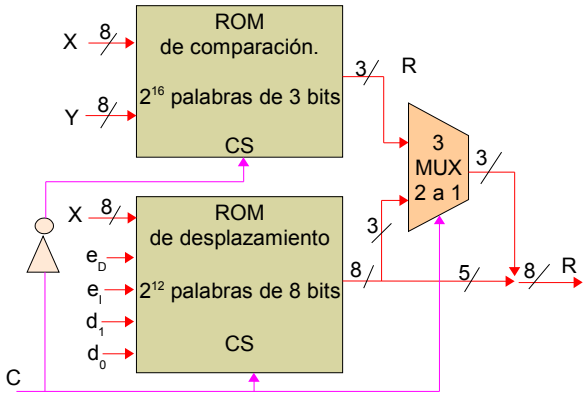
Así pues, el tamaño es 2^{17} palabras de 8 bits.

Cuando la operación sea de comparación se estarán malgastando 5 bits de salida.
Cuando la operación sea de desplazamiento se estarán malgastando 4 bits de entrada.

Si un objetivo es que no se malgaste ni un bit de memoria, se podría utilizar un CI con el tamaño justo y exacto para cada operación; y realizar las conexiones de la siguiente forma:

1º. Inhibiremos un CI y desinhibiremos el otro aplicando la señal C a uno y su complemento al otro en la entrada de selección de chip CS.

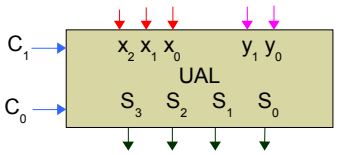
2º. De un CI salen 3 líneas y del otro salen 8. En caso de que las salidas de las ROM's no sean triestado, las 3 salidas de la ROM de comparación deben ser multiplexadas con las 3 más significativas de la ROM de desplazamiento.



De esta forma, la memoria consumida es $2^{16}+2^{12}$ palabras de 8 bits.

Se pretende realizar una Unidad Aritmético Lógica (UAL) como la mostrada en la figura, con dos entradas de datos *X* ($x_2x_1x_0$) e *Y* (y_1y_0), una entrada de control *C* (c_1c_0) y una salida de datos *S* ($S_3S_2S_1S_0$). El funcionamiento de la UAL viene descrito por la siguiente tabla:

c_1c_0 = 00 : suma	S = X + Y
c_1c_0 = 01 : resta	S = X - Y
c_1c_0 = 10 : comparación	Si X > Y entonces S = 1 0 0 0 Si X = Y entonces S = 0 1 0 0 Si X < Y entonces S = 0 0 1 0
c_1c_0 = 11 : sacar X	S = X (s_3=0, $s_2s_1s_0$=$x_2x_1x_0$)



Para resolver el problema, siga obligatoriamente los siguientes apartados:

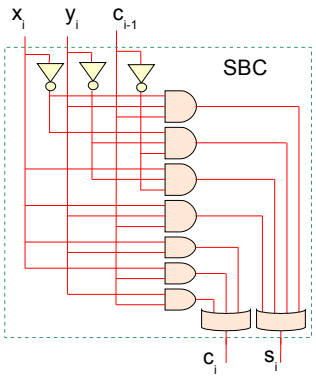
a) Utilizando únicamente puertas lógicas, diseñe un Sumador Binario Completo (SBC).

b) Diseñe un sumador/restador binario de números de tres bits utilizando SBC's como el diseñado en el apartado *a*) y las puertas lógicas necesarias.

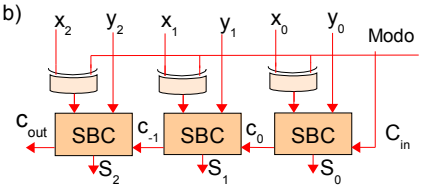
c) Diseñe un comparador de números de tres bits utilizando únicamente un módulo de memoria ROM. ¿Cuál es el tamaño necesario para este módulo de memoria ROM?. Indique claramente el significado de cada una de sus entradas y cada una de sus salidas. Escriba el contenido de la memoria ROM en forma de tabla.

d) Utilizando únicamente los módulos diseñados en los apartados anteriores y los módulos combinacionales necesarios, diseñe la UAL pedida.

a)

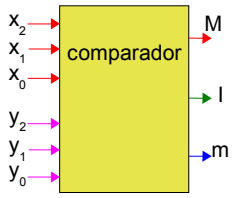


b)

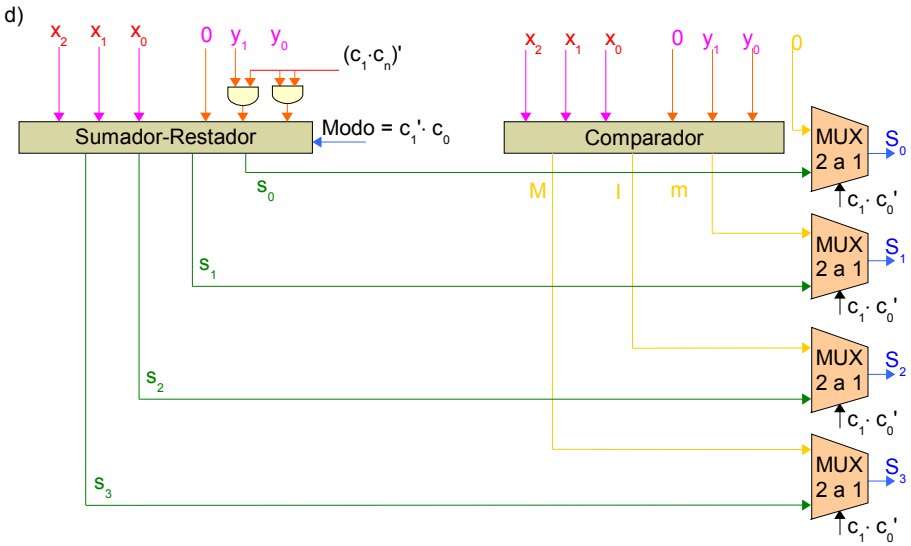


c)

X	Y	M I m
0 0 0	0 0 0	0 1 0
0 0 0	0 0 1	0 0 1
. . .	. . .	. . .
0 0 0	1 1 1	0 0 1
0 0 1	0 0 0	1 0 0
0 0 1	0 0 1	0 1 0
0 0 1	0 1 0	0 0 1
. . .	. . .	. . .
0 0 1	1 1 1	0 0 1
0 1 0	0 0 0	1 0 0
0 1 0	0 1 0	0 1 0
0 1 0	0 1 1	0 0 1
. . .	. . .	. . .



d)



✎ Utilizando el método de la *suma condicional*, la suma de 2 números binarios de 60 bits cada uno, ¿cuántos pasos necesita?.

1er paso: 60 palabras de longitud 1 bit.
2º paso: 30 palabras de longitud 2 bits.
3er paso: 15 palabras de longitud 4 bits. Atención a esta transición. Es necesario añadir 4 ceros a la izquierda.
4º paso: 8 palabras de longitud 8 bits.
5º paso: 4 palabras de longitud 16 bits.
6º paso: 2 palabras de longitud 32 bits.
7º paso: 1 palabras de longitud 64 bits.

✎ Utilizando el método de la suma condicional para sumar 2 números de 16 bits cada uno, ¿cuántos bits del resultado se conocen de forma correcta al finalizar el paso 4?.

1er paso: 16 palabras de longitud 1 bit. Ya se conoce 1 bit.
2º paso: 8 palabras de longitud 2 bits. Ya se conocen 2 bits.
3er paso: 4 palabras de longitud 4 bits. Ya se conocen 4 bits.
4º paso: 2 palabras de longitud 8 bits. Ya se conocen 8 bits.

✎ Sean dos números binarios de 4 bits representados en complemento a dos: M=0101 y N=1100. Lleve a cabo su multiplicación utilizando el *algoritmo de Booth modificado*.

A				N				N_{-1}		
A_3	A_2	A_1	A_0	N_3	N_2	N_1	N_0	N_{-1}		
0	0	0	0	1	1	0	0	0	valores iniciales	
0	0	0	0	1	1	0	0	0	no operar	1er ciclo
0	0	0	0	0	1	1	0	0	desplazar	
0	0	0	0	0	1	1	0	0	no operar	2º ciclo
0	0	0	0	0	0	1	1	0	desplazar	
1	0	1	1	0	0	1	1	0	A ← A - M	3er ciclo
1	1	0	1	1	0	0	1	1	desplazar	
1	1	0	1	1	0	0	1	1	no operar	4º ciclo
1	1	1	0	1	1	0	0	1	desplazar	

```
   M      0 1 0 1
   N x    1 1 0 0  (0)
-------------------------
0 0 0 0 0 0 0 0    0-0   1er ciclo
0 0 0 0 0 0 0      0-0   2º ciclo
1 1 0 1 1          1-0   3er ciclo
0 0 0 0            1-1   4º ciclo
-------------------------
1 1 0 1 1 0 0
```

Nótese que si los multiplicáramos utilizando algún algoritmo de *lápiz y papel*, no se estaría teniendo en cuenta que *N* es negativo y el resultado no sería el correcto.