Tema IV

Unidad aritmético-lógica

4.1 Sumadores binarios
- 4.1.1 Semisumador binario (SSB)
- 4.1.2 Sumador binario completo (SBC)
- 4.1.3 Sumador binario serie
- 4.1.4 Sumador binario paralelo con propagación del arrastre
- 4.1.5 Sumador-restador binario paralelo con propagación del arrastre

4.2 Sumadores de alta velocidad
- 4.2.1 Características de los arrastres
- 4.2.2 Sumadores con anticipación del arrastre

4.3 Sumadores en código BCD
- 4.3.1 Organización de los sumadores en código BCD

4.4 Multiplicadores binarios
- 4.4.1 Multiplicación de “lápiz y papel” de números sin signo
- 4.4.2 Mejoras en el algoritmo de “lápiz y papel”

4.7 Estructura de la unidad aritmético-lógica (ALU)
- 4.7.1 ALU´s integradas

4.9 Operaciones de desplazamiento
- 4.9.1 Clasificación de las operaciones de desplazamiento
- 4.9.2 Ejemplo: Diseño de un registro de desplazamiento de 4 bits
- 4.9.3 Estructura de los registros de desplazamiento

4.10 Operaciones de comparación
- 4.10.1 Utilizando un circuito combinacional
- 4.10.2 Utilizando un circuito secuencial
- 4.10.3 Utilizando un sumador

NO SON OBJETO DE EXAMEN

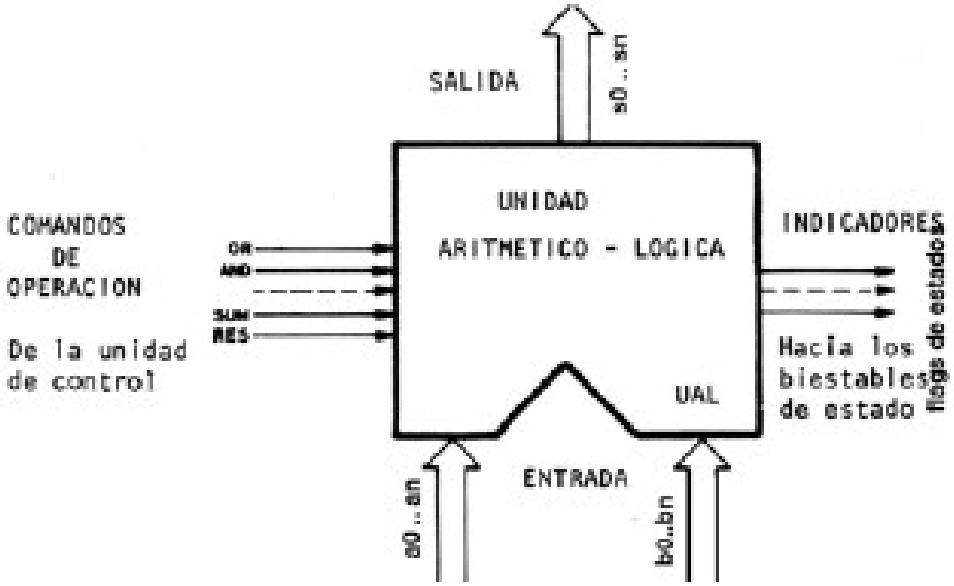
4.2.3 Sumadores de suma condicional
4.2.4 Sumadores con selección del arrastre
4.2.5 Sumadores con detección de la finalización del arrastre
4.2.6 Sumadores que minimizan el número de arrastres
4.2.7 Sumadores con arrastre almacenado
4.3.2 Restador en código BCD
4.4.3 Multiplicación en complemento a 2: Algoritmo de Booth
4.4.4 Ejemplo: Algoritmo de Booth
4.5 Multiplicadores de alta velocidad (completo)
4.6 Divisores binarios (completo)
4.8 Aritmética en coma flotante (completo)

La UAL

- Elemento que realiza las operaciones aritméticas y lógicas entre los datos.
- La unidad de control, memoria y Entrada/Salida, son las encargadas de suministrar los datos y de recibirlos una vez procesados.
- Los datos llegan a la UAL a través de los registros y los resultados que se generan también se almacenan en registros

Operaciones típicas

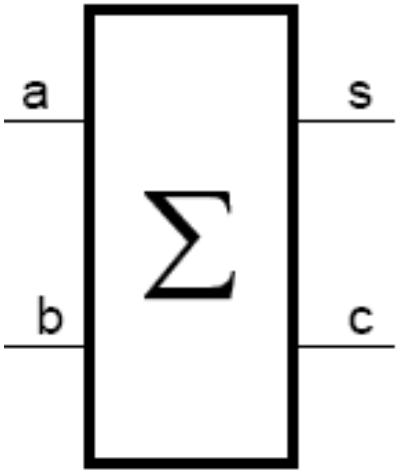
- Sumar
- Restar
- Multiplicar
- Desplazamiento de registros
- Comparaciones

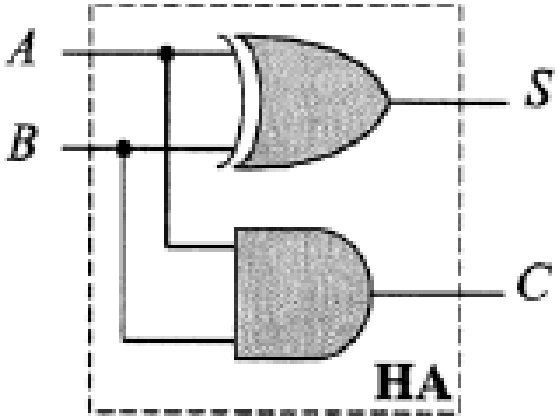


4.1 Sumadores binarios

- Un sumador binario se puede considerar como un conversor de código que recibe a la entrada dos números binarios *x* e *y* de *n* bits cada uno y produce una salida *s* de *n+1* bits que es la suma de los operandos

4.1.1 Semisumador binario (SSB)





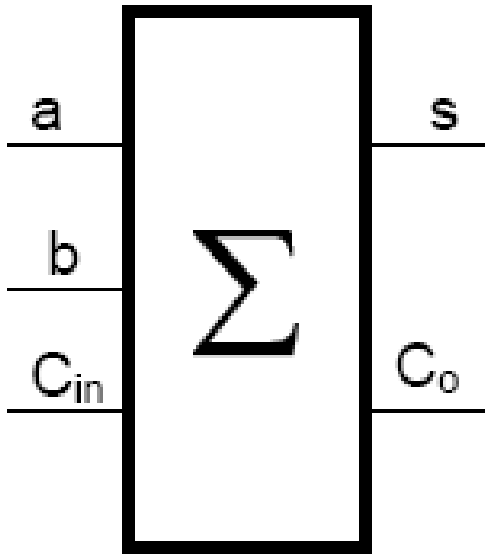
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

4.1.2 Sumador binario completo (SBC)

- Se diferencia del semisumador porque tiene una tercer entrada, C_{i-1}, de arrastre de las etapa anterior, que le permite encadenarse con otros SBC para el diseño de circuitos de suma de números de *n* bits (n>1).



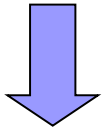
x_i	y_i	c_{i-1}	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 4.3: Tabla de verdad del SBC

$$s_i = \bar{x}_i\,\bar{y}_i\,c_{i-1} + \bar{x}_i\,y_i\,\bar{c}_{i-1} + x_i\,\bar{y}_i\,\bar{c}_{i-1} + x_i\,y_i\,c_{i-1}$$

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

Sacando factor común c_{i-1}



$$s_i = (\bar{x}_i\,\bar{y}_i + x_i\,y_i)\,c_{i-1} + (x_i\,\bar{y}_i + \bar{x}_i\,y_i)\,\bar{c}_{i-1} = x_i \oplus y_i \oplus c_{i-1}$$

$$c_i = x_i\,y_i + (x_i\,\bar{y}_i + \bar{x}_i\,y_i)\,c_{i-1} = x_i\,y_i + (x_i \oplus y_i)\,c_{i-1}$$

Sumador con dos semisumadores

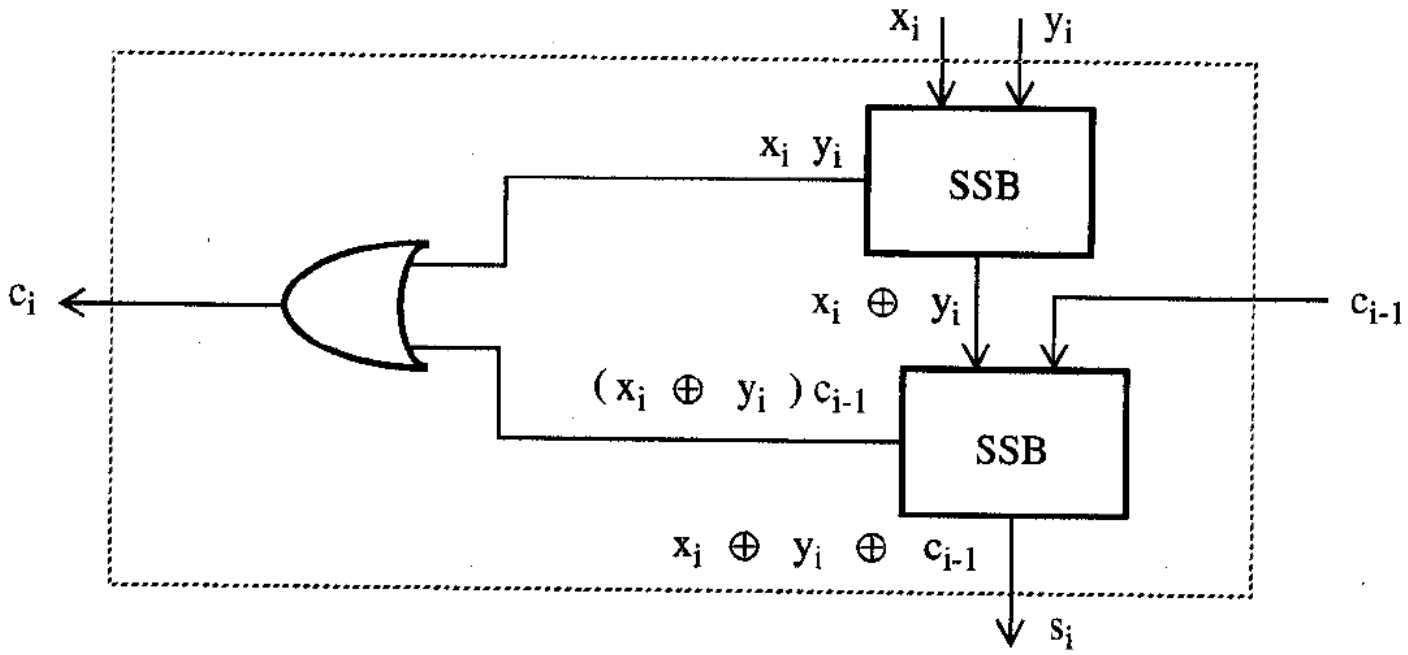
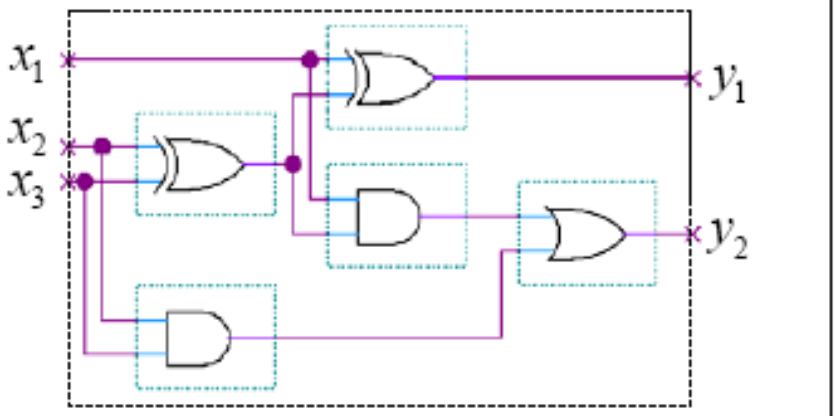


Figura 4.8: Realización de un SBC con 2 SSB´s

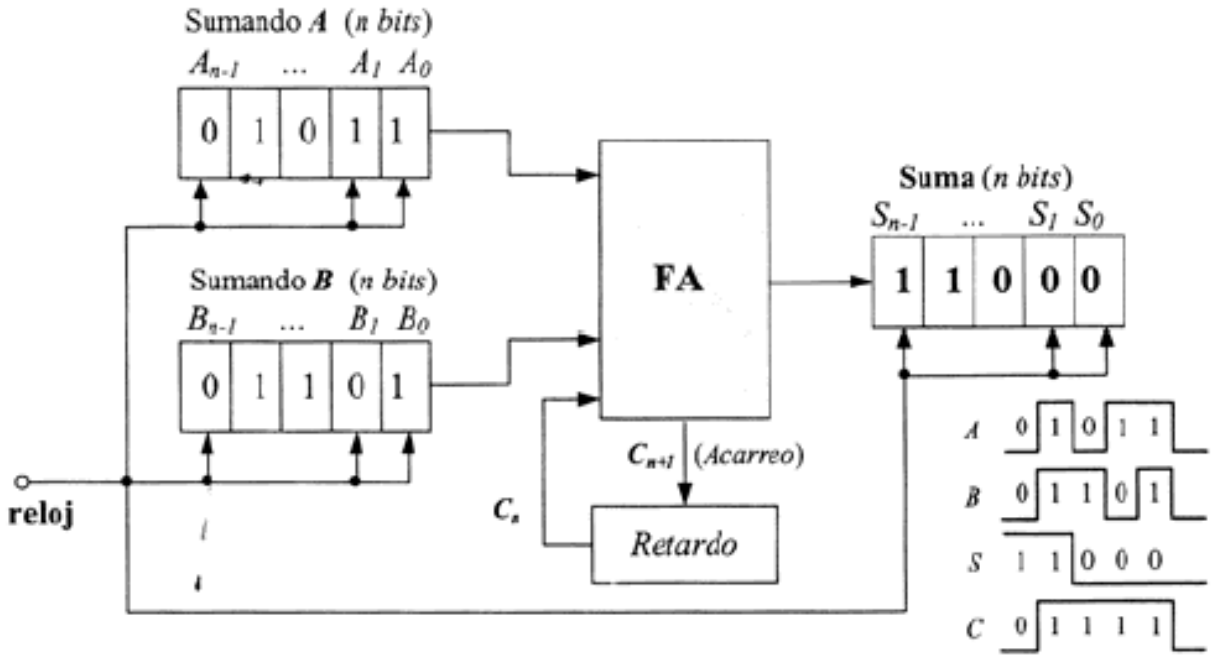
- En la figura se muestra un circuito lógico con tres entradas (x_1, x_2, x_3) y dos salidas (y_1, y_2). Indique cuál de las siguientes afirmaciones acerca de este circuito es correcta



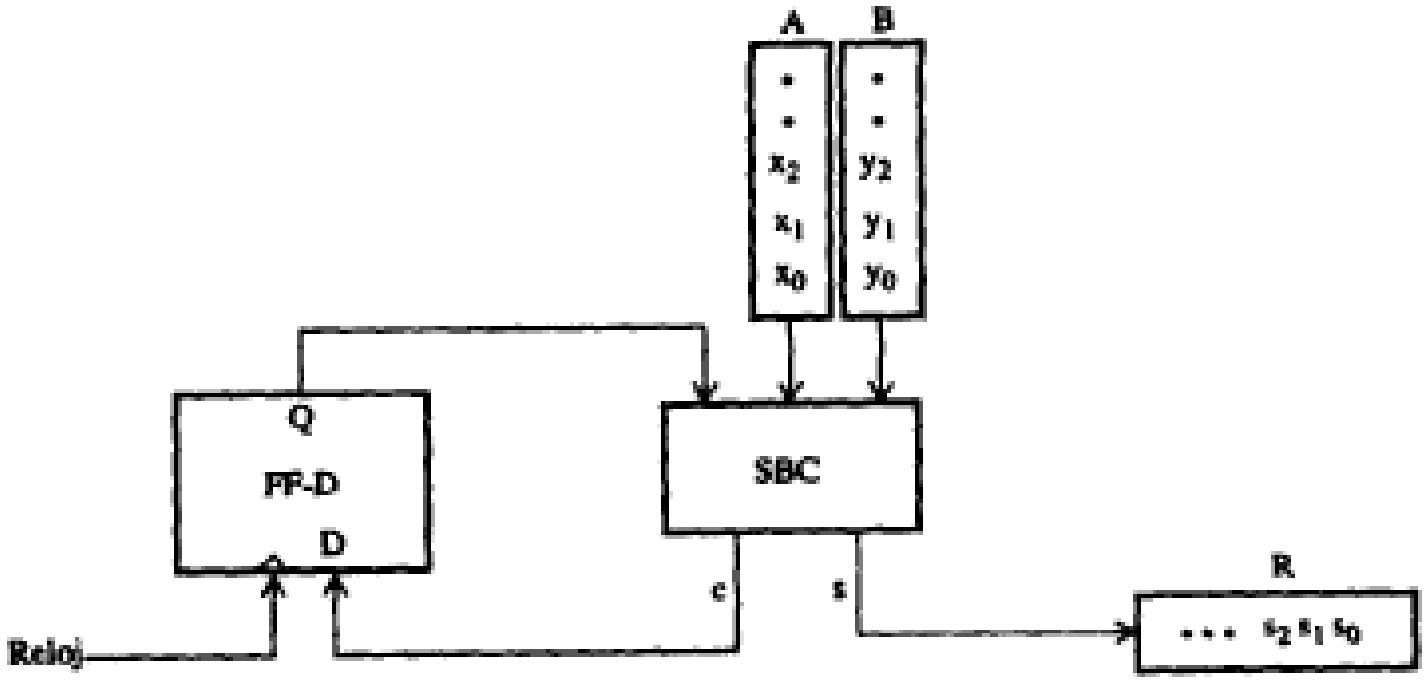
A) Es un elemento de memoria D.
B) Es un sumador binario completo.
C) Las dos anteriores son correctas.
D) Ninguna de las anteriores es correcta.

4.1.3 Sumador binario serie de *n* bits

- Con un SBC de 1 bit, dos registros y un elemento de memoria
- Complejidad es independiente del número de bits que hay que sumar
- El tiempo de operación crece linealmente con el número de bits *n*.

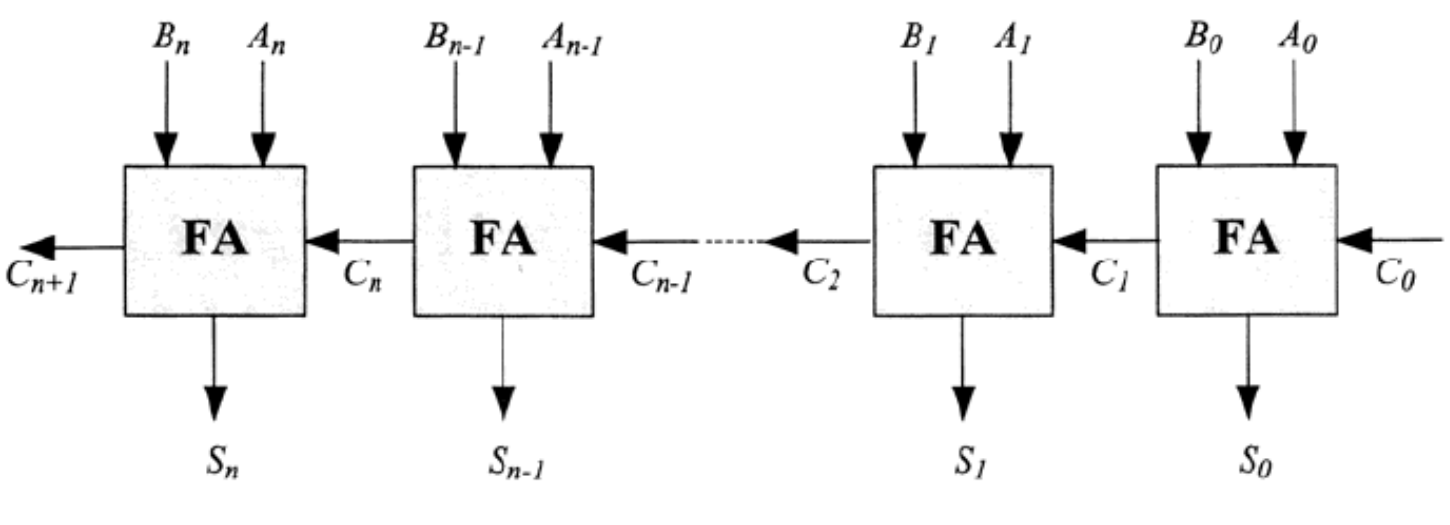


Sumador Binario Serie de n bits



4.1.4 Sumador binario paralelo con propagación del arrastre

- Mejora la velocidad de los sumadores binarios serie.
- Se conecta una cadena de SBC's de forma que se introduzcan en paralelo todos los bits de cada uno de los dos operandos.
- Para sumar n bits se encadenan n SBC's.



4.1.5 Sumador-restador binario paralelo con propagación del arrastre

- M=0 SUMA; M=1 Resta
- Para restar representamos en número en complemento a 2
  - Se complementan todos los bits (puerta XOR)
  - Se suma 1

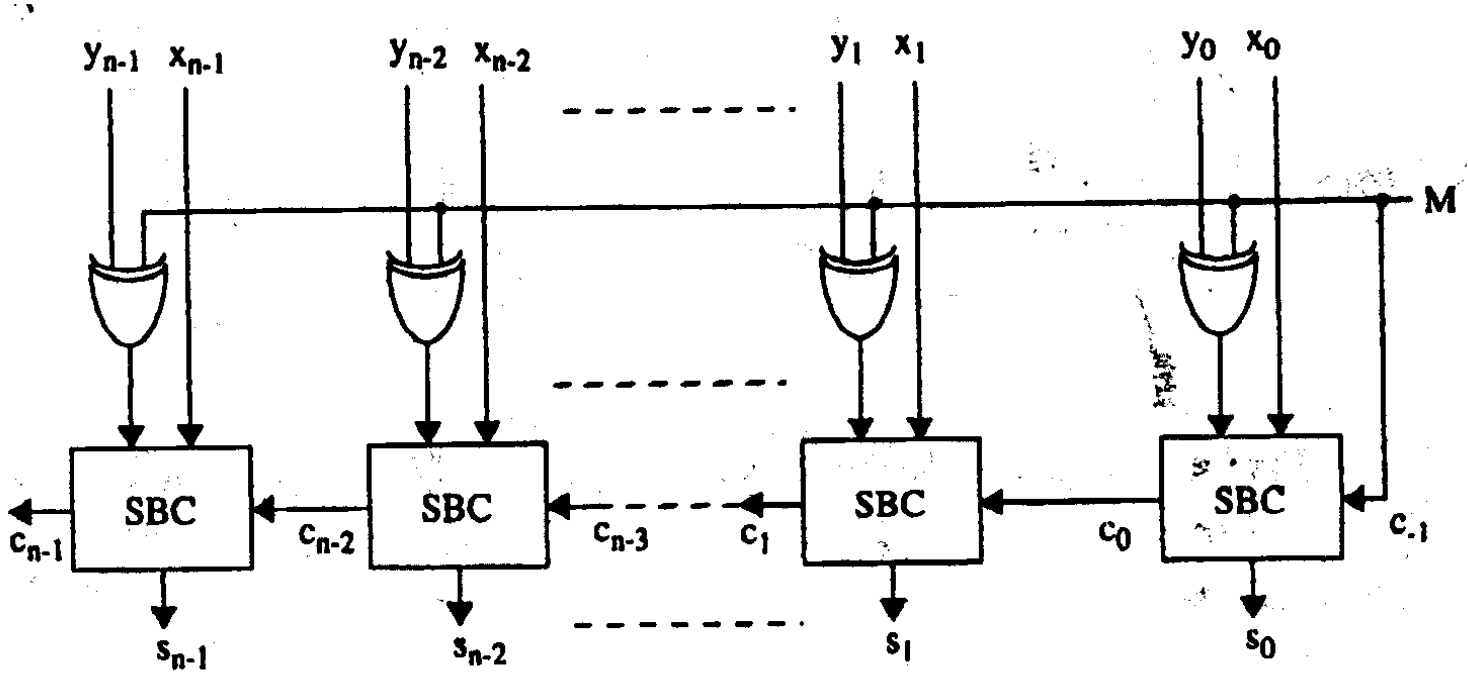
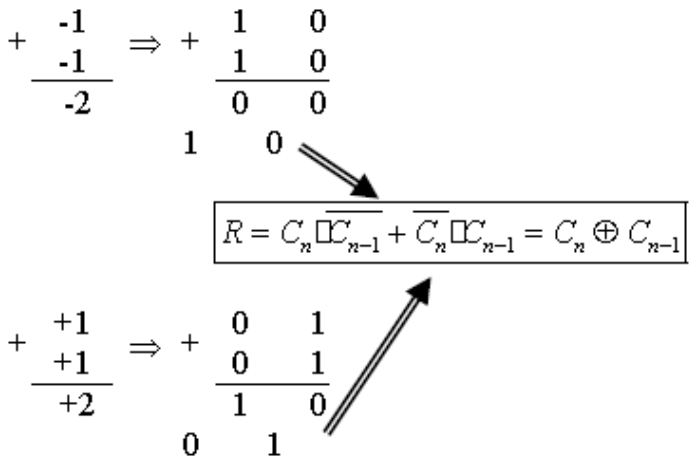


Figura 4-12 Sumador-restador binario paralelo con propagación de arrastre

Detección del rebose en el sumador-restador con propagación de arrastre

- ***Rebose***:
  - Efecto que se produce cuando se realiza una operación aritmética entre dos o más números, cuyo resultado es mayor a la capacidad de representación del sistema, interpretando de esta manera un error en el resultado
- Cuando se suman números con signo,
  - La suma de dos números de diferente signo no produce nunca rebose.
  - La suman dos números del mismo signo, el resultado puede producir rebose.



x_{n-1}	y_{n-1}	s_{n-1}	Rebose (R)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Tabla 4.5: Condiciones de rebose

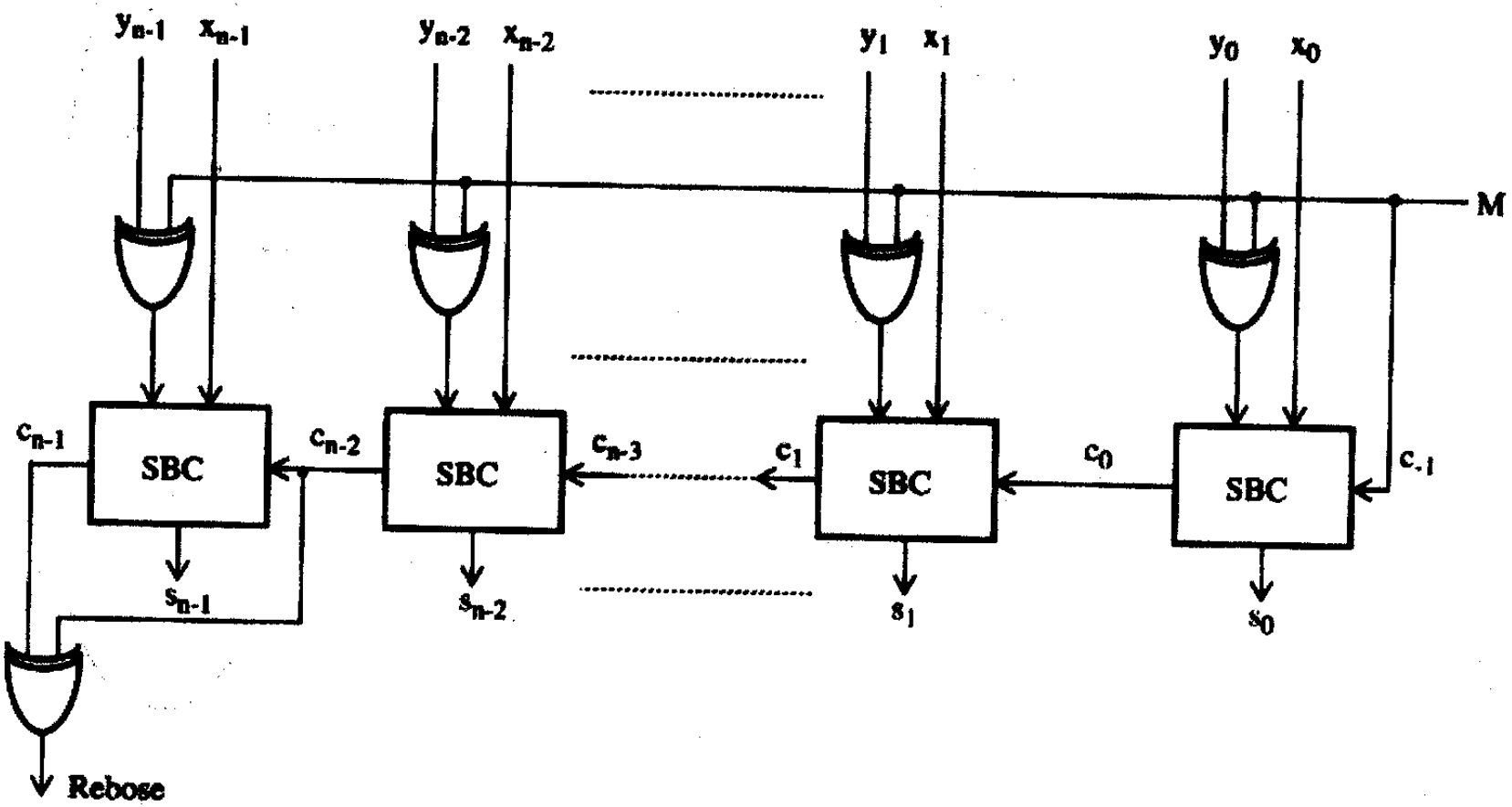
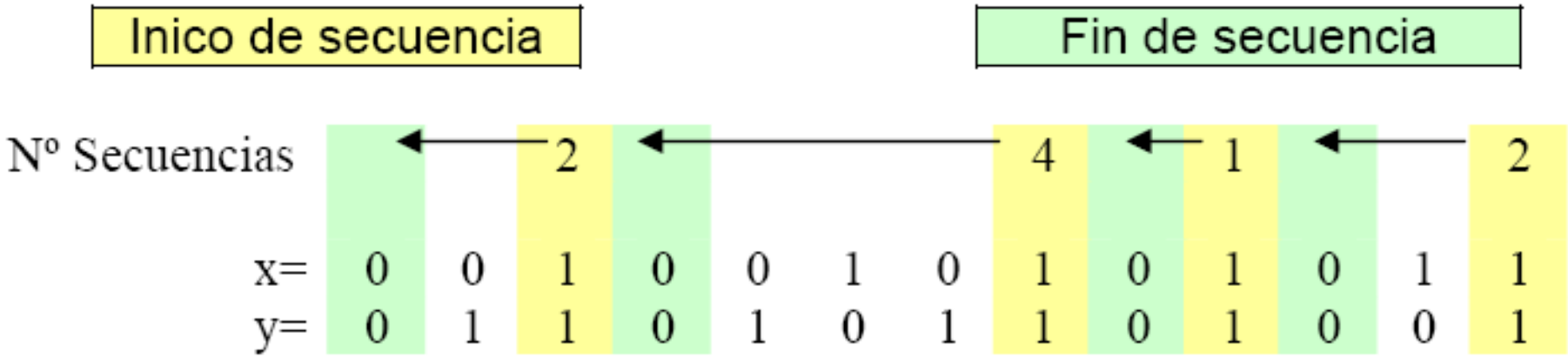


Figura 4.13: Sumador-restador binario paralelo con propagación del arrastre y detección del rebose

4.2 Sumadores de alta velocidad

	y = 0	y = 1
x = 0	---	p
x = 1	p	g

- **4.2.1 Características de los arrastres**
  - Un arrastre se generará en la posición *i-ésima* si $(x_i+y_i)>1$
  - Un arrastre se propagará de la posición *i-ésima* a la *(i+1)-ésima* si $(x_i+y_i)=1$
- **La SECUENCIAS DE ARRASTRE**
  - Se iniciarán en una suma será cuyos valores de entradas sean $x_i=y_i=1$
  - Continuarán a través de las etapas en las que $x_i \neq y_i$
  - Pararán cuando lleguen a una etapa en la que $x_i=y_i$



4.2.2 Sumadores con anticipación del arrastre

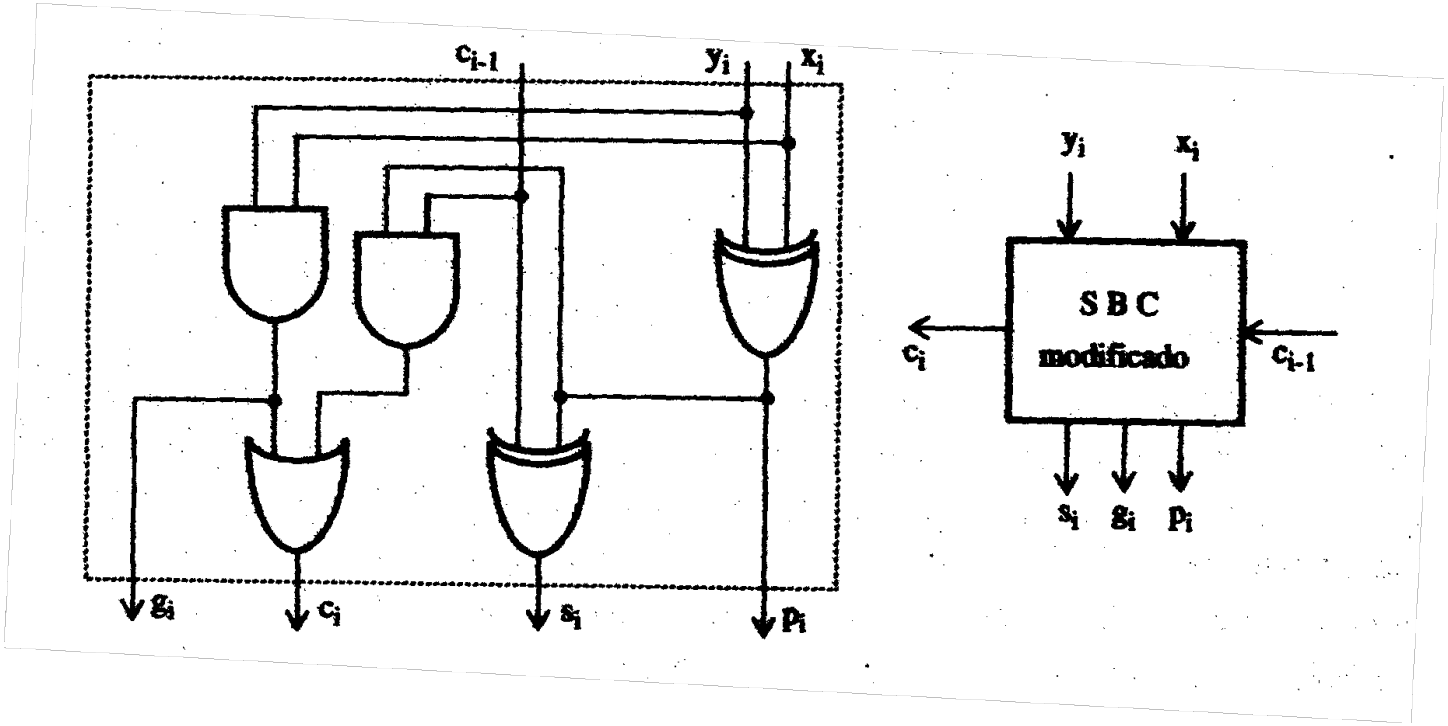
- Su principio básico es el de reducir el retardo producido por la propagación de los arrastres de los SBC's de menor peso a los de mayor peso.
- Generando la entrada de arrastre de la entrada i-ésima directamente a partir de los bits de entrada a las etapas precedentes

$$s_i = (\bar{x}_i\,\bar{y}_i + x_i y_i)\,c_{i-1} + (x_i \bar{y}_i + \bar{x}_i\, y_i)\,\bar{c}_{i-1} = x_i \oplus y_i \oplus c_{i-1}$$

$$c_i = x_i y_i + (x_i \bar{y}_i + \bar{x}_i\, y_i)\,c_{i-1} = x_i y_i + (x_i \oplus y_i)\,c_{i-1}$$

- Si llamamos a $x_i \cdot y_i = g_i$ (arrastre generado)
- Y a $x_i \oplus y_i = p_i$
- Las ecuaciones anteriores queda:

$$s_i = p_i \oplus c_{i-1}; \qquad c_i = g_i + c_{i-1} \cdot p_i$$



$$s_i = p_i \oplus c_{i-1}; \qquad c_i = g_i + c_{i-1} \cdot p_i$$

- Del la ecuación anterior se deduce que

$$c_0 = g_0 + p_0 c_{-1}$$

$$c_1 = g_1 + p_1 c_0 = g_1 + p_1 g_0 + p_1 p_0 c_{-1}$$

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

.....

$$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \ldots + p_i p_{i-1} \ldots p_1 g_0 + p_i p_{i-1} \ldots p_0 c_{-1}$$

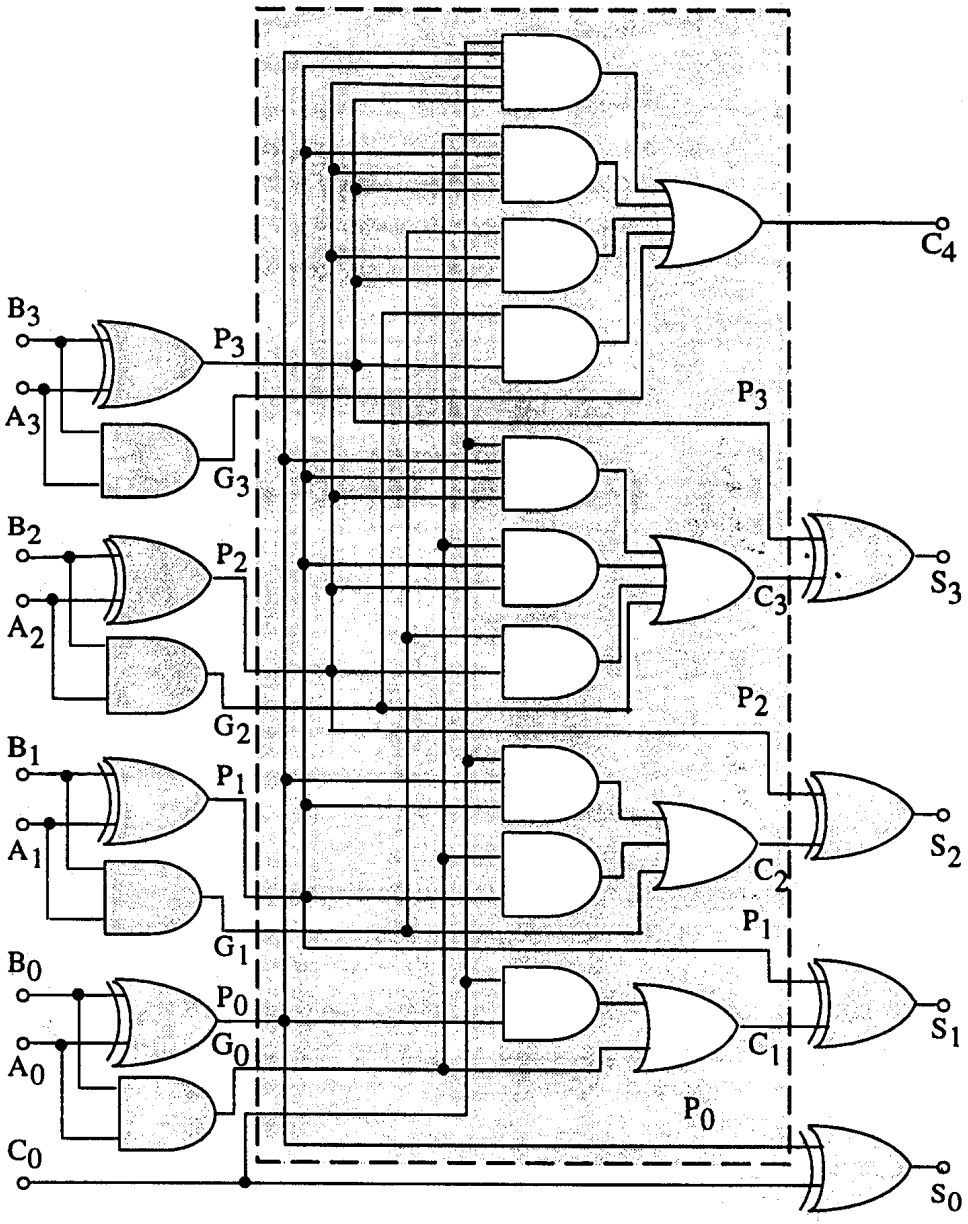


Fig. 5.6. Sumador paralelo de 4 bits con lógica de acarreo adelantado.

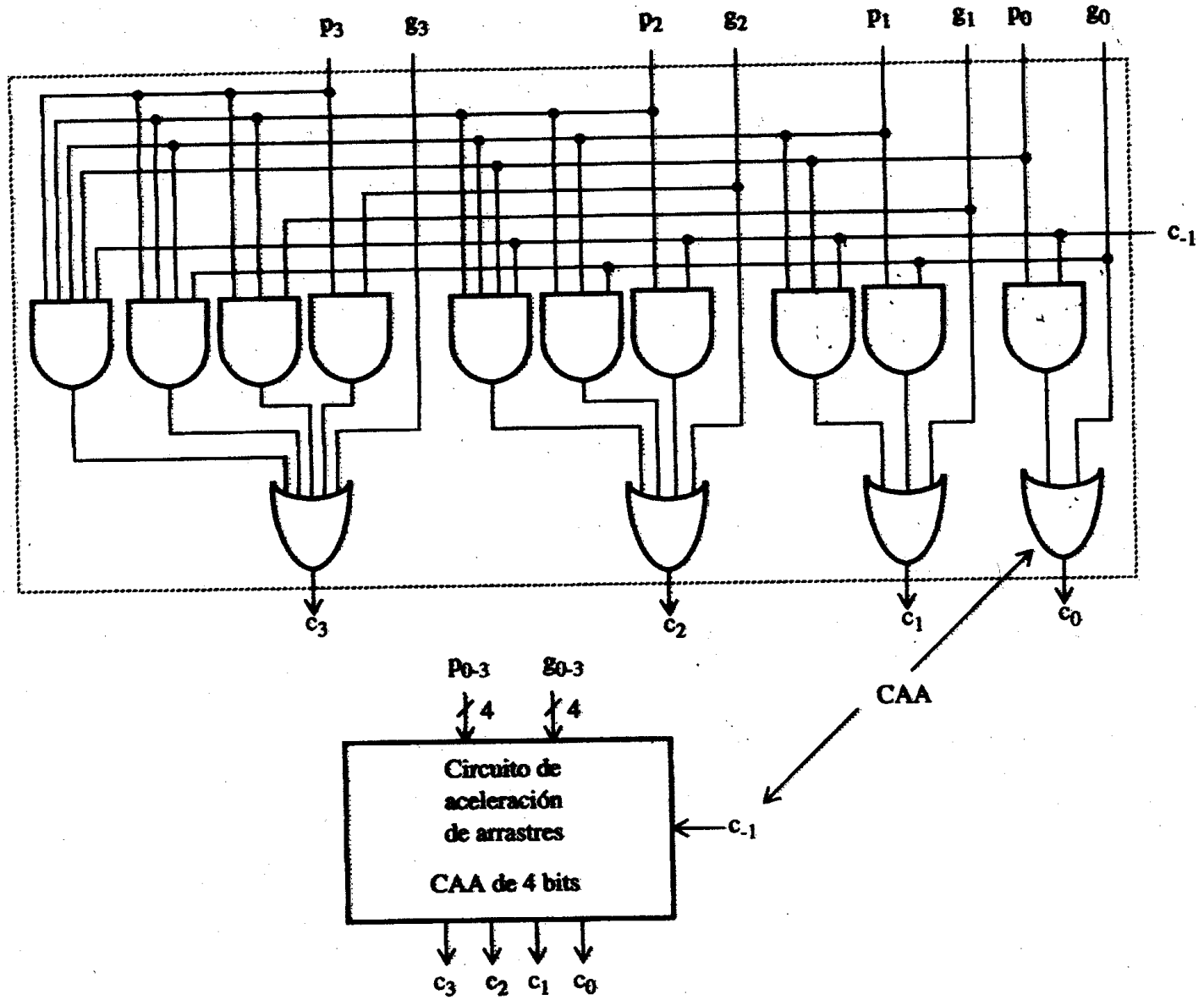
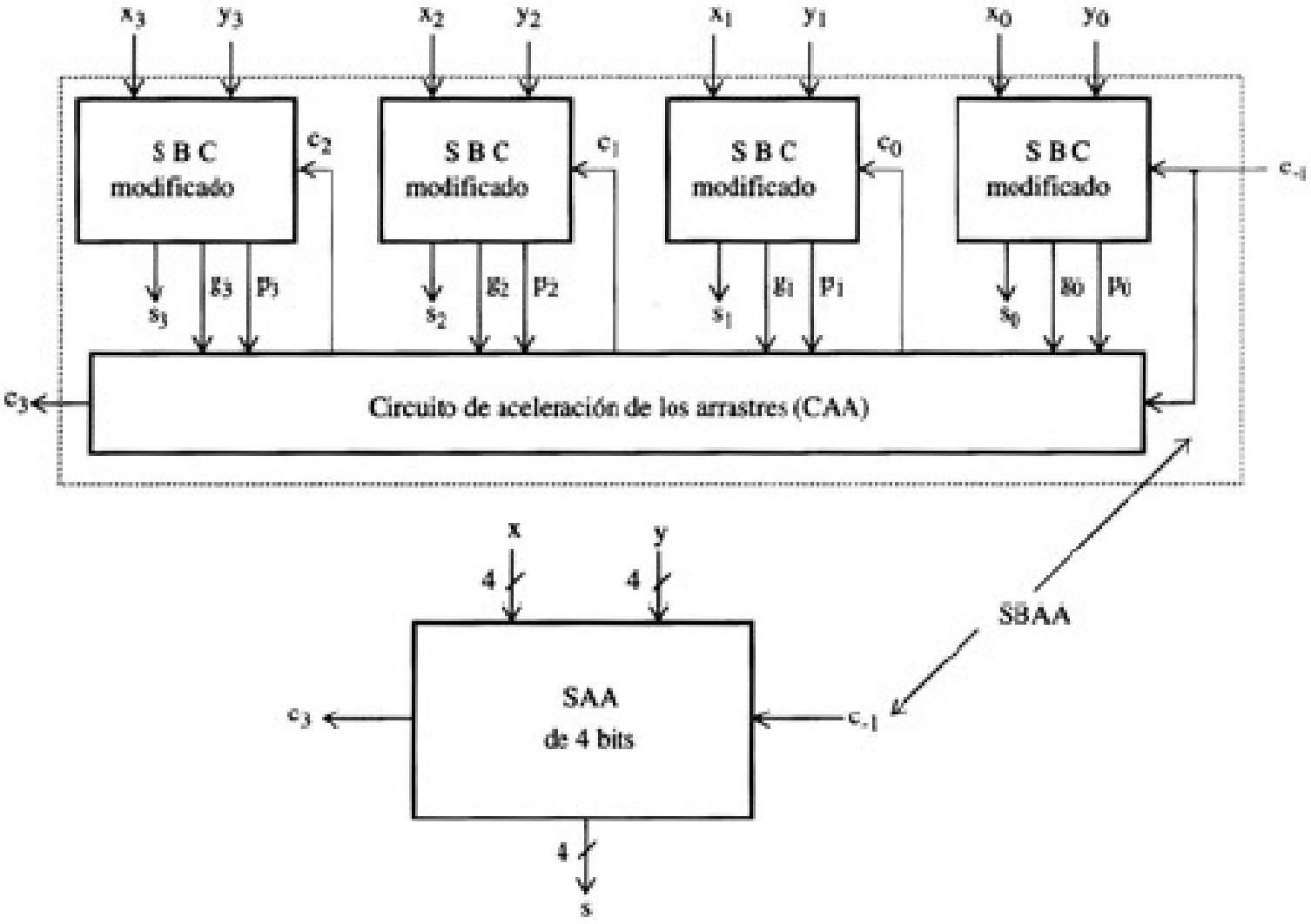
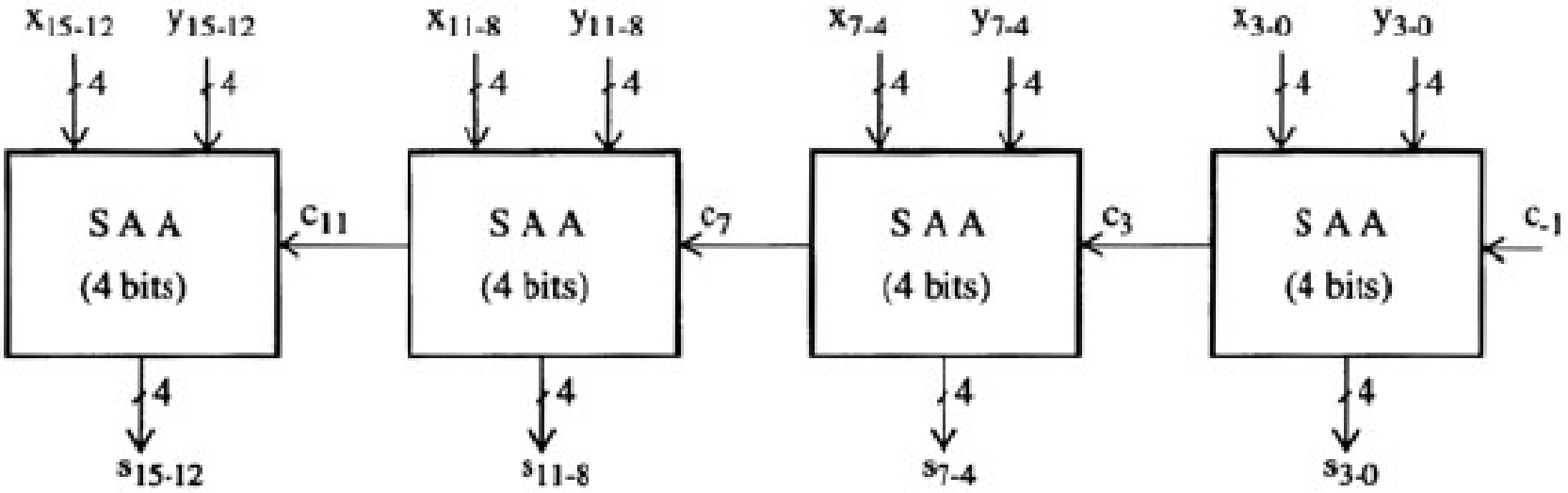


Figura 4.16: Circuito de aceleración de arrastres (CAA) de 4 bits

Sumador con aceleración de los 4 bits

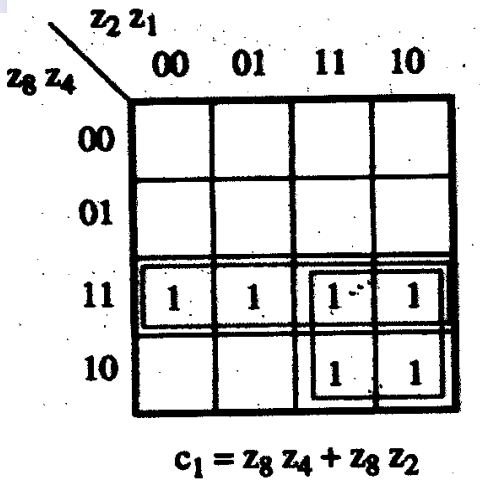


Sumador de 16 bits construido con 4 SAA de 4 bits



4.3 Sumadores en código BCD

- Se realizan como sumadores binarios naturales, añadiéndoles unos circuitos de corrección que garanticen la codificación de los resultados cuando el resultado de la suma sea mayor que 9.
- Si se suman dos dígitos BCD en un sumador binario de 4 bits el resultado es correcto si es menor que 10.
- Cuando es mayor o igual que 10 la suma no es correcta y debe ser corregida sumando 6 en BCD al resultado anterior.
- El circuito lógico que detecta cuando es necesario corregir el resultado (c = 1) se obtiene de la forma siguiente:
  - Si hay un arrastre en el primer sumador (k=1). Esto sucede cuando la suma de los dígitos BCD es mayor de 15. (suma > 15)
  - Si la suma está entre 10 y 15. Las configuraciones desde 1010 hasta 1111



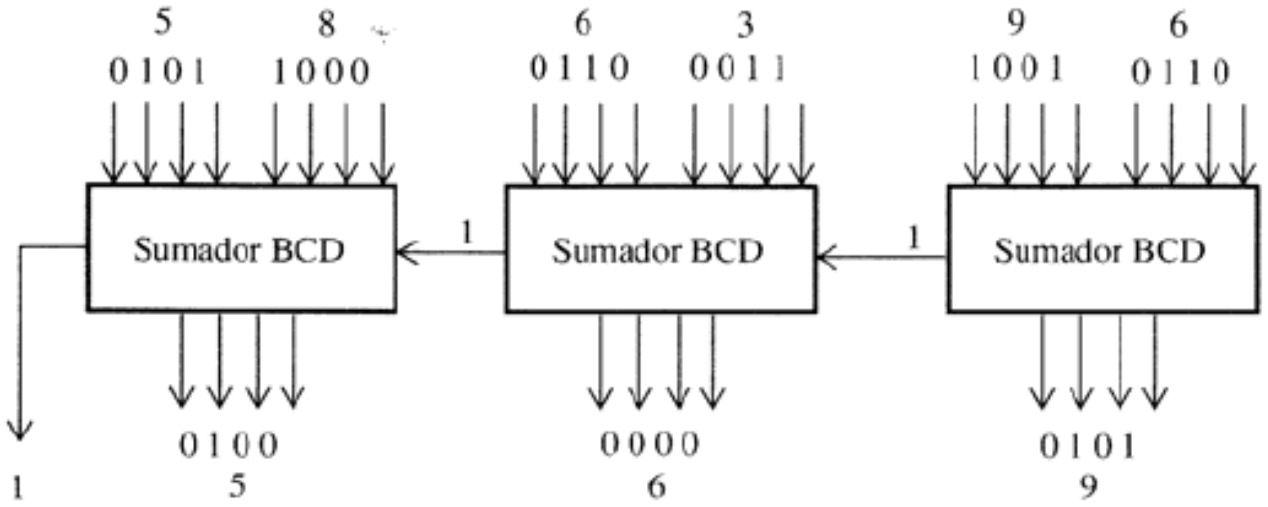
$c_1 = z_8 z_4 + z_8 z_2$

Sumador de dos dígitos en código BCD

$$c = k + c_1 = k + z_8 z_4 + z_8 z_2$$

4.3.1 Organización de los sumadores en código BCD

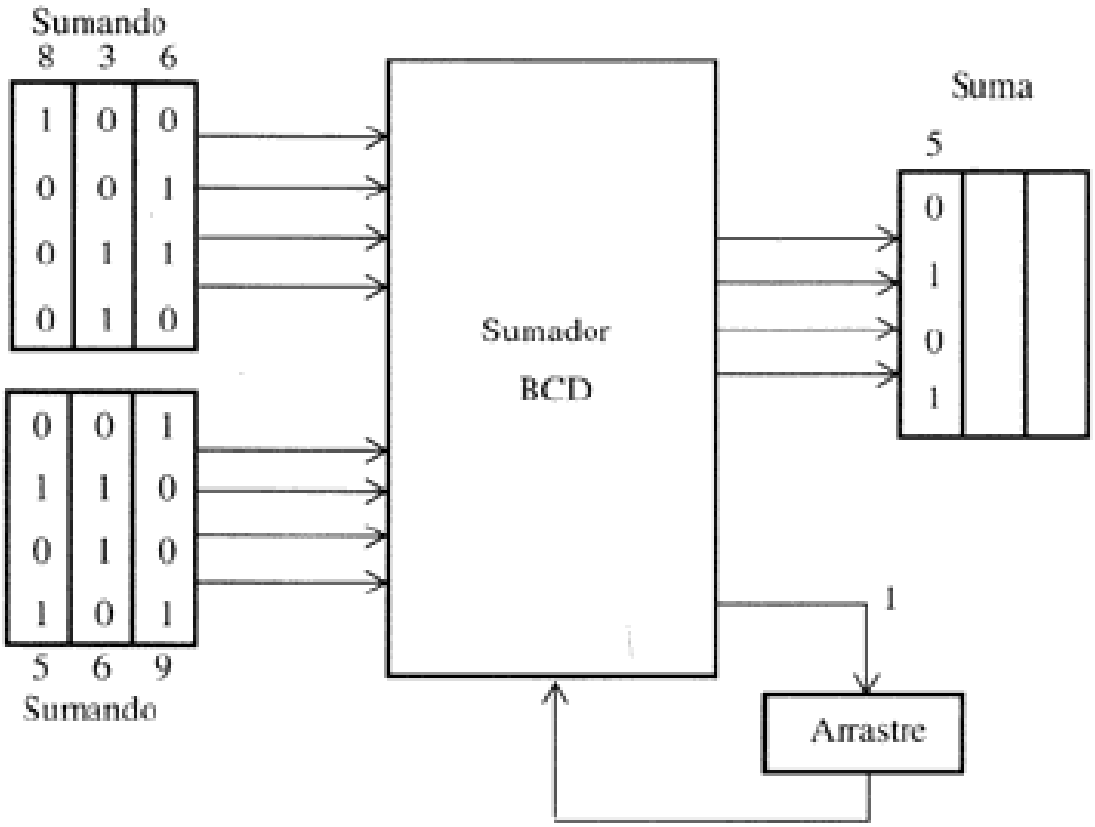
- *Sumador paralelo*:
  - Contiene n sumadores BCD.
  - El arrastre de salida de cada sumador se conecta al arrastre de entrada del siguiente.
  - La suma se genera en paralelo



Sumador BCD paralelo

Sumador dígito serie, bit paralelo

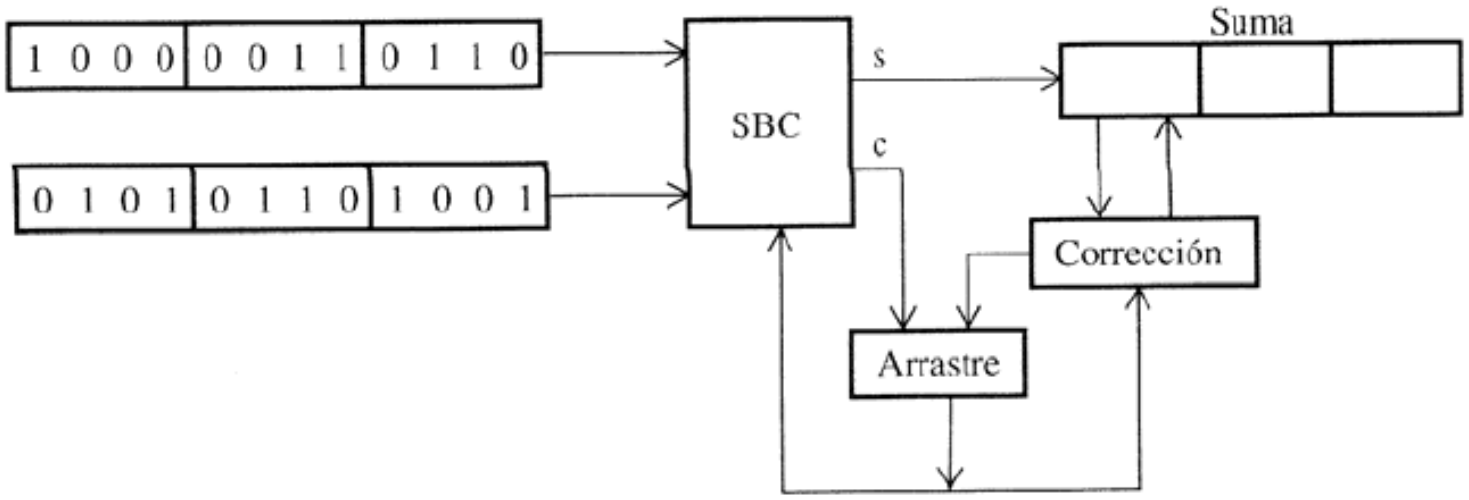
- Los dígitos se aplican en serie a un único sumador BCD,
- Los bits de cada dígito se aplican en paralelo



Sumador BCD dígito-serie, bit-paralelo

Sumador dígito serie, bit serie

- Es la más lenta, los bits se desplazan de uno en uno a través de un SBC.
- La suma binaria después de cuatro desplazamientos debe ser corregida para obtener un dígito válido en BCD.

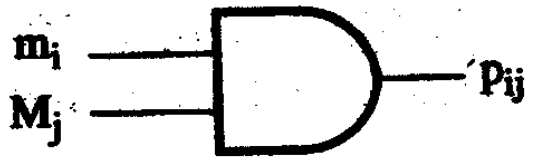


Sumador BCD dígito-serie, bit-serie

4.4 Multiplicadores binarios

4.4.1 Multiplicación de “lápiz y papel” de números sin signo

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



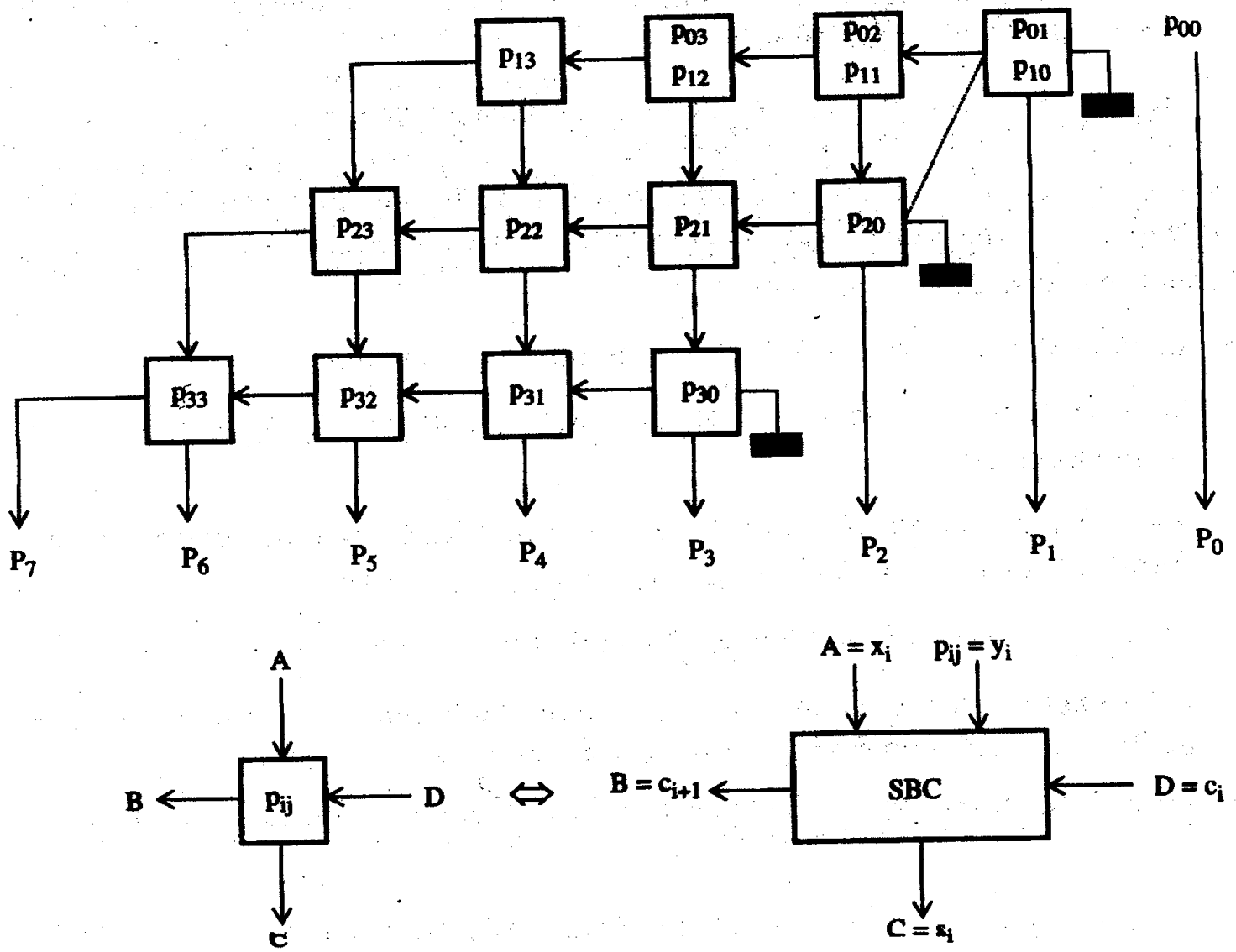


Figura 4.24. Multiplicador combinacional de 4 × 4 bits compuesto por 12 SBC's

4.4.2 Mejoras en el algoritmo de "lápiz y papel"

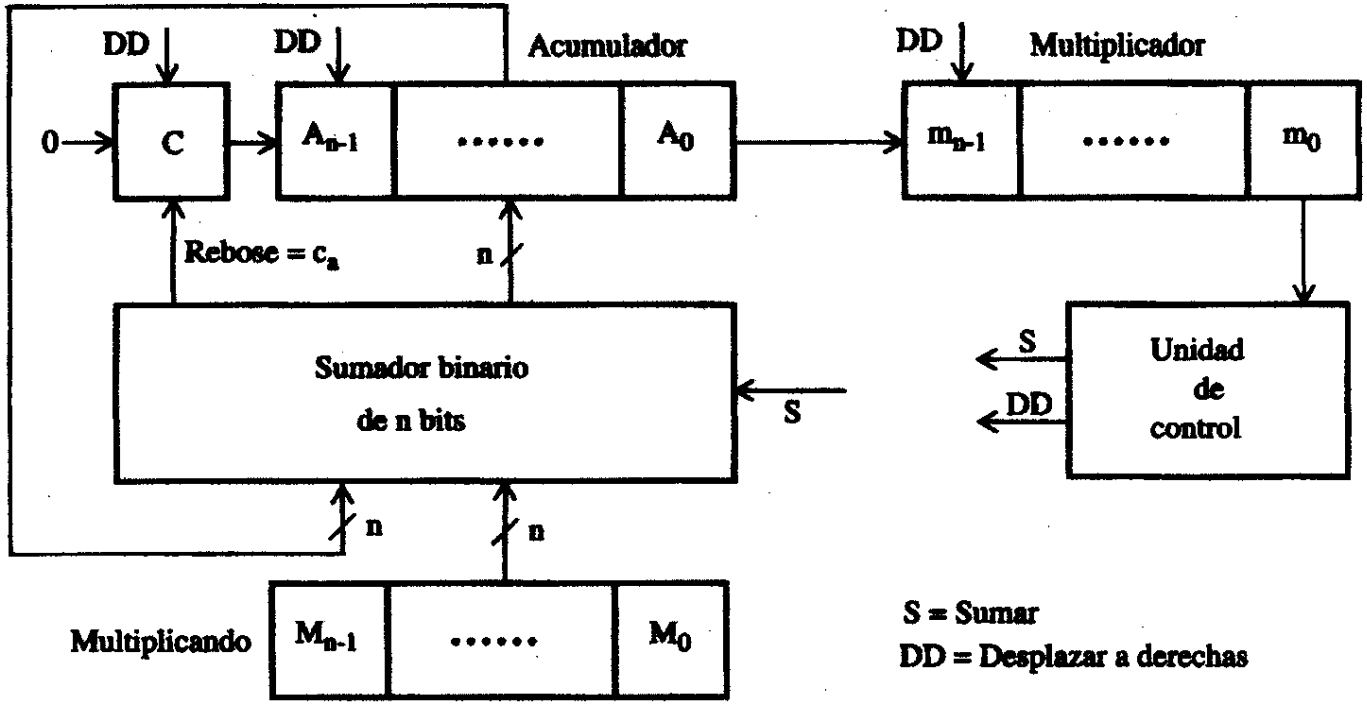


Figura 4.35: Esquema de un multiplicador que mejora el algoritmo de "lapiz y papel"

C	A	m			
0	0000	1011	Valores iniciales		
0	1001	1011	S	1er ciclo	
0	0100	1101	DD		
0	1101	1101	S	2º ciclo	
0	0110	1110	DD		
0	0011	0111	DD	3er ciclo	
0	1100	0111	S	4º ciclo	(Producto en A, m)
0	0110	0011	DD		

Ejemplo numérico del algoritmo de "lapiz y papel mejorado" (M contiene 1001)

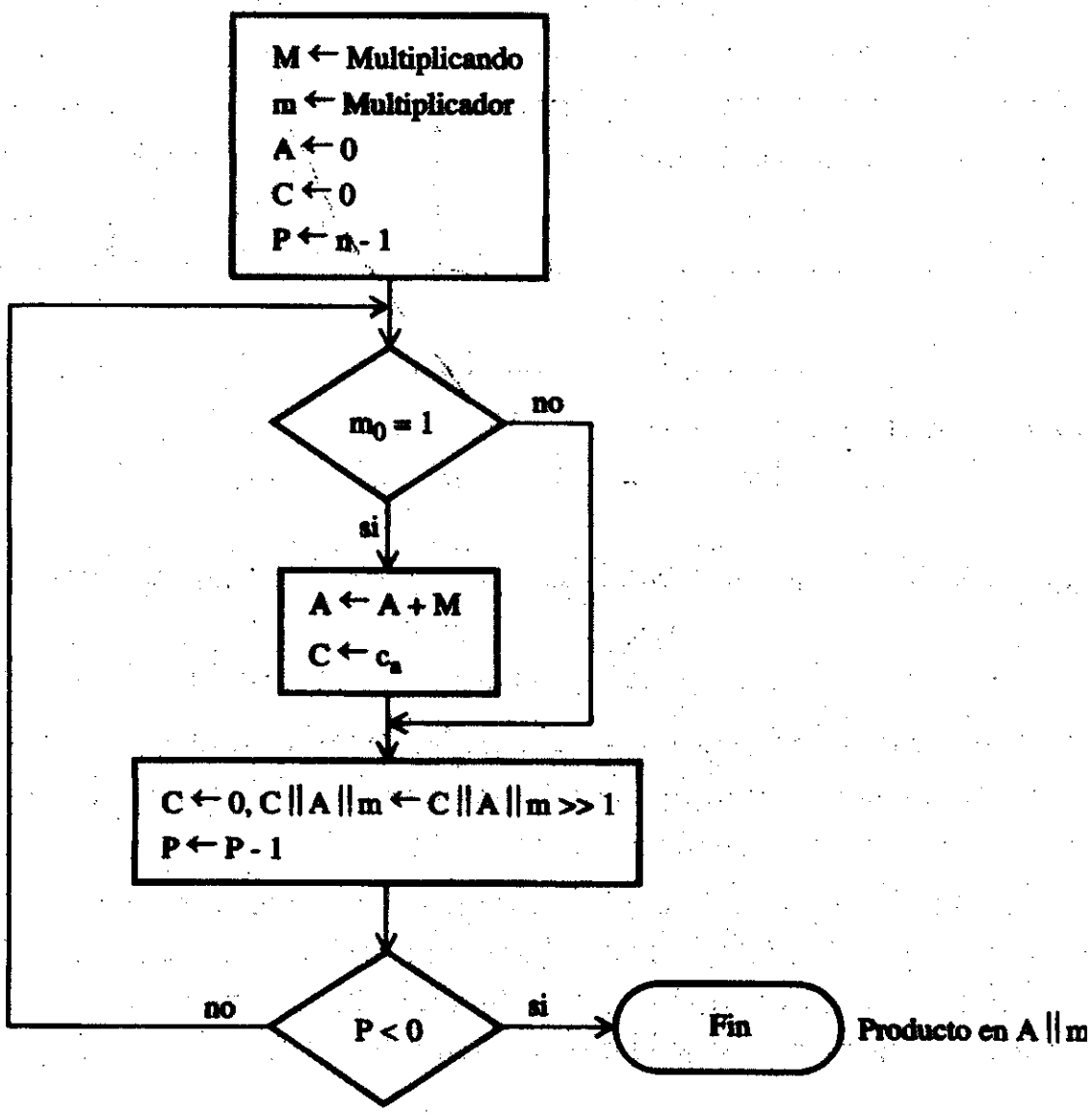
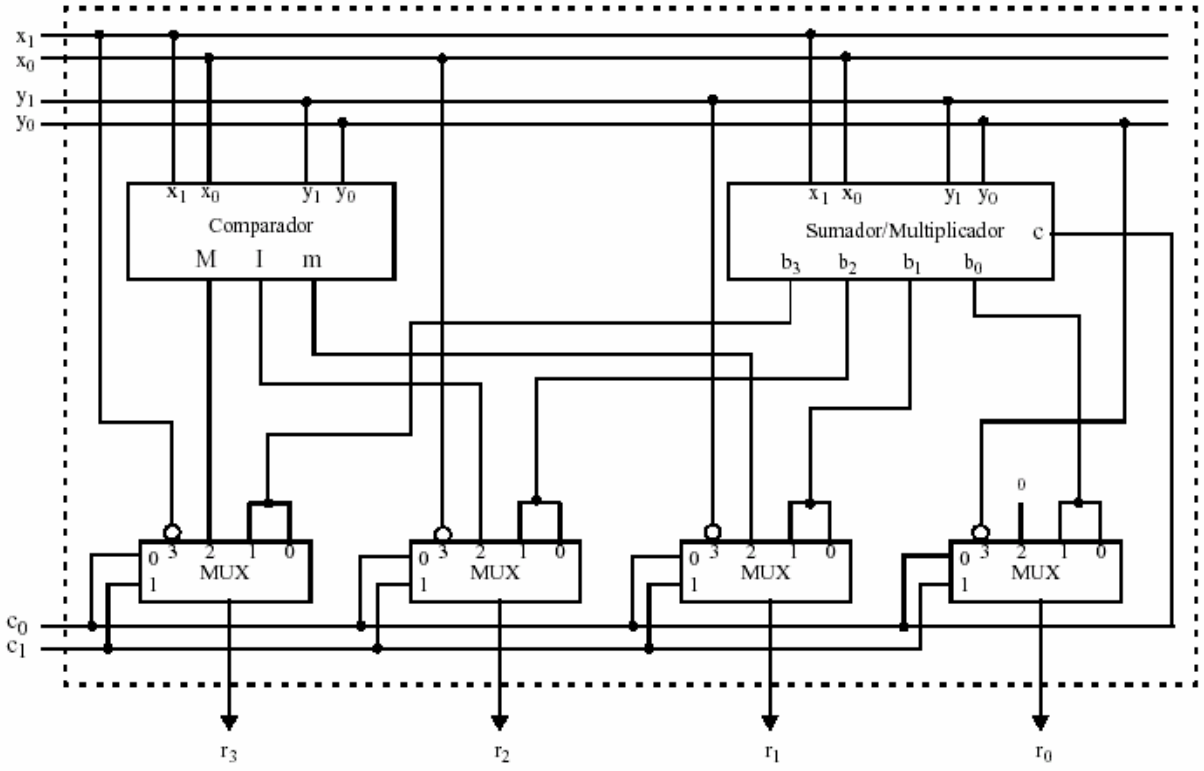


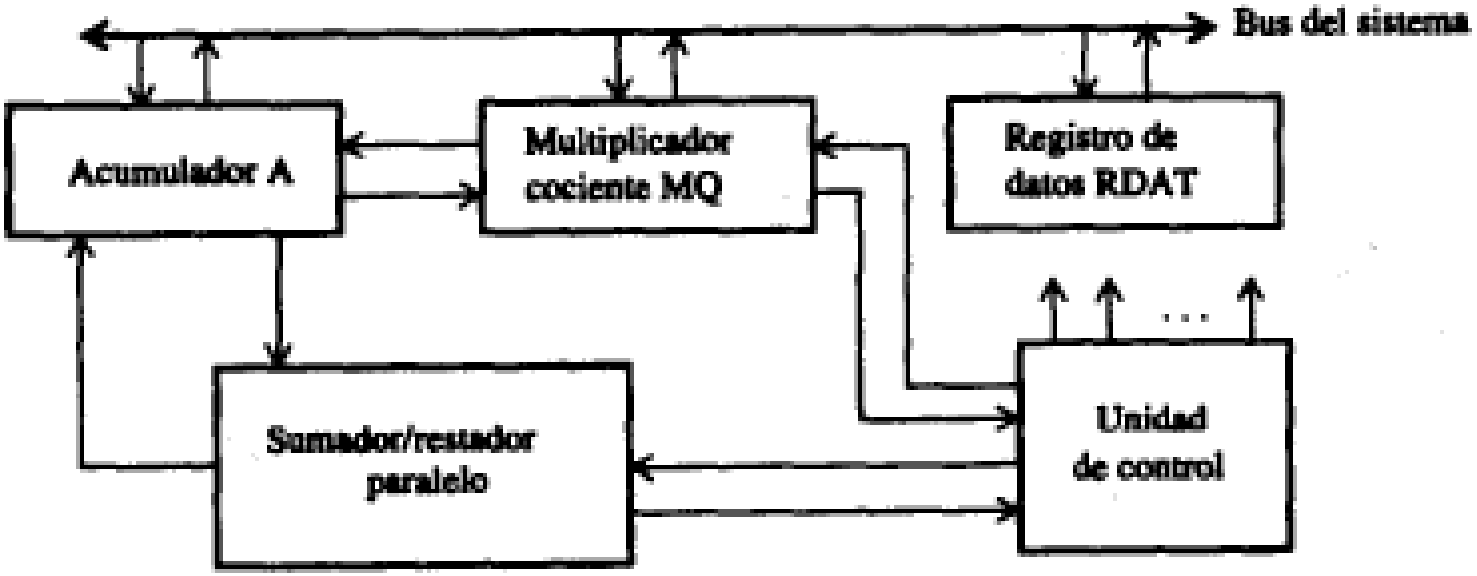
Figura 4.37: Diagrama de flujo de la multiplicación de números binarios sin signo

4.7 Estructura de la unidad aritmético-lógica (ALU)

- Complejidad de la UAL viene impuesta por los tipos de operaciones que puede efectuar y la forma que las ejecuta.
- Si se usan algoritmos análogos para diferentes operaciones se puede simplificar su diseño.

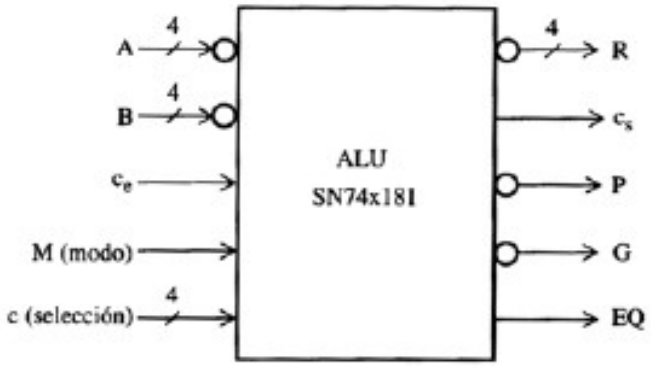
La estructura básica de una unidad lógica aritmética suele consistir en utilizar multiplexores con tantas entradas como operaciones queremos que realice dicha ALU y en cada entrada colocar el circuito que ha de realizar la operación correspondiente




Bus del sistema
Acumulador A
Multiplicador cociente MQ
Registro de datos RDAT
Sumador/restador paralelo
Unidad de control

4.7.1 ALU´s integradas

ALU SN74181



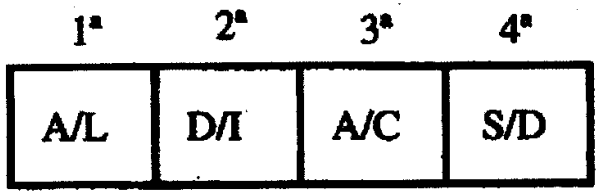
Selección $c_3c_2c_1c_0$	M = 1 Función lógica	M = 0 Función aritmética v(R) = s mod 16
0 0 0 0	$R = \overline{A}$	$s = v(A) - 1 + c_e$
0 0 0 1	$R = \overline{A} \vee \overline{B}$	$s = v(A \wedge B) - 1 + c_e$
0 0 1 0	$R = \overline{A} \vee B$	$s = v(A \wedge \overline{B}) - 1 + c_e$
0 0 1 1	R = 1 1 1 1	$s = 1111 + c_e$
0 1 0 0	$R = \overline{A} \wedge \overline{B}$	$s = v(A) + v(A \vee \overline{B}) + c_e$
0 1 0 1	$R = \overline{B}$	$s = v(A \wedge B) + v(A \vee \overline{B}) + c_e$
0 1 1 0	$R = A \oplus \overline{B}$	$s = v(A) - v(B) - 1 + c_e$
0 1 1 1	$R = A \vee \overline{B}$	$s = v(A \vee \overline{B}) + c_e$
1 0 0 0	$R = \overline{A} \wedge B$	$s = v(A) + v(A \vee B) + c_e$
1 0 0 1	$R = A \oplus B$	$s = v(A) + v(B) + c_e$
1 0 1 0	R = B	$s = v(A \wedge \overline{B}) + v(A \vee B) + c_e$
1 0 1 1	$R = A \vee B$	$s = v(A \vee B) + c_e$
1 1 0 0	R = 0 0 0 0	$s = v(A) + v(A) + c_e$
1 1 0 1	$R = A \wedge \overline{B}$	$s = v(A \wedge B) + v(A) + c_e$
1 1 1 0	$R = A \wedge B$	$s = v(A \wedge \overline{B}) + v(A) + c_e$
1 1 1 1	R = A	$s = v(A) + c_e$

Tabla 4.15: Funciones de la ALU SN74181

4.9 Operaciones de desplazamiento

4.9.1 Clasificación de las operaciones de desplazamiento

- Tratamiento del bit de signo
  - Aritméticos (A) → No afecta al signo
  - Lógicos (L) → Interviene el signo
- Sentido de desplazamiento
  - Derecha (D)
  - Izquierda (I)
- Tratami. Bits que rebosan
  - Abierto (A) se pierden los bits que rebosan
  - Cerrado (C) interviene el bt de rebose
- Longitud de registros
  - Simples (S) → Registro único
  - Dobles (D) → Pareja de registros



1ª letra:	A: Algebraico	L: Lógico
2ª letra:	D: Derecha	I: Izquierda
3ª letra:	A: Abierto	C: Cerrado
4ª letra:	S: Simple	D: Doble

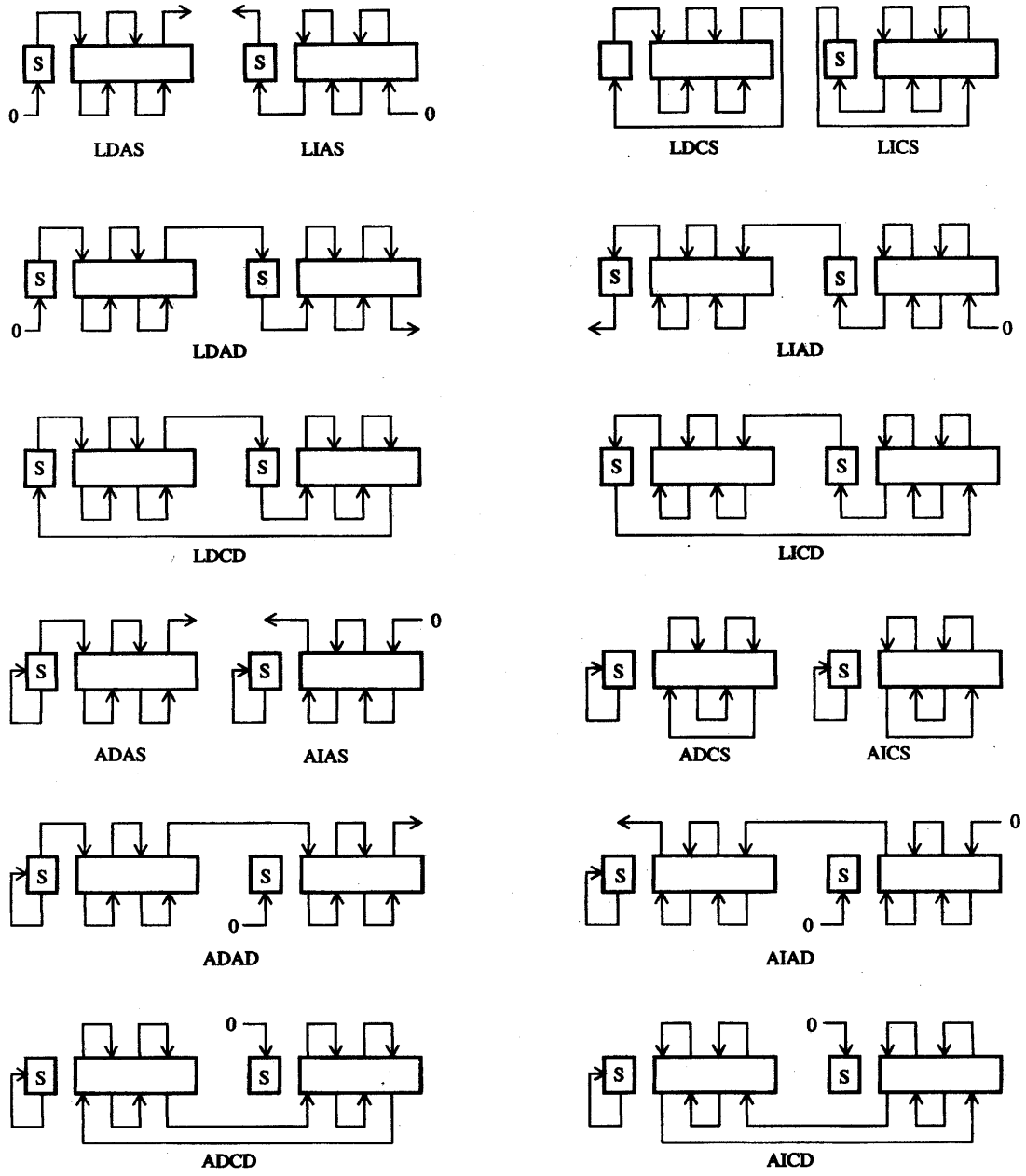


Figura 4.70: Representación gráfica de las operaciones de desplazamiento

- El contenido inicial de un registro de desplazamiento de 4 bits es 0111. Entonces, se realiza una operación de desplazamiento, tras lo cual el contenido del registro pasa a ser 0110. Señale cuál de las afirmaciones siguientes es correcta.

A) La operación de desplazamiento podría ser AIAS.

B) La operación de desplazamiento podría ser AICS.

C) Las dos anteriores son correctas.

D) Ninguna de las anteriores.

Respuesta

- Véase el apartado 4.9.1 del texto base de teoría. El contenido inicial del registro de desplazamiento es 0111. Tras la operación de desplazamiento pasa a ser 0110. La respuesta A, "*la operación de desplazamiento podría ser AIAS*", es correcta. El bit más significativo se mantiene, y se produce un desplazamiento hacia la izquierda de los tres bits menos significativos, introduciendo un 0 en el bit menos significativo.
- La respuesta B, "*la operación de desplazamiento podría ser AICS*", es falsa. En caso de haberse producido este desplazamiento, los tres bits menos significativos deberían seguir siendo 1, cosa que no sucede.
- Por lo anterior, las respuestas C y D son falsas.
- Respuesta: A (La operación de desplazamiento podría ser AIAS)

- Diseñar un registro de desplazamiento de 4 bits que sea capaz de realizar los desplazamientos indicados en la siguiente tabla de la verdad

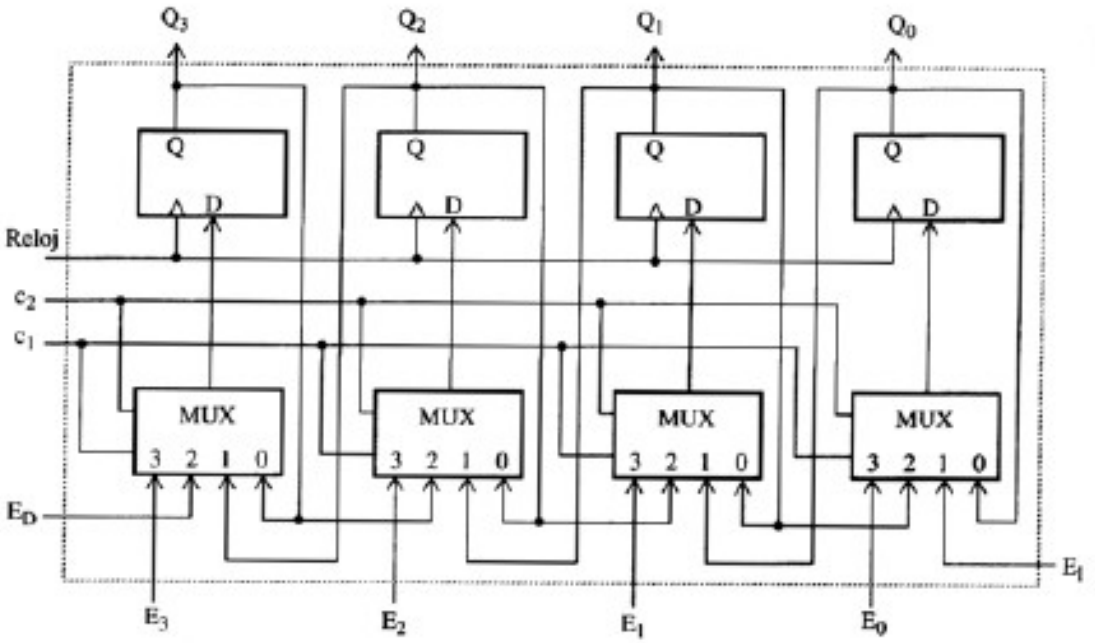
La descripción de la salida del circuito es la siguiente:

$$Q(t+1) = \begin{cases} Q(t) & \text{si Control = NOP (no operación)} \\ E(t) & \text{si Control = CARGA} \\ (Q_2, Q_1, Q_0, E_I) & \text{si Control = LIAS} \\ (E_D, Q_3, Q_2, Q_1) & \text{si Control = LDAS} \end{cases}$$

Operación	c_2	c_1
NOP	0	0
LIAS	0	1
LDAS	1	0
CARGA	1	1

Tabla 4.19: Codificación de las entradas de control

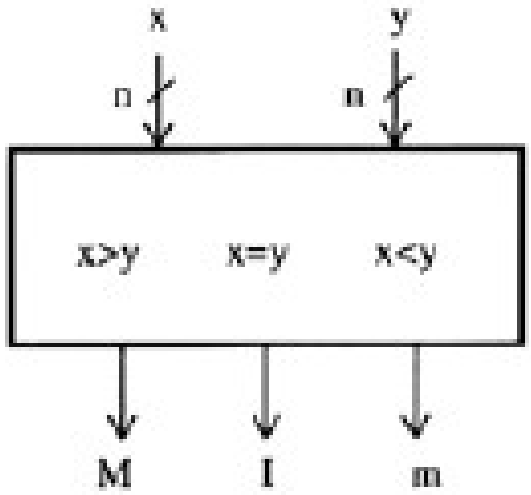
Operación	c_2	c_1
NOP	0	0
LIAS	0	1
LDAS	1	0
CARGA	1	1



4.10 Operaciones de comparación

- Son elementos que en base a introducirle dos números de n bits (x , y) entregan a su salida mediante tres señales el valor de la comparación M (x>y), I (x=y) y m (x<y).
- Un circuito comparador se puede realizar de tres formas:
  - Utilizando un circuito combinacional
  - Utilizando un circuito secuencial
  - Utilizando un sumador

$$M=1 \quad I=0 \quad m=0 \quad \text{si } x>y$$
$$M=0 \quad I=1 \quad m=0 \quad \text{si } x=y$$
$$M=0 \quad I=0 \quad m=1 \quad \text{si } x<y$$



4.10.1 Utilizando un circuito combinacional

- Comparador de un bit

expresiones booleanas para *M*, *I* y *m*:

$$M = x\,\overline{y}$$

$$I = \overline{x}\,\overline{y} + x\,y = \overline{x\,\overline{y} + \overline{x}\,y}$$

$$m = \overline{x}\,y$$

x	y	M	I	m
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Tabla de verdad de un comparador de 1 bit

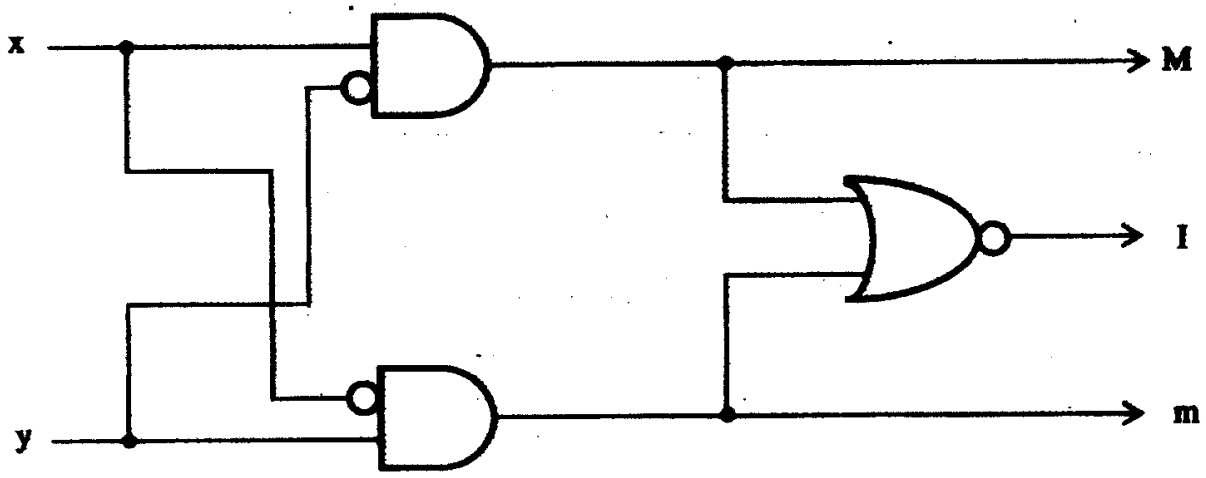


Figura 4.77: Circuito lógico de un comparador de 1 bit

Comparador combinacional para n bits

$x = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ $y = (y_{n-1}, y_{n-2}, \ldots, y_1, y_0)$

$M_i = 1$ si $x_i > y_i$

$I_i = 1$ si $x_i = y_i$

$m_i = 1$ si $x_i < y_i$

- Cada par de bit se realiza la comparación con un comparador de un bit

$$M = M_{n-1} + I_{n-1} M_{n-2} + I_{n-1} I_{n-2} M_{n-3} + \ldots\ldots + I_{n-1} I_{n-2} \ldots I_1 M_0$$

$$I = I_{n-1} I_{n-2} \ldots\ldots I_1 I_0$$

$$m = m_{n-1} + I_{n-1} m_{n-2} + I_{n-1} I_{n-2} m_{n-3} + \ldots\ldots + I_{n-1} I_{n-2} \ldots I_1 M_0$$

la explicación de estas expresiones es inmediata, así

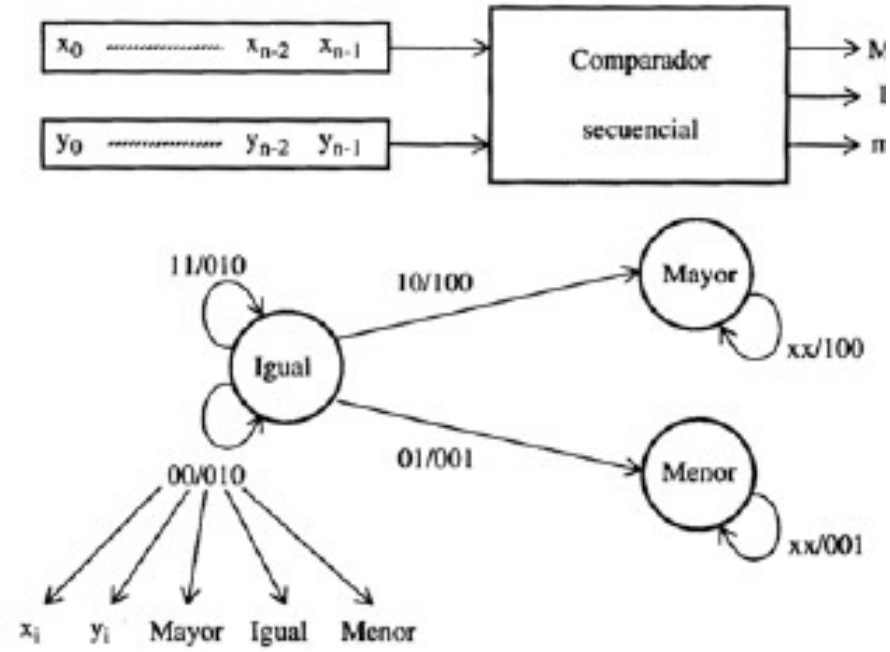
$$x > y \quad \text{si} \quad (x_{n-1} > y_{n-1}) \quad \text{ó} \quad ((x_{n-1} = y_{n-1}) \text{ y } (x_{n-2} > y_{n-2})) \quad \text{ó} \ldots.$$

y análogamente para las otras salidas.

Los circuitos comparadores tienen además 3 entradas de expansión M_{-1}, I_{-1} y m_{-1} con la finalidad de poder realizar las ecuaciones anteriores mediante la asociación de circuitos comparadores.

4.10.2 Utilizando un circuito secuencial

- Un comparador recibe los bits x_i, y_i de forma serie, comenzando por los más significativos.
- El circuito parte de un estado inicial en el que considera que los dos números son iguales y continua en este estado mientras $x_i = y_i$.
- Cuando detecta que $x_i \# y_i$ puede decidir si $x > y$ o $x < y$.
- A partir de ese instante el circuito no cambia de estado.
- El coste del comparador secuencial es independiente de la longitud n de los números, que sin embargo si que afecta al tiempo de ejecución



4.10.3 Utilizando un sumador

- Dado que la UAL dispone de un sumador, la forma más usual de realizar la comparación de dos operandos es restando uno de otro y comprobar el signo del resultado.
- Los bits analizados son: el carry (C) o llevada, el cero (Z) , el de signo (N) y el rebosamiento o overflow (V).
- El procedimiento difiere según sean números con signo o sin signo

Arrastre (C)	$\text{Resultado} \geq 2^n \Rightarrow C = 1$ $\text{Resultado} < 2^n \Rightarrow C = 0$
Rebose (V)	$x_{n-1}\, y_{n-1}\, \bar{r}_{n-1} + \bar{x}_{n-1}\, \bar{y}_{n-1}\, r_{n-1} = 1 \Rightarrow V = 1$ $x_{n-1}\, y_{n-1}\, \bar{r}_{n-1} + \bar{x}_{n-1}\, \bar{y}_{n-1}\, r_{n-1} = 0 \Rightarrow V = 0$
Signo resultado (N)	$\text{Resultado} < 0 \Rightarrow N = 1$ $\text{Resultado} \geq 0 \Rightarrow N = 0$
Resultado cero (Z)	$\text{Resultado} = 0 \Rightarrow Z = 1$ $\text{Resultado} \neq 0 \Rightarrow Z = 0$

Tabla 4.21: Registros de condición de las operaciones aritméticas

a) *Comparación de números positivos sin signo.* Al efectuar la operación (4.22) se tiene:

si $x \geq y \Rightarrow r \geq 2^n \Rightarrow C = 1$, si $x < y \Rightarrow r < 2^n \Rightarrow C = 0$

De acuerdo con esto y teniendo en cuenta los resultados de la Tabla 4-17 se pueden dar las condiciones que determinan la relación existente entre *x* e *y*. En la Tabla 4.22 se muestran las condiciones de comparación en el caso de números positivos sin signo.

Operación: x - y	Condición
$x \geq y$	$C = 1$
$x < y$	$C = 0$
$x > y$	$C = 1$ y $Z = 0 \Rightarrow (\overline{C} + Z = 0)$
$x \leq y$	$C = 0$ ó $Z = 1 \Rightarrow (\overline{C} + Z = 1)$
$x = y$	$Z = 1$
$x \neq y$	$Z = 0$

Tabla 4.22: Comparación de números positivos sin signo

b) *Comparación de números con signo representados en complemento a 2*. En general, al efectuar la operación $r = x - y$ el signo del resultado indicará cual es el mayor de los operandos, es decir:

$$\text{si } r \geq 0 \Rightarrow x \geq y \Rightarrow N = 0 \quad \text{(resultado positivo o cero)}$$
$$\text{si } r < 0 \Rightarrow x < y \Rightarrow N = 1 \quad \text{(resultado negativo)}$$

Sin embargo, el signo del resultado puede no ser el correcto cuando se produce la condición de rebose. Esta condición se detecta por la ALU poniendo el bit de rebose *V* a 1. Por consiguiente cuando V = 1, el signo del resultado de la operación no es el indicado por *N* sino que es el contrario. En consecuencia, al hacer la operación r = x - y, se puede asegurar que:

$$r \geq 0 \Rightarrow x \geq y \text{ si } (N = 0 \text{ y } V = 0) \text{ ó } (N = 1 \text{ y } V = 1) \Rightarrow N \oplus V = 0$$
$$r < 0 \Rightarrow x < y \text{ si } (N = 1 \text{ y } V = 0) \text{ ó } (N = 0 \text{ y } V = 1) \Rightarrow N \oplus V = 1$$

De acuerdo con esto, en la Tabla 4.23 se dan las condiciones que permiten calcular la relación existente entre *x* e *y* cuando representan números con signo en complemento a 2. Compárense los resultados con el caso de números positivos sin signo dados en la Tabla 4.22.

Operación: x - y	Condición
$x \geq y$	$N \oplus V = 0$
$x < y$	$N \oplus V = 1$
$x > y$	$Z = 0 \text{ y } (N \oplus V) = 0 \Rightarrow (Z + (N \oplus V)) = 0$)
$x \leq y$	$Z = 1 \text{ ó } (N \oplus V) = 1 \Rightarrow (Z + (N \oplus V)) = 1$)
$x = y$	$Z = 1$
$x \neq y$	$Z = 0$

Tabla 4.23: Comparación de números representados en complemento a 2

Cuestión

- Empleando únicamente una ROM, se pretende diseñar un comparador de dos números de 4 bits que genere las funciones “mayor que”, “menor que” e “igual que”. Indique cuál de las siguientes ROM podría emplearse.

A) 2^4 palabras, 4 bits/palabra

B) 2^8 palabras, 4 bits/palabra

C) Las dos anteriores

D) Ninguna de las anteriores

- La pregunta está basada en el problema 4-44.
- En un caso general para se necesitará una memoria ROM de un tamaño mínimo: ($2^{\text{núm. entradas}}$) palabras × (núm. salidas) bits/palabra
- En el caso del comparador propuesto el número de entradas es 4 + 4 = 8, y el número de salidas es 3, por lo que se necesitará una memoria ROM de un tamaño mínimo de 2^8 palabras × 3 bits/palabra. La respuesta A propone una capacidad con un número de palabras inferior al necesario y no vale.
- Sin embargo la respuesta B tiene capacidad mayor que la necesaria por lo que se podría emplear para diseñar el comparador. Respuesta: B(2^8 palabras × 4 bits/palabra)