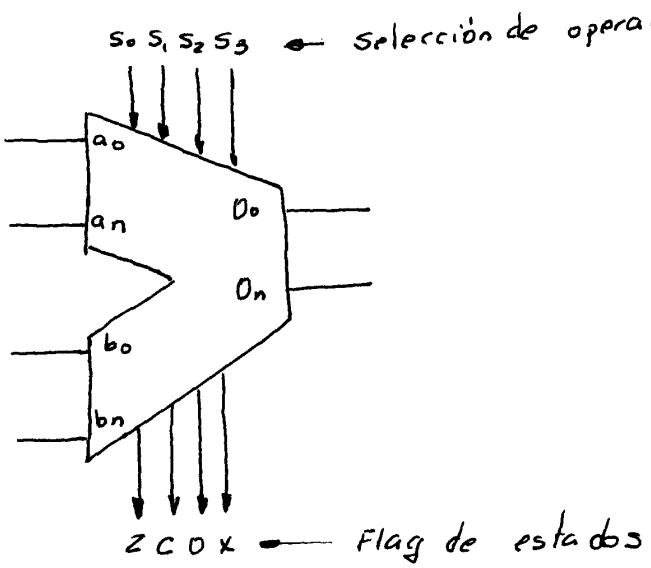
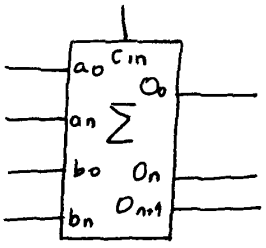
4.- Unidad aritmético-lógica (ALU)

Introducción

ALU ⇒ Unidad aritmético-lógica ⇒ realiza las operaciones aritméticas y lógicas entre los datos.



1.- Sumadores binarios



$a + b = O$

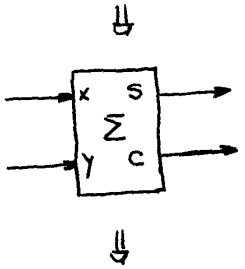
C_{IN} = Carry in (entrada carry)

$C_{OUT} = O_{n+1}$ = Carry out

Estrategia para realizar un sumador → descomponerlo en sumadores más pequeños.

1.1 Semisumador binario

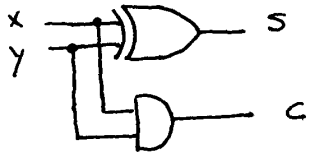
↳ sumador del bit menos significativo



x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

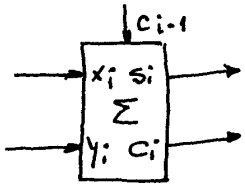
$s = x \oplus y$

$c = x\,y$

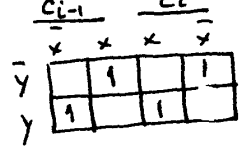


1.2. Sumador completo

↳ suma dos bits más el carry de entrada obtenido de la suma de los bit interiores en peso



c_{i-1}	x	y	s	c
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



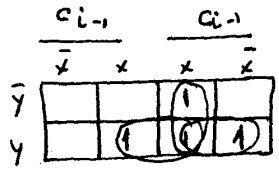
$s = \bar{c}_{i-1}\bar{x}y + \bar{c}_{i-1}x\bar{y} + c_{i-1}\bar{x}\bar{y} + c_{i-1}xy$

$s = \bar{c}_{i-1}(\bar{x}y + x\bar{y}) + c_{i-1}(\bar{y}\bar{y} + xy) =$

$s_1 = \bar{c}_{i-1}(x \oplus y) + c_{i-1}\overline{(x \oplus y)} =$

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

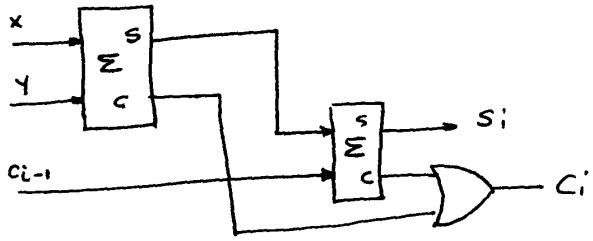
(c)



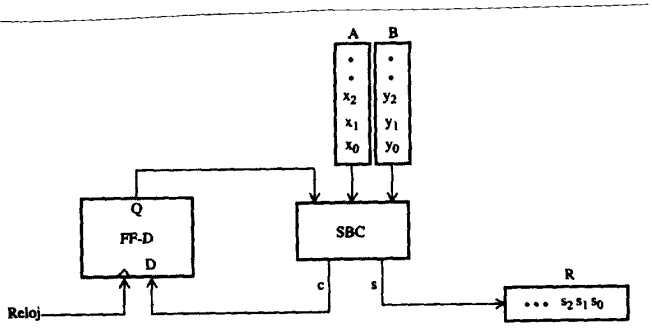
$$c = c_{i-1}\bar{x} + xy + c_{i-1}y$$

También $\Rightarrow$ $c = xy + c_{i-1}x\bar{y} + c_{i-1}\bar{x}y = xy + c_{i-1}(x \oplus y)$

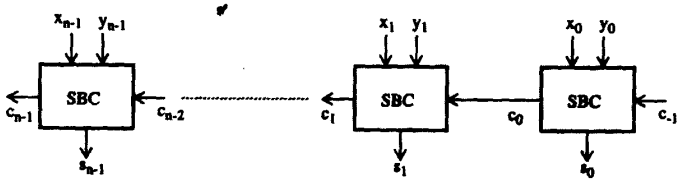
xy → semisumador; $(x \oplus y)$ → semisumador



1.3 Sumador binario serie



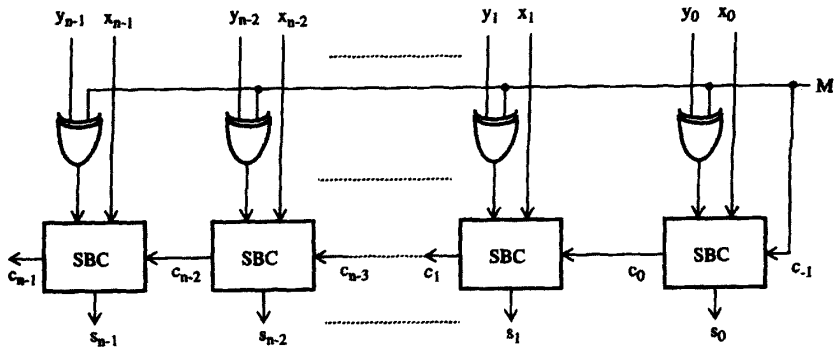
1.4 Sumador binario paralelo con propagación de arrastre



1.5 Sumador restador paralelo con propagación de arrastre

Sumador ⇒ $x + y$

restador ⇒ Sumador de x + complemento a 2 de y

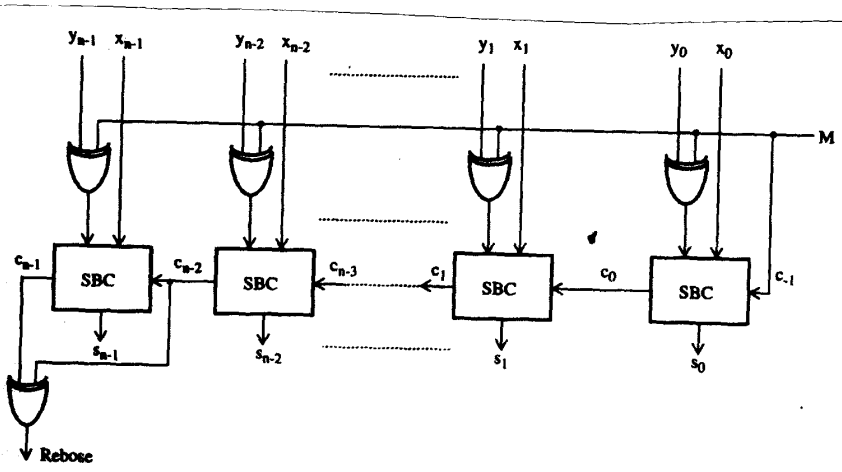


Suma ⇒ $M = 0$

Resta ⇒ $M = 1$

Rebose

- $x + y$ (positivos) de $x_{n-1} + y_{n-1}$ se puede propagar carry y afectar al signo bit n
- $x + y$ (negativos) de $x_{n-1} + y_{n-1}$ se puede propagar carry y afectar al signo bit n poniendolo a $\emptyset$ y falseando resultado.



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

2.- Sumadores de alta velocidad

Estrategias para resolver el problema de la propagación del arrastre

- Aceptar la existencia de los arrastres
- Anticipación del arrastre
- Suma condicional
- Detección de la finalización del arrastre
- Minimización del nº de arrastres
- Arrastre almacenado

¿Cuándo hay arrastre?

- Generación → en la posición $i \longleftrightarrow x_i + y_i > 1$
- Propagación → si llega de $i-1$ se propagará a $i+1 \longleftrightarrow x_i + y_i = 1$

2.- Sumadores con anticipación de arrastre

Reduce el retardo en base a generar la entrada de arrastre de la etapa i-ésima directamente a partir de los bits de entrada a las etapas precedentes $i-1, i-2, \ldots, 0$, en lugar de esperar la propagación.

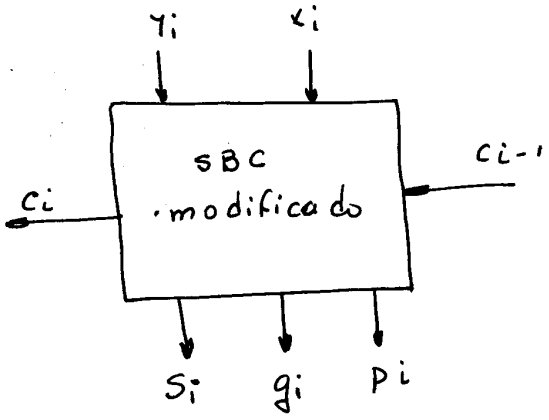
$$s_i = (x_i \oplus y_i) \oplus c_{i-1} \quad (p)$$

$$c_i = (x_i y_i) + (x_i \oplus y_i)\, c_{i-1} \quad (g)$$

} Apdo 1.2 (Pg. 2)

$$s_i = p_i \oplus c_{i-1}$$

$$c_i = g_i + p_i \cdot c_{i-1}$$



$s_i = p_i \oplus c_{i-1}$

$c_i = g_i \oplus p_i \cdot c_{i-1}$

$\Downarrow$

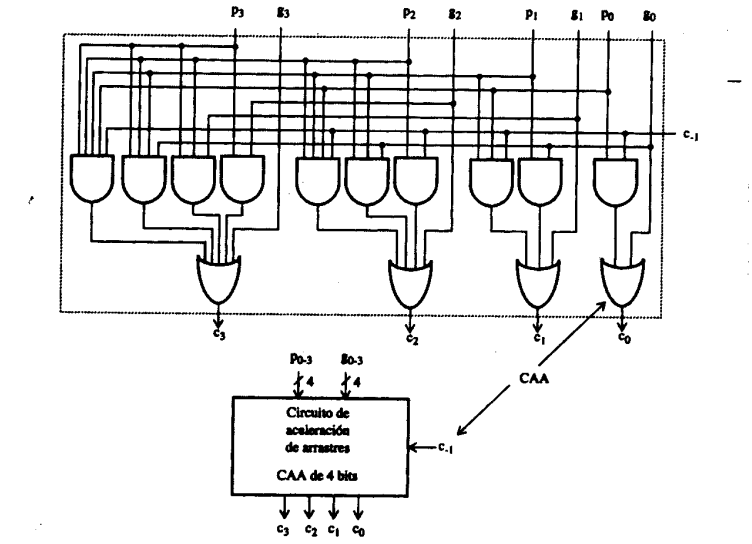
$c_0 = g_0 \oplus p_0 \cdot c_{-1}$

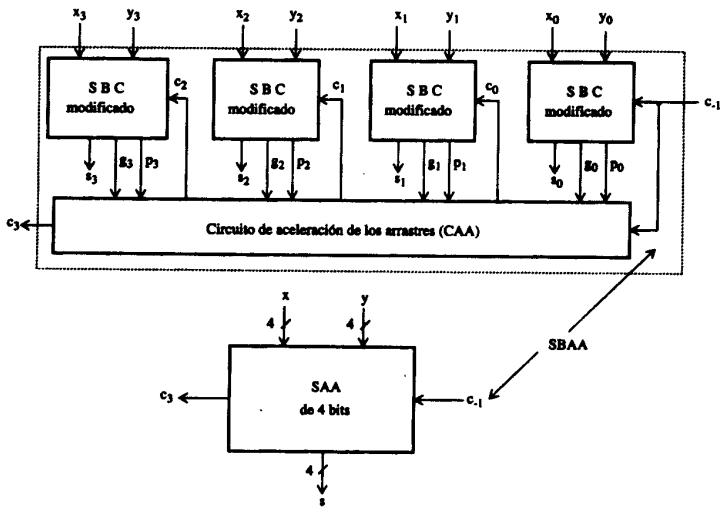
$c_1 = g_1 + p_1 c_0 = g_1 + p_1 (g_0 \oplus p_0 c_{-1}) =$

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

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

$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} p_{i-2} \cdots p_0 c_{-1}$



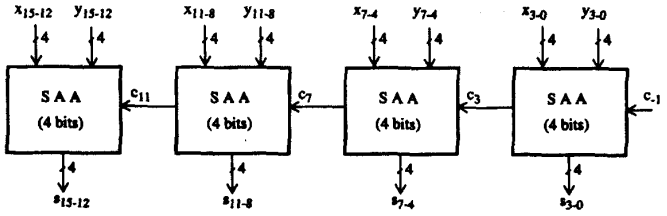


Sumador muchos bits ⇒ circuitería compleja

⇓

Agrupación de sumadores con aceleración de arrastre con propagación de arrastre entre ellos

⇓



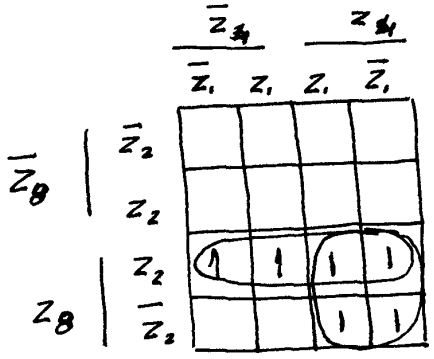
3.- Sumadores en código BCD

- Suma 2 dígitos menor 9 ⇒ correcto
- " " " > 9 ⇒ resultado incorrecto y hay que corregirlo.

Corrección
- Resultado suma > 15 ⇒ arrastre $k=1$
- " " entre 10 y 15 ($c_1=1$)

⇓

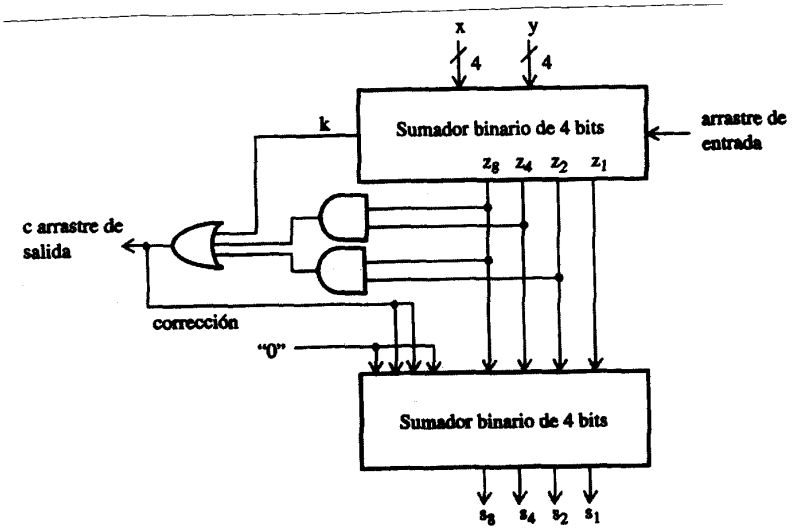
$z_8 z_4 z_2 z_1 = 1010$ y $z_8 z_4 z_2 z_1 = 1111$



$c_1 = z_8 z_2 + z_8 z_4$

⇓

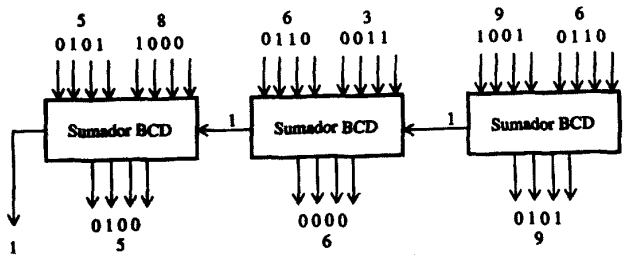
Corrección ⇒ sumar 6 al resultado de la suma

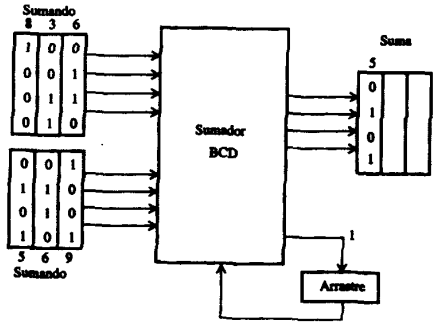


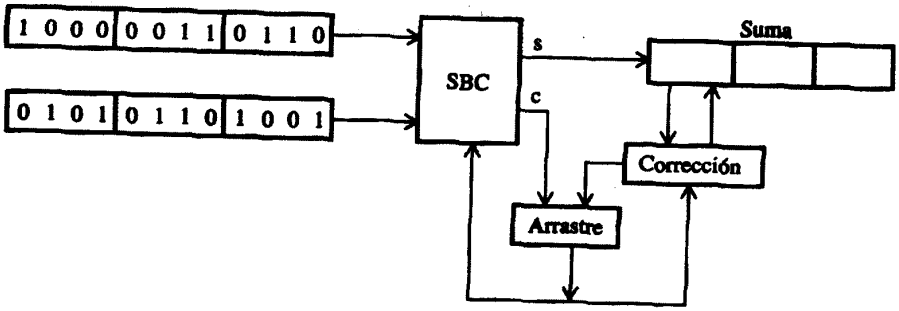
3.1 Organización de sumadores en código BCD

La suma de 2 números de n dígitos decimales codificados en BCD se puede hacer

- Sumador paralelo
- " dígito-serie, bit-paralelo
- " " " , bit-serie







4. Multiplicadores binarios

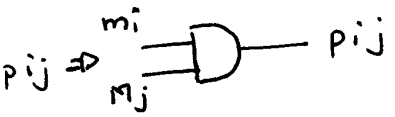
1.- Multiplicación de "lápiz y papel" de números sin signo

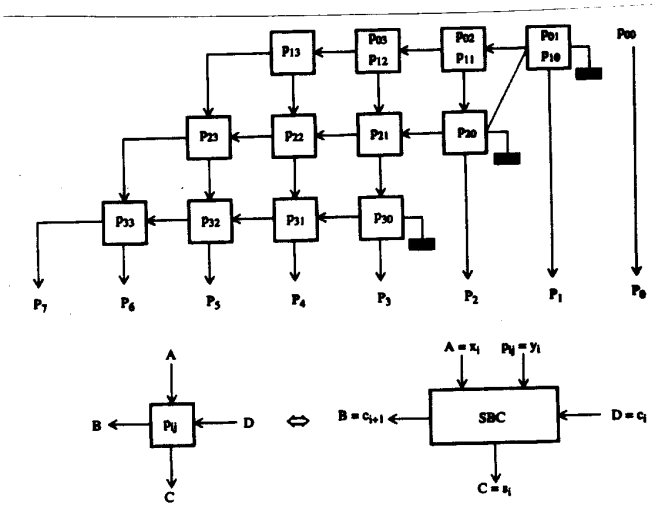
```
M =   1 0 0 1      multiplicando
m =   1 0 1 1      multiplicador
      ---------
      1 0 0 1
    1 0 0 1        Productos parciales
  0 0 0 0
1 0 0 1
---------------
1 1 0 0 0 1 1      Producto
```

Consideraciones

- La multiplicación genera 4 productos parciales (pi)
- Los prod. parciales
  - multiplicador = 0 ⇒ ∅
  - " = 1 ⇒ multiplicando
- Cada prod. parcial se desplaza 1 lugar izda
- Multiplicar 2 nºs binarios de n bits ⇒ 2n bit result

```
        M3 M2 M1 M0
        m3 m2 m1 m0
      ---------------
        P03 P02 P01 P00
     P13 P12 P11 P10
  P23 P22 P21 P20
P33 P32 P31 P30
-------------------------------
P6  P5  P4  P3  P2  P1  P0
```

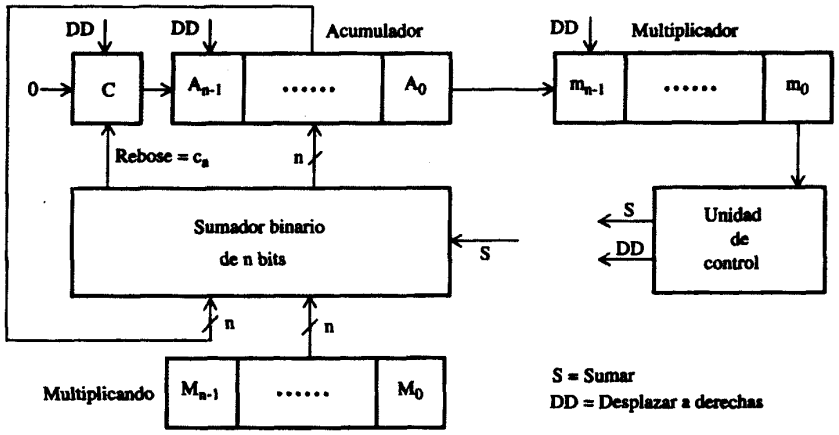
⇒ pij ⇒ 



La velocidad depende del retardo de propagación de los arrastres.

2.- Mejoras en el algoritmo de "lápiz y papel"

Mejoras:
- Sumar los productos parciales cuando se producen, sin esperar al final.
- Ahorrar tiempo, ya que un $\emptyset \Rightarrow$ desplazam. izda



Operación:

a) Unidad ctrl lee los bit del multiplicador de uno en uno

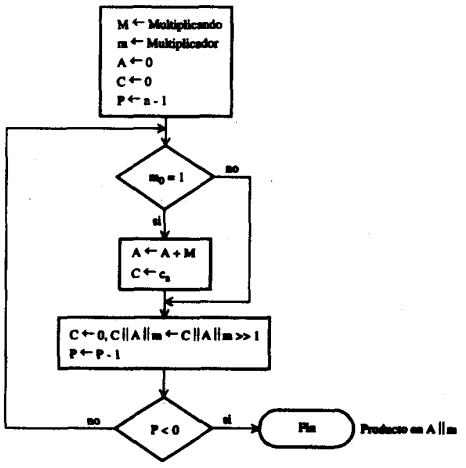
b) Si $m_0 = 1 \Rightarrow$ Multiplicando + A → A

c) Todos los bits de los reg. C, A y m desplaz. 1bit dcha

$\emptyset \rightarrow C$

$C \rightarrow A_{n-1}$; $A_{n-1} \rightarrow A_{n-2} \cdots$;

$A\emptyset \rightarrow m_{n-1}$; $\ldots m_1 \rightarrow m_0$; m_0 = pierde



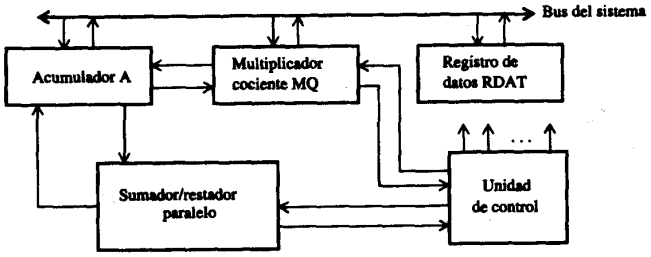
d) $m_0 = 0 \Rightarrow$ no suma y solo desplaza

e) Operaciones se repiten para todo bit de multiplicador.

f) Resultado en A y m

7.- Estructura de la unidad aritmético-lógica (ALU)

Complejidad ALU
- Tipos de operaciones que puede ejecutar
- Forma de ejecutar las operaciones.

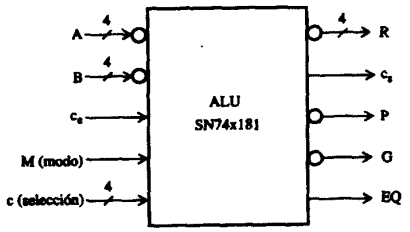


1.- ALU's integradas

SN74181 ⇒ ALU 4 bits

⇓

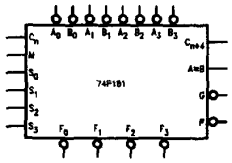
A, B = datos entrada (4bit)
R = resultado (4bit)
Ce = entrada de arrastre
Cs = salida " "
P = Propag. arrastre
G = Gener. "
EQ = 1 ⟷ A = B



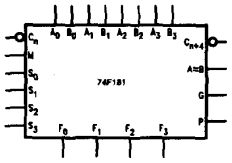
Selección $c_3c_2c_1c_0$	M = 1 Función lógica	M = 0 Función aritmética v(R) = s mod 16
0000	$R = \overline{A}$	$s = v(A) - 1 + c_e$
0001	$R = \overline{A} \vee \overline{B}$	$s = v(A \wedge B) - 1 + c_e$
0010	$R = \overline{A} \vee B$	$s = v(A \wedge \overline{B}) - 1 + c_e$
0011	R = 1111	$s = 1111 + c_e$
0100	$R = \overline{A} \wedge \overline{B}$	$s = v(A) + v(A \vee \overline{B}) + c_e$
0101	$R = \overline{B}$	$s = v(A \wedge B) + v(A \vee \overline{B}) + c_e$
0110	$R = A \oplus \overline{B}$	$s = v(A) - v(B) - 1 + c_e$
0111	$R = A \vee \overline{B}$	$s = v(A \vee \overline{B}) + c_e$
1000	$R = \overline{A} \wedge B$	$s = v(A) + v(A \vee B) + c_e$
1001	$R = A \oplus B$	$s = v(A) + v(B) + c_e$
1010	R = B	$s = v(A \wedge \overline{B}) + v(A \vee B) + c_e$
1011	$R = A \vee B$	$s = v(A \vee B) + c_e$
1100	R = 0000	$s = v(A) + v(A) + c_e$
1101	$R = A \wedge \overline{B}$	$s = v(A \wedge B) + v(A) + c_e$
1110	$R = A \wedge B$	$s = v(A \wedge \overline{B}) + v(A) + c_e$
1111	R = A	$s = v(A) + c_e$

Tabla 4.15: Funciones de la ALU SN74181

Operation Table



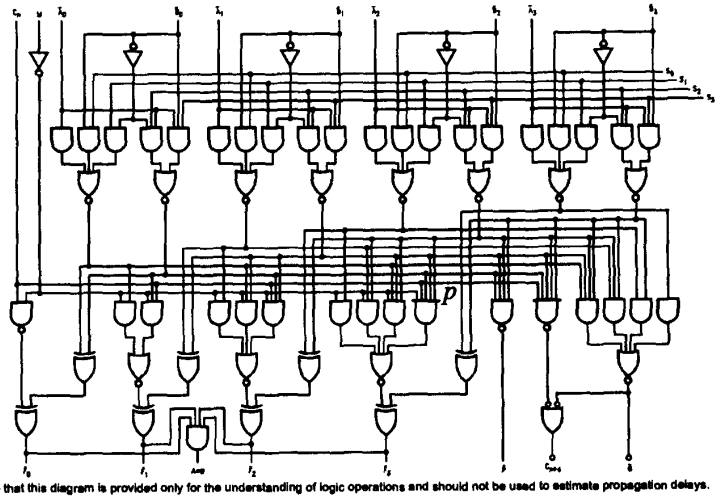
a. All Input Data Inverted



b. All Input Data True

S_0	S_1	S_2	S_3	Logic (M=H)	Arithmetic ($M=L$, C_0=Inactive)	Arithmetic ($M=L$, C_0=Active)
L	L	L	L	$\overline{A}$	A minus 1	A
H	L	L	L	$\overline{A} \cdot \overline{B}$	$A \cdot B$ minus 1	$A \cdot B$
L	H	L	L	$\overline{A} + B$	$A \cdot \overline{B}$ minus 1	$A \cdot \overline{B}$
H	H	L	L	Logic "1"	minus 1 (2s comp.)	Zero
L	L	H	L	$\overline{A + B}$	A plus $(A + \overline{B})$	A plus $(A + \overline{B})$ plus 1
H	L	H	L	$\overline{B}$	$A \cdot B$ plus $(A + \overline{B})$	$A \cdot B$ plus $(A + \overline{B})$ plus 1
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1	A minus B
H	H	H	L	$A + \overline{B}$	$A + \overline{B}$	$A + \overline{B}$ plus 1
L	L	L	H	$\overline{A} \cdot B$	A plus $(A + B)$	A plus $(A + B$ plus 1
H	L	L	H	$A \oplus B$	A plus B	A plus B plus 1
L	H	L	H	B	$A \cdot \overline{B}$ plus $(A + B)$	$A \cdot \overline{B}$ plus $(A + B)$ plus 1
H	H	L	H	$A + B$	$A + B$	$A + B$ plus 1
L	L	H	H	Logic "0"	A plus A $(2 \times A)$	A plus A $(2 \times A)$ plus 1
H	L	H	H	$A \cdot \overline{B}$	A plus $A \cdot B$	A plus $A \cdot B$ plus 1
L	H	H	H	$A \cdot B$	A plus $A \cdot \overline{B}$	A plus $A \cdot \overline{B}$ plus 1
H	H	H	H	A	A	A plus 1
L	L	L	L	$\overline{A}$	A	A plus 1
H	L	L	L	$\overline{A + B}$	$A + B$	$A + B$ plus 1
L	H	L	L	$\overline{A} \cdot B$	$A + \overline{B}$	$A + \overline{B}$ plus 1
H	H	L	L	Logic "0"	minus 1 (2s comp.)	Zero
L	L	H	L	$\overline{A} \cdot \overline{B}$	A plus $(A \cdot \overline{B})$	A plus $A \cdot \overline{B}$ plus 1
H	L	H	L	$\overline{B}$	$A \cdot \overline{B}$ plus $(A + B)$	$A \cdot B$ plus $(A + B)$ plus 1
L	H	H	L	$A \oplus B$	A minus B minus 1	A minus B
H	H	H	L	$A \cdot \overline{B}$	$A \cdot \overline{B}$ minus 1	$A \cdot \overline{B}$
L	L	L	H	$\overline{A} + B$	A plus $A \cdot B$	A plus $A \cdot B$ plus 1
H	L	L	H	$\overline{A \oplus B}$	A plus B	A plus B plus 1
L	H	L	H	B	$A \cdot B$ plus $(A + \overline{B})$	$A \cdot B$ plus $(A + \overline{B})$ plus 1
H	H	L	H	$A \cdot B$	$A \cdot B$ minus 1	$A \cdot B$
L	L	H	H	Logic "1"	A plus A $(2 \times A)$	A plus A $(2 \times A)$ plus 1
H	L	H	H	$A + \overline{B}$	A plus $(A + B)$	A plus $(A+\overline{B})$ plus 1
L	H	H	H	$A + B$	A plus $(A + \overline{B})$	A plus $(A+\overline{B})$ plus 1
H	H	H	H	A	A minus 1	A

Logic Diagram



Please note that this diagram is provided only for the understanding of logic operations and should not be used to estimate propagation delays.

74F181

ALU. 12

9. Operaciones de desplazamiento

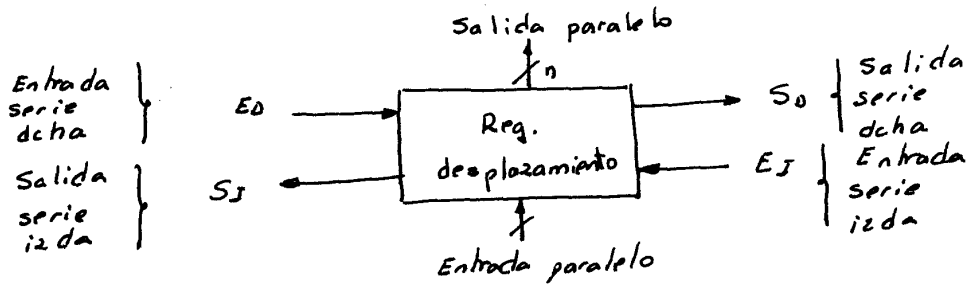
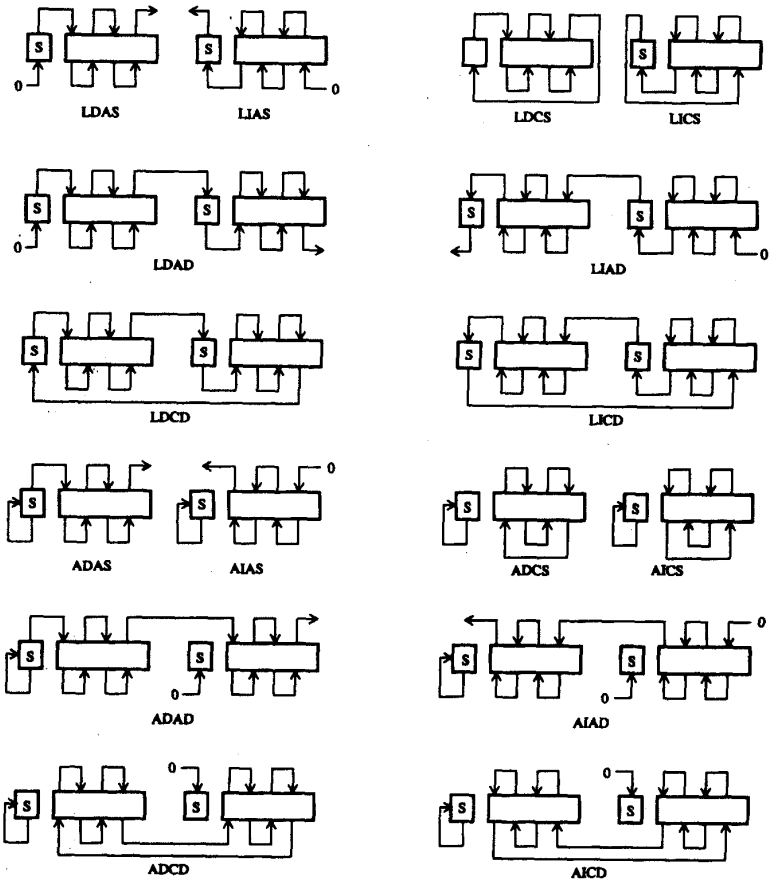


Diagrama de bloques

1.- Clasificación de los desplazamientos

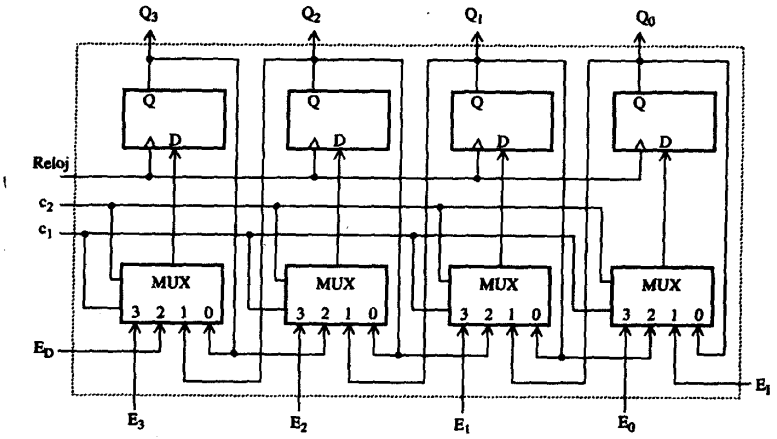
- Tratamiento bit signo
  - Aritméticos (A) → No afecta al signo
  - Lógicos (L) → Interviene signo
- Sentido desplazamiento
  - Dcha (D)
  - Izda (I)
- Tratam. bits que rebosan
  - Abierto (A) → se pierde
  - Cerrado (C) → intervienen
- Longitud registros
  - Simples (S) → único registro
  - Dobles (D) → pareja registros



2.- Diseño de un reg. desplazamiento 4 bits

Tabla verdad

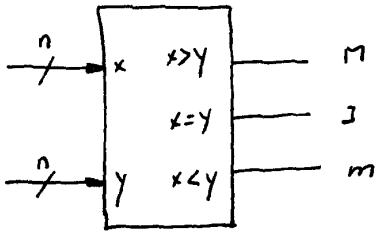
Operación	C_2	C_1	
NOP	0	0	
LIAS	0	1	→ Logi. Izd. Abier. Simple
LDAS	1	0	→ Logi. Dcha. Abier. Simple
CARGA	1	1	→ Paralelo



3.- Tipos registros

- Entrada paralelo / salida paralelo
- " serie / " serie
- " paralelo / " serie
- " serie / " paralelo

10.- Operaciones de comparación.



Comparador

Formas diseño comparador
- Circuito combinacional
- " secuencial
- Sumador

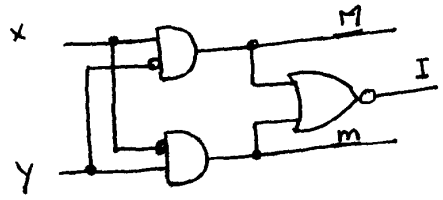
1. Circuito combinacional

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

$M = x\bar{y}$

$m = \bar{x}y$

$I = \bar{x}\bar{y} + xy = \overline{\bar{x}y + x\bar{y}}$



Para n bits

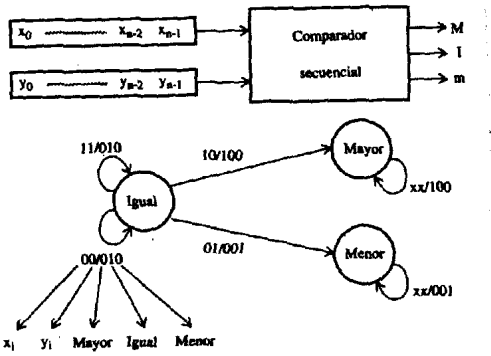
$$M = M_{n-1} + \bar{I}_{n-1} M_{n-2} + I_{n-1} I_{n-2} M_{n-3} + \dots + I_{n-1} I_{n-2} \dots I_1 M_0$$

$$I = I_{n-1} I_{n-2} I_{n-3} \dots I_0$$

$$m = m_{n-1} + \bar{I}_{n-1} m_{n-2} + \bar{I}_{n-1} I_{n-2} m_{n-3} + \dots + I_{n-1} I_{n-2} \dots I_1 m_0$$

2.- Con circuito secuencial

El comparador de 1 bit recibe los datos en serie y compara bit a bit desde el más significativo. Se toma la decisión de ">" o "<" en cuanto hay una diferencia. Si no la hay ⇒ x = y



3.- Utilizando un sumador

Restando los números y analizando los flags de arrastre (carry "C") y cero (Z). ⇒ $x - y \Rightarrow x + C2(y) \Rightarrow x + 2^n - y$

Posibilidades

- 2 nº positivos sin signo
  - $x = y \Rightarrow Z = 1$
  - $x > y \Rightarrow Z = 0$ y $C = 1$
  - $x < y \Rightarrow Z = 0$ y $C = 0$
- 2 nº con signo en Comple. a 2
  - $x = y \Rightarrow Z = 1$
  - $x > y \Rightarrow Z = 0$ y $N \oplus V$
  - $x < y \Rightarrow N \oplus V = 1$