FUNDAMENTOS DE SISTEMAS DIGITALES

Tema 2:
Lógica combinacional (I):
Funciones aritmético-lógicas

Programa

1. **Representación conjunta de números positivos y negativos.**
2. **Sumadores y restadores.**
3. **Sumadores en complemento a 1.**
4. **Comparadores.**
5. **Unidades aritmético-lógicas.**

1. Representación conjunta de números positivos y negativos

- Si tenemos **n** bits para representar un dato numérico positivo, hay $\mathbf{2^n}$ posibles combinaciones diferentes que nos darían la posibilidad de representar cantidades desde el 0 ($000\ldots0_2$) hasta el 2^n-1 ($111\ldots1_2$).
  - En **binario puro**, el valor de una cantidad se calcula mediante la fórmula siguiente:

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

- Para representar números positivos y negativos de forma conjunta, se usa **1** bit para el signo, y los **n-1** restantes para la magnitud.

- Sistemas de representación conjunta de números positivos y negativos:
  - Signo-magnitud.
  - Complemento a 1.
  - Complemento a 2.

Signo-magnitud

- El primer bit representa el signo (0: positivo; 1: negativo).
- Los restantes bits representan la magnitud, igual que en binario puro.
  - Fórmula:

$$A = (-1)^{a_{n-1}} \cdot \sum_{i=0}^{n-2} a_i \cdot 2^i$$

- El número 0 tiene una doble representación: 00…0 y 10…0.
- Sumar dos números de igual signo: se suman las magnitudes y se conserva el signo.
- Sumar dos números de distinto signo: es complicado.
  1. Se resta el de mayor magnitud menos el de menor magnitud.
  2. Se coloca el signo del de mayor magnitud.
- Un circuito para sumar números en signo-magnitud necesitaría incorporar un sumador, un restador, un comparador para signos y un comparador para magnitudes.

Complemento a 1

- Los números positivos se representan como en signo-magnitud (empiezan por 0).
- La representación de los números negativos se obtiene tomando el número en positivo y complementando (negando) todos sus bits (empiezan por 1).
- Fórmula: $$A = \sum_{i=0}^{n-2} (a_i - a_{n-1}) \cdot 2^i$$
- El número 0 tiene una doble representación: 00…0 y 11…1.
- Sumar dos números: no importa el signo, se suman directamente.
- Restar dos números: sumar el minuendo más el complementario del sustraendo.
  - A-B = A+(-B)
- Un circuito para sumar y restar números en complemento a 1 estaría formado por un sumador y un complementador.
  - Problema: doble representación del 0 $\Rightarrow$ a veces el resultado sale mal y hay que corregirlo sumando 1.

Complemento a 2

- Los números positivos se representan como en signo-magnitud (empiezan por 0).
- La representación de los números negativos se obtiene tomando el número en positivo, complementando (negando) todos sus bits y sumando 1.
- Fórmula: $A = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i$
- El número 0 tiene representación única: 00…0.
- Sumar dos números: no importa el signo, se suman directamente.
- Restar dos números: sumar el minuendo más el complementario del sustraendo.
  - A-B = A+(-B)
- Un circuito para sumar y restar números en complemento a 2 estaría formado por un sumador y un complementador.
  - El resultado sale siempre correcto si está dentro de rango.

Representación conjunta de números positivos y negativos

- Valores de combinaciones binarias:

Configuraciones Binarias (4 bits)	*Nº equivalente en decimal* según las distintas representaciones en binario			
	Binario Puro	*S-M*	*C-1*	*C-2*
0 000	0	+0	+0	+0
0 001	1	+1	+1	+1
0 010	2	+2	+2	+2
0 011	3	+3	+3	+3
0 100	4	+4	+4	+4
0 101	5	+5	+5	+5
0 110	6	+6	+6	+6
0 111	7	+7	+7	+7
1 000	8	-0	-7	-8
1 001	9	-1	-6	-7
1 010	10	-2	-5	-6
1 011	11	-3	-4	-5
1 100	12	-4	-3	-4
1 101	13	-5	-2	-3
1 110	14	-6	-1	-2
1 111	15	-7	-0	-1

Representación conjunta de números positivos y negativos

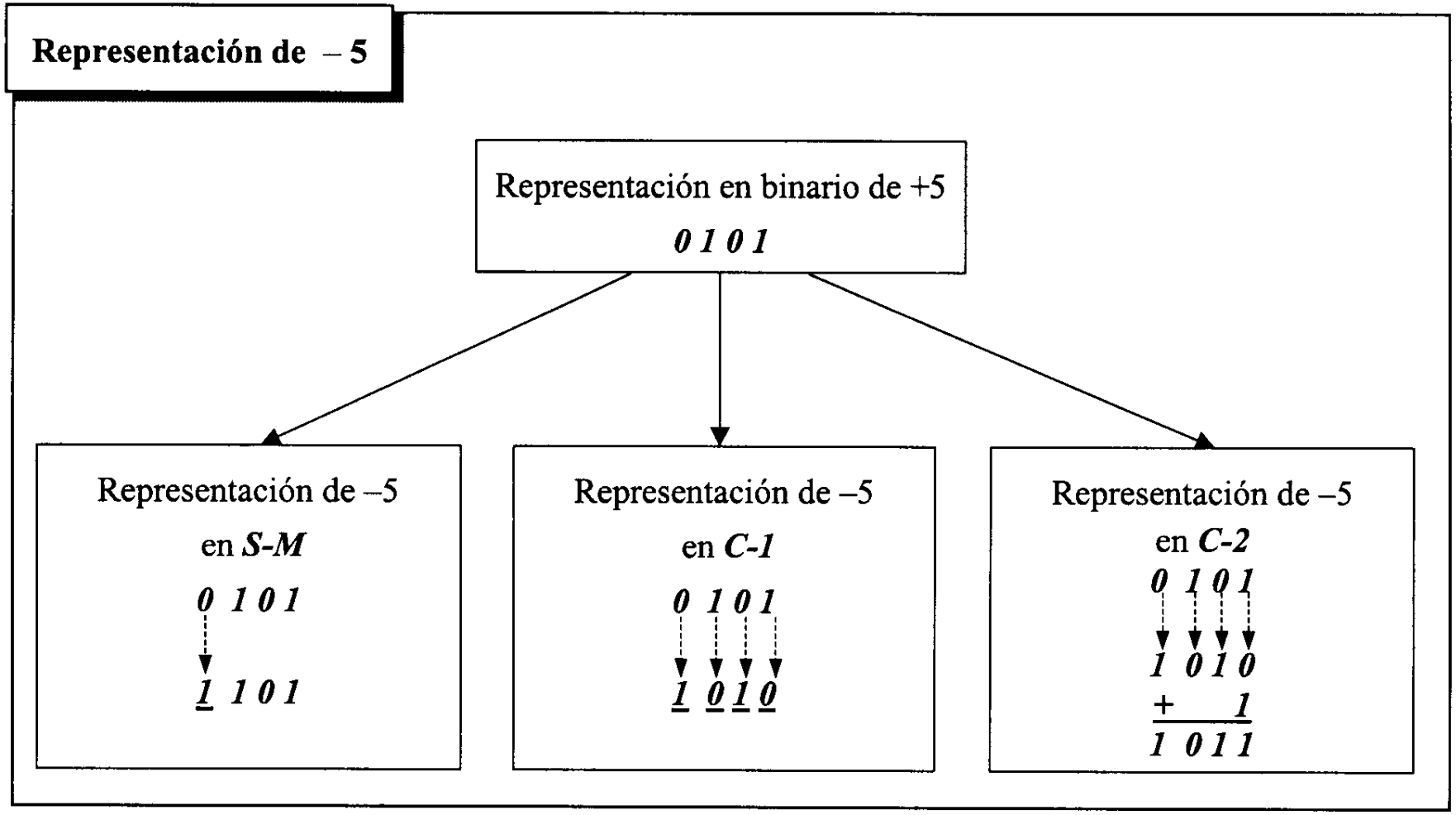
- Combinaciones binarias según valores:

Decimal	Signo y Magnitud (S-M)	Complemento a 1 (C-1)	Complemento a la base 2 (C-2)
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000 ó 1000	0000 ó 1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	(*)	(*)	1000

(*) Hace falta un bit más para representarlo

Representación conjunta de números positivos y negativos

- Ejemplo de conversión de base 10 a binario en diferentes sistemas:



Representación conjunta de números positivos y negativos

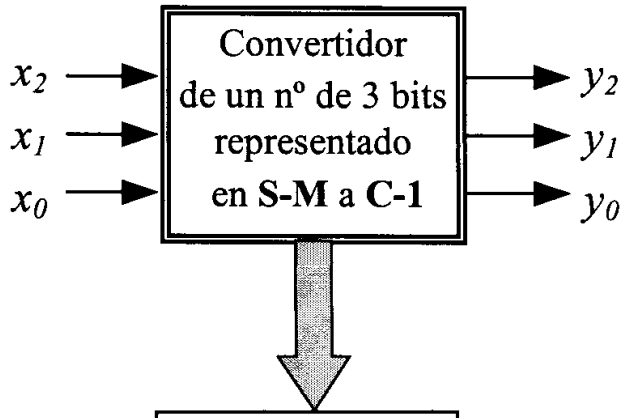
- Circuito de conversión de S-M a C-1:

$$y_2 = x_2$$

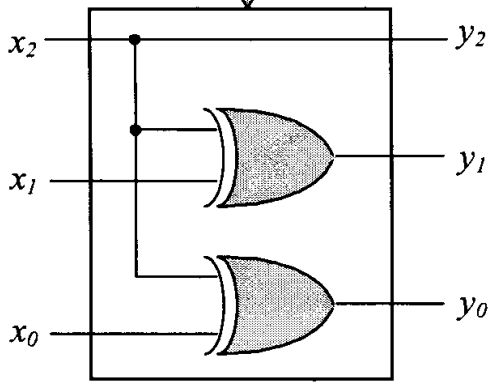
$$y_1 = \bar{x}_2 x_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + x_2 \bar{x}_1 \bar{x}_0 + x_2 \bar{x}_1 x_0 = x_2 \oplus x_1$$

$$y_0 = \bar{x}_2 x_1 x_0 + \bar{x}_2 \bar{x}_1 x_0 + x_2 \bar{x}_1 \bar{x}_0 + x_2 x_1 \bar{x}_0 = x_2 \oplus x_0$$

Signo y Magnitud (S-M)
$x_2\ x_1\ x_0$
0 1 1
0 1 0
0 0 1
0 0 0
1 0 0
1 0 1
1 1 0
1 1 1



Complemento a 1 (C-1)
$y_2\ y_1\ y_0$
0 1 1
0 1 0
0 0 1
0 0 0
1 1 1
1 1 0
1 0 1
1 0 0



Programa

1. Representación conjunta de números positivos y negativos.
2. Sumadores y restadores.
3. Sumadores en complemento a 1.
4. Comparadores.
5. Unidades aritmético-lógicas.

Sumadores y restadores: semisumador

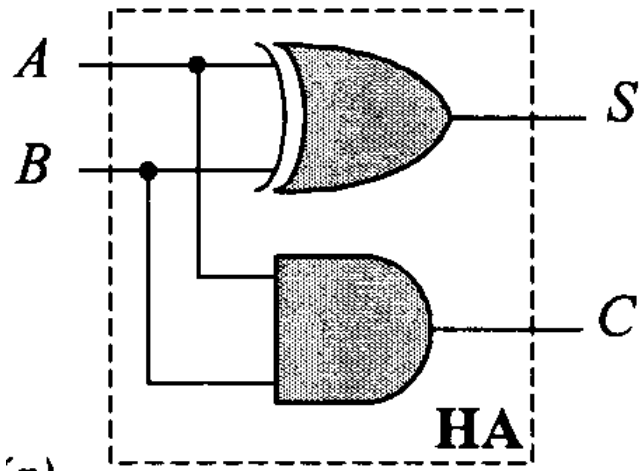
- El semisumador (*half adder*) es un circuito que suma dos bits de entrada A y B y devuelve un bit de resultado S y un bit de arrastre (acarreo o *carry*) C.

- Síntesis del semisumador: inmediata a partir de su tabla de verdad.

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

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

$$C = AB$$



Sumadores y restadores: sumador completo

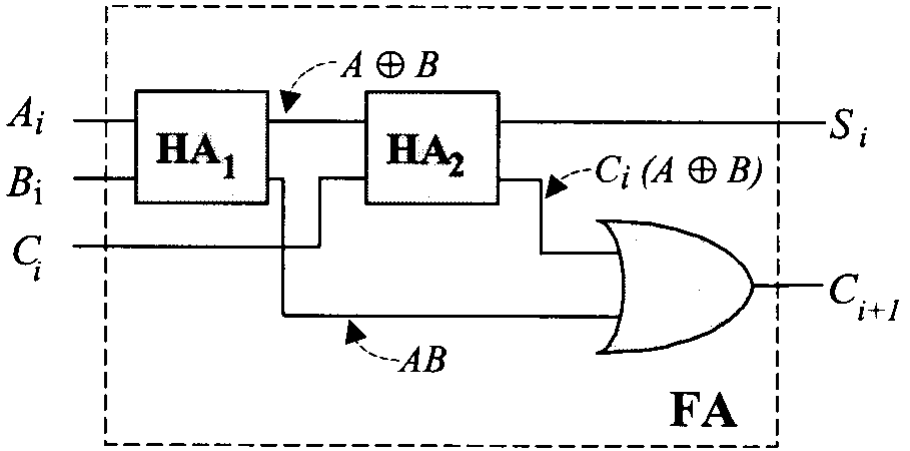
- El sumador completo (*full adder*) es un circuito que suma tres bits de entrada A_i y B_i y C_{i-1} y devuelve un bit de resultado S_i y un bit de arrastre (acarreo) C_i.

- Síntesis del sumador completo a partir de semisumadores:

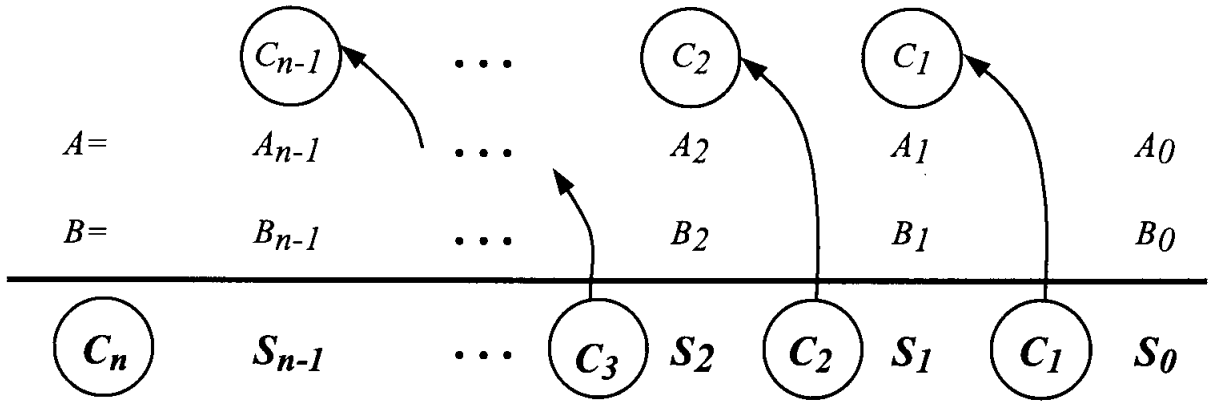
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	*0*	*0*
0	0	1	*1*	*0*
0	1	0	*1*	*0*
0	1	1	*0*	*1*
1	0	0	*1*	*0*
1	0	1	*0*	*1*
1	1	0	*0*	*1*
1	1	1	*1*	*1*

$$S_i = C_i \oplus A_i \oplus B_i$$

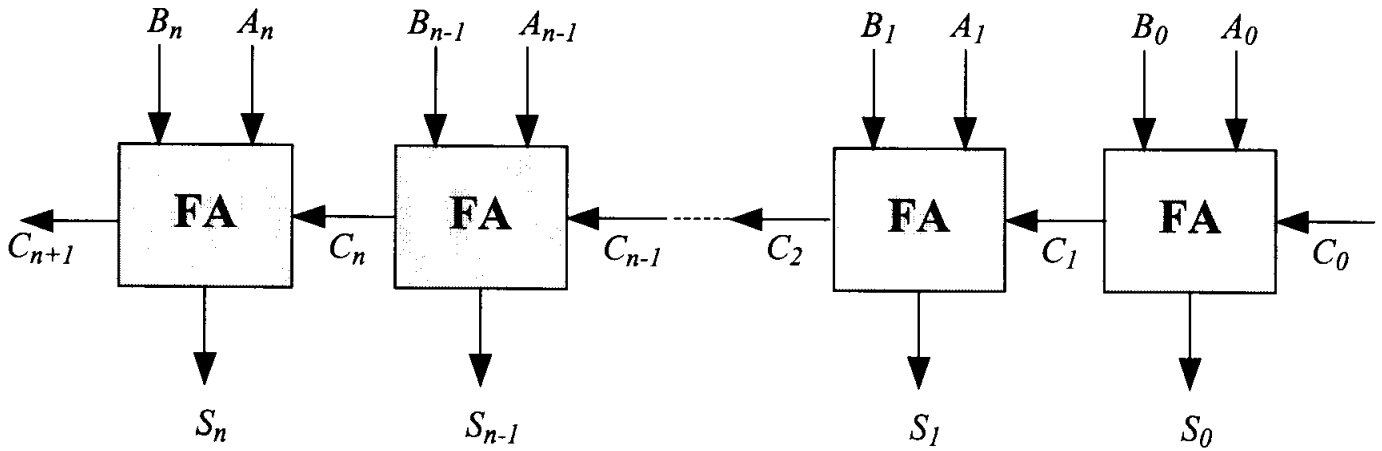
$$C_{i+1} = A_i B_i + C_i(A_i \oplus B_i)$$



Sumador paralelo para palabras de n bits



$$S=C_n\,S_{n-1}\,S_{n-2}\cdots S_2\,S_1\,S_0$$



- Pega: propagación de arrastre muy lenta.

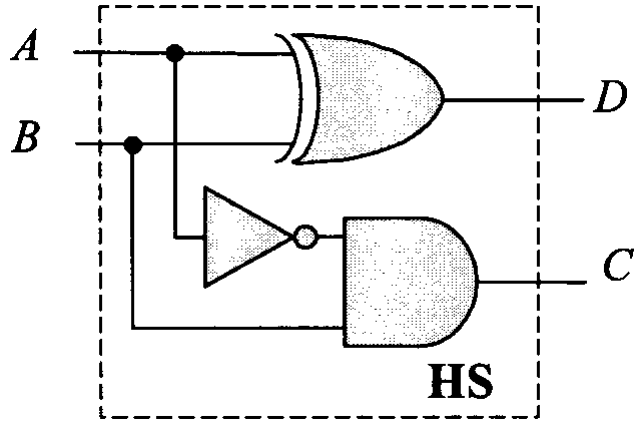
Sumadores y restadores: semirrestador

- El semirrestador (*half substracter*) es un circuito que resta dos bits de entrada A y B y devuelve un bit de resultado D y un bit de arrastre (acarreo) C.

- Síntesis del semirrestador: inmediata a partir de su tabla de verdad.

A	B	D	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = \overline{A}B + A\overline{B} = A \oplus B$$
$$C = \overline{A}B$$



Sumadores y restadores: restador completo

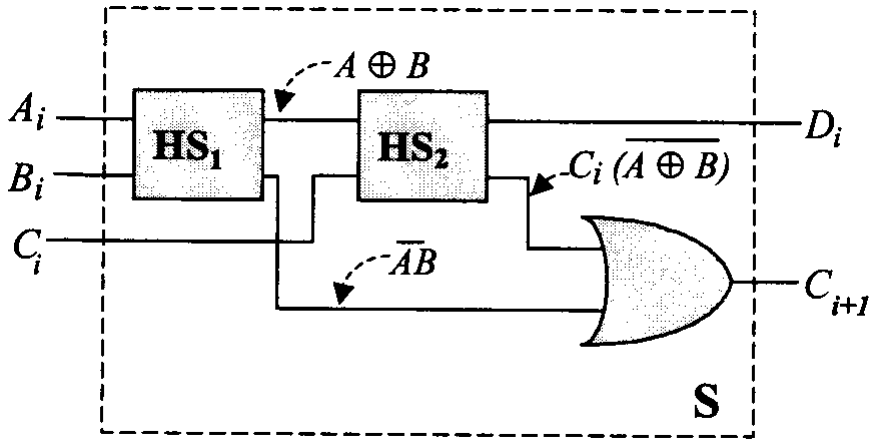
- El restador completo (*full substracter*) es un circuito que resta tres bits de entrada A_i y B_i y C_{i-1} y devuelve un bit de resultado D_i y un bit de arrastre (acarreo) C_i.

- Síntesis del restador completo a partir de semirrestadores:

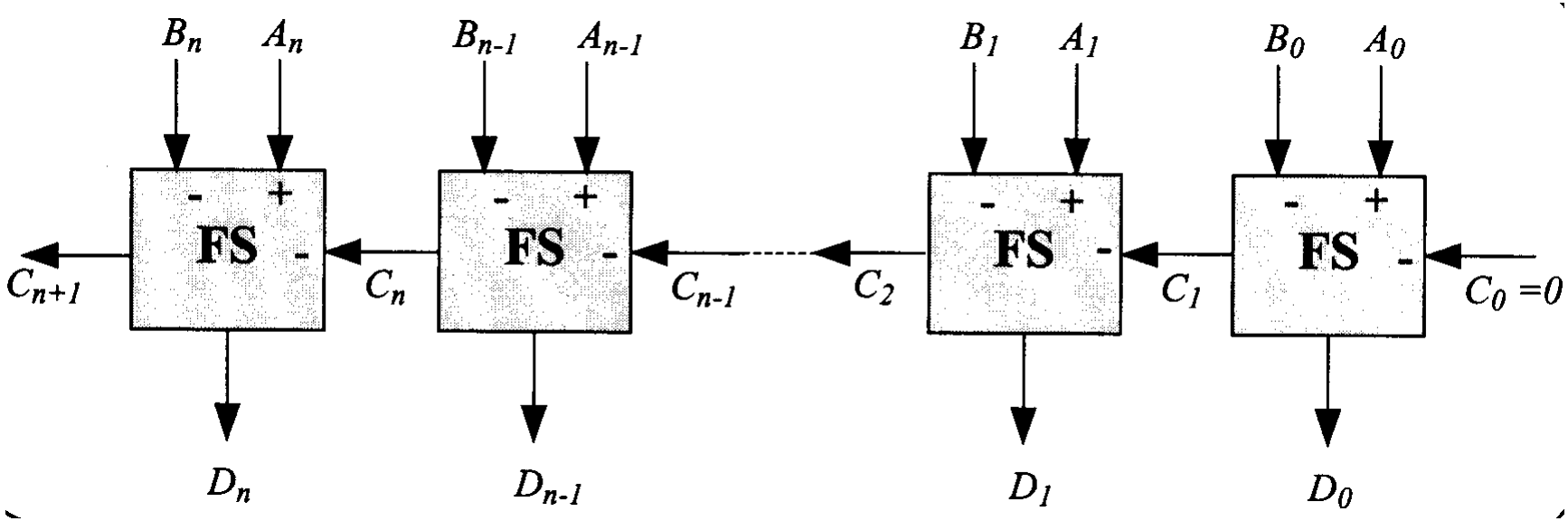
A_i	B_i	C_i	D_i	C_{i+1}
0	0	0	*0*	*0*
0	0	1	*1*	*1*
0	1	0	*1*	*1*
0	1	1	*0*	*1*
1	0	0	*1*	*0*
1	0	1	*0*	*0*
1	1	0	*0*	*0*
1	1	1	*1*	*1*

$$D_i = C_i \oplus A_i \oplus B_i$$

$$C_{i+1} = \overline{A}_i \cdot B_i + C_i\left(\overline{A_i \oplus B_i}\right)$$



Restador paralelo para palabras de n bits

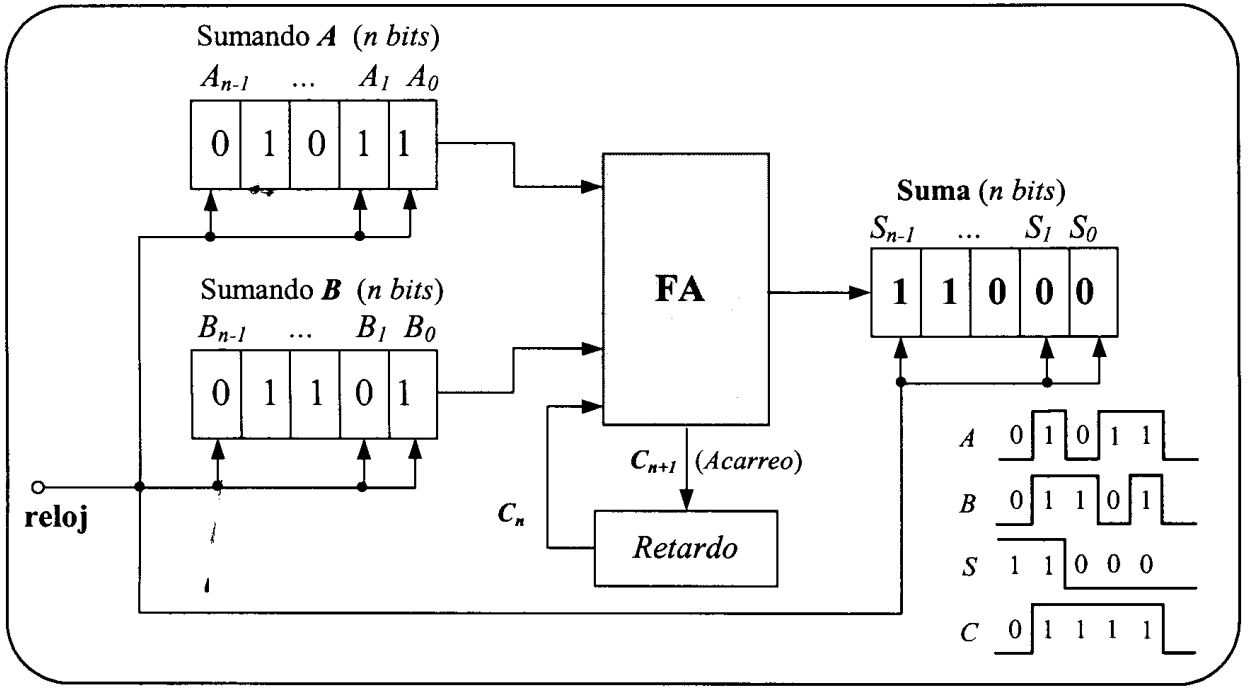


- Pega: propagación de arrastre muy lenta.

Sumadores y restadores: sumador serie

- Es útil cuando los bits de datos de las entradas no llegan en paralelo, sino como trenes de impulsos.

- Circuitería necesaria:
  - Un sumador completo.
  - Un reloj que controla la entrada síncrona de los datos de entrada mediante los trenes de impulsos.
  - Un registro que almacena el acarreo y lo presenta a la entrada del sumador completo cuando se suma la siguiente pareja de bits de entrada.



Sumador paralelo con acarreo adelantado

- La pega del sumador paralelo que hemos visto antes es que el acarreo se propaga con lentitud de una pareja de bits a otra.

- Adelantamiento de acarreo (*lookahead carry*): los acarreos se pueden calcular y no tienen por qué propagarse.

- Es posible definir dos funciones:
  - $\mathbf{G_i}$: vale 1 si la pareja de bits $\mathbf{A_i}$ y $\mathbf{B_i}$ generan acarreo $\Rightarrow$ $\mathbf{G_i = A_i \cdot B_i}$
  - $\mathbf{P_i}$: vale 1 si la pareja de bits $\mathbf{A_i}$ y $\mathbf{B_i}$ pueden propagar un acarreo procedente de una pareja de bits anterior $\Rightarrow$ $\mathbf{P_i = A_i \oplus B_i}$

- Entonces: $C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i = G_i + P_i \cdot C_i$

  $S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$

Sumador paralelo con acarreo adelantado

- El circuito que calcule los acarreos materializará las siguientes funciones lógicas:

$C_1 = G_0 + P_0 \cdot C_0$

$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot [G_0 + P_0 \cdot C_0] = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

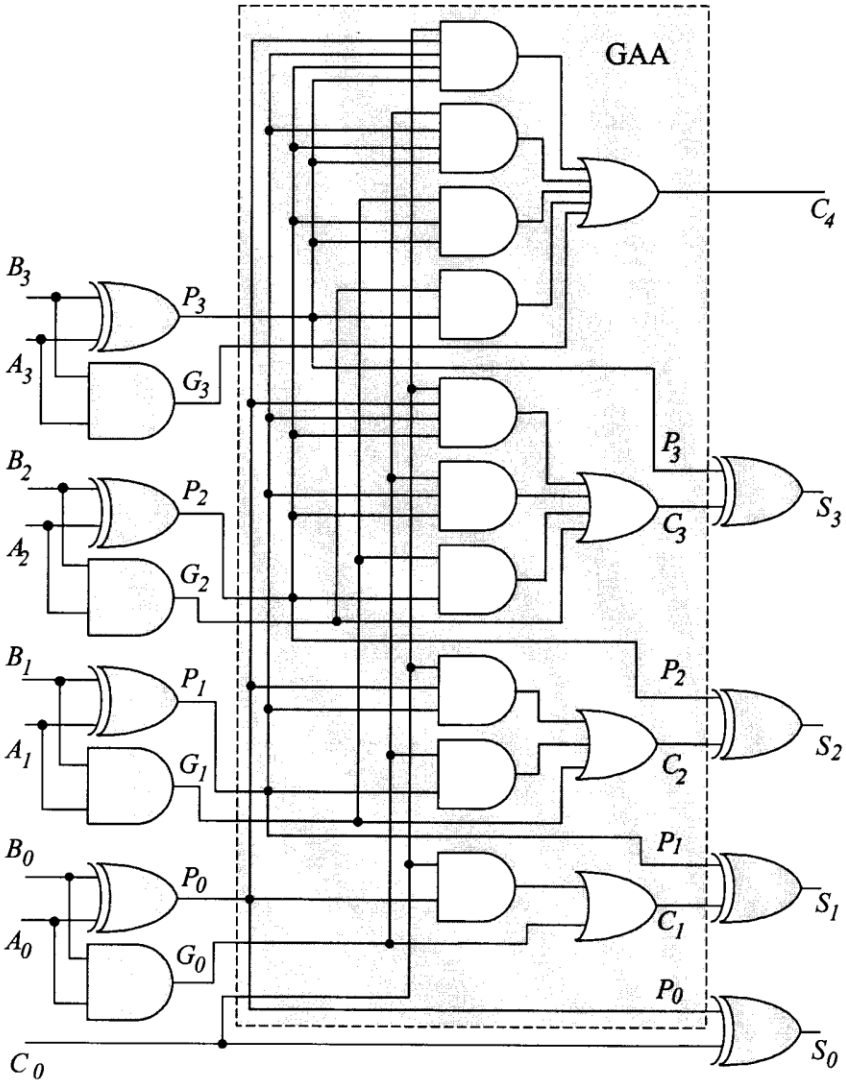
$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot [G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0] = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$

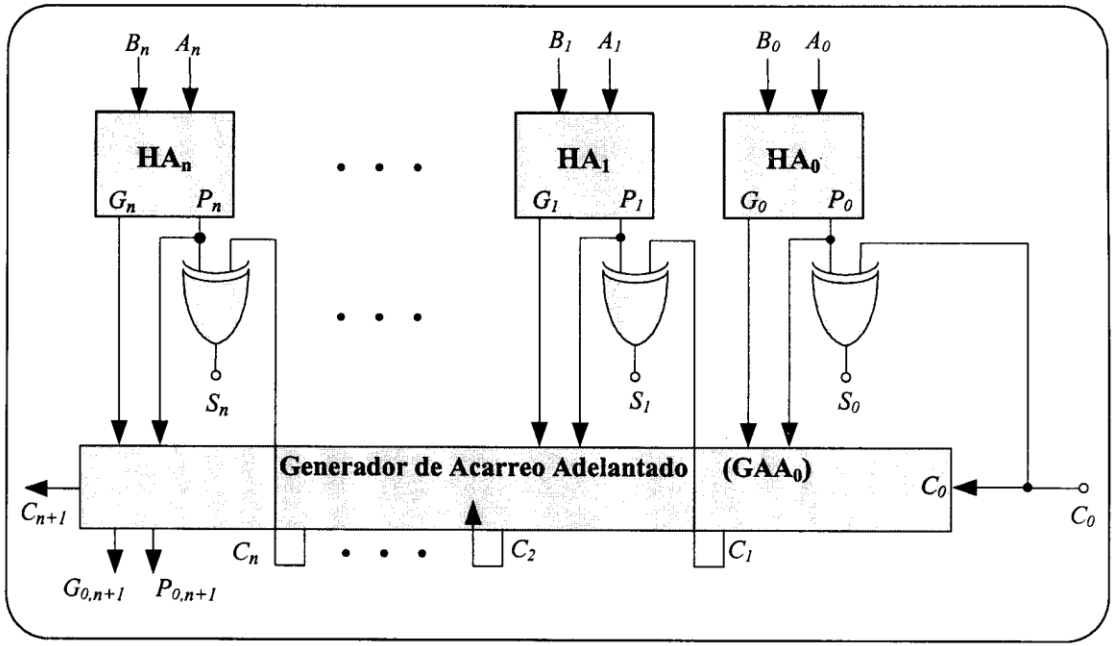
$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot [G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0] =$
$= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

- El sumador total implementará además:

$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$

Sumador paralelo con acarreo adelantado





Programa

1. Representación conjunta de números positivos y negativos.
2. Sumadores y restadores.
3. Sumadores en complemento a 1.
4. Comparadores.
5. Unidades aritmético-lógicas.

Sumadores en complemento a 1

- Vamos a ver qué modificaciones hay que incorporar a un sumador para que realice sumas de números en complemento a 1.

- Al sumar números en complemento a 1:
  1. Se suman las parejas de bits propagando el acarreo (como hemos visto en apartados anteriores).
  2. Se desprecia el acarreo saliente de la pareja de bits de mayor peso.
  3. Si el acarreo saliente anterior es igual a 1, esto indica que el resultado es incorrecto, y hay que sumarle 1 para corregirlo.

- En cualquier caso, se puede producir **rebose** (desbordamiento) cuando el resultado necesita ser representado con más bits que los operandos originales.
  - El rebose sólo se puede dar al sumar dos números del mismo signo (porque el módulo del resultado es mayor que el de los sumandos).
  - No se puede dar rebose al sumar dos números de distinto signo (porque el módulo del resultado es menor que el del sumando de mayor módulo).
  - **El rebose se detecta porque, en caso de producirse, el signo del resultado sale mal**.

Sumadores en complemento a 1

- Ejemplos para n=2 bits:

C-1	00	01	10	11
Decimal	+0	+1	-1	-0

  - Resultado que sale bien sin necesidad de aplicar corrección:

```
   00     (+0)          01     (+1)
 + 01     (+1)        + 10     (-1)
 ----                 ----
   01  →  (+1)          11  →  (-0)
```

  - Resultado que necesita corrección:

```
   10     (-1)
 + 11     (-0)
 ----
  101  →  (+1)
```

```
   10     (-1)
 + 11     (-0)
 ----
  101
    └→ + 1
 ----
   10  →  (-1)
```

  - Resultado incorrecto que sale mal a causa del rebose (para que saliese bien habría que aumentar el tamaño de los datos de entrada y del sumador):

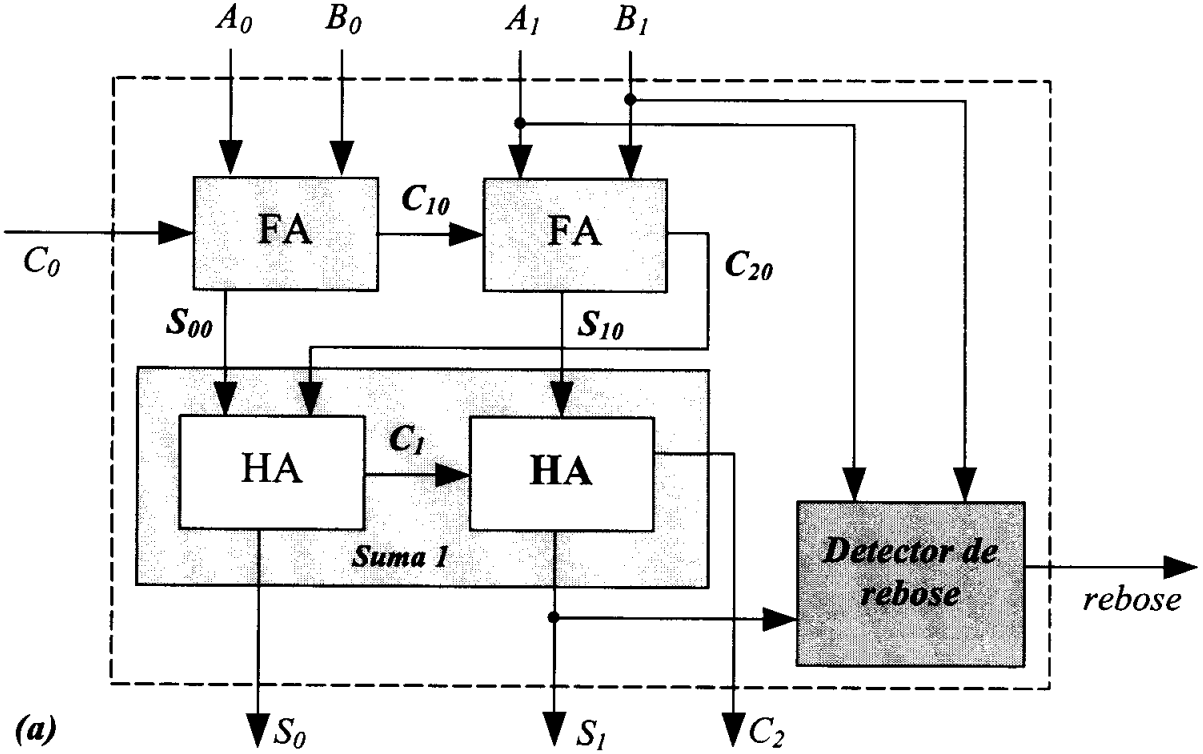
```
   01     (+1)
 + 01     (+1)
 ----
  010  →  (-1)
```

Sumadores en complemento a 1

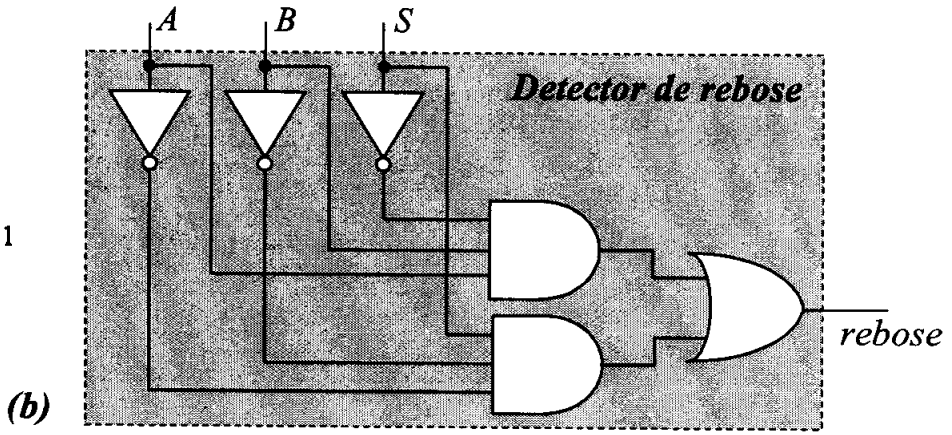
- El circuito sumador de números en complemento a 1 debe:

1. Sumar en binario puro todos los bits de los operandos, sin distinguir signo de magnitud.
2. Si el acarreo es 0 y no hay rebose, el resultado es correcto.
3. Si el acarreo es 1 y no hay rebose, sumar un 1 al resultado.
4. Si hay rebose (porque el signo del resultado es incorrecto), dar señal de error.
   - Ecuación del rebose: $rebose = \overline{A}_1\ \overline{B}_1\ S_1 + A_1\ B_1\ \overline{S}_1$

Sumadores en complemento a 1



$$rebose = \overline{A}_1\ \overline{B}_1\ S_1 + A_1\ B_1\ \overline{S}_1$$



Programa

1. Representación conjunta de números positivos y negativos.
2. Sumadores y restadores.
3. Sumadores en complemento a 1.
4. Comparadores.
5. Unidades aritmético-lógicas.

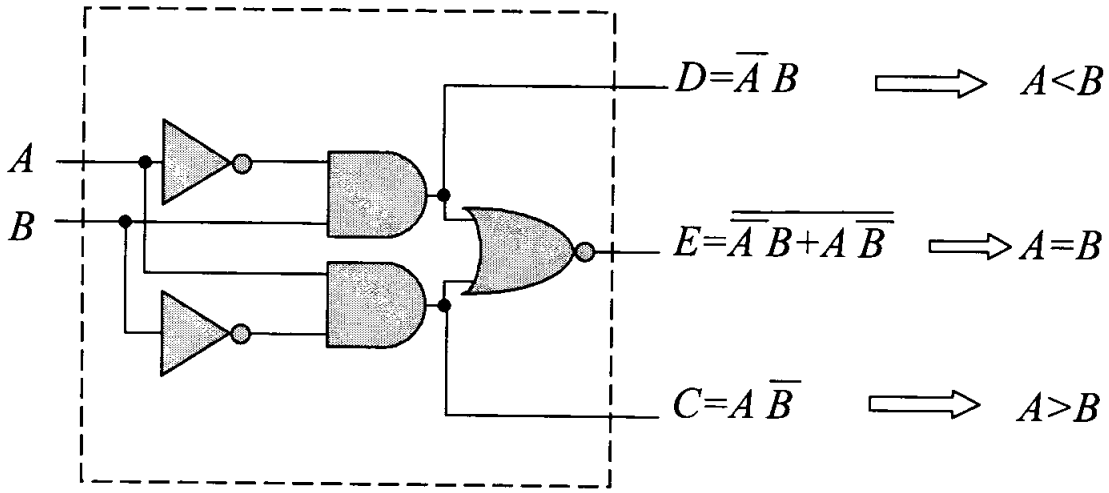
Comparadores

- Un comparador de dos palabras de **n** bits A y B es un circuito que determina cuál de estas palabras representa un valor mayor, cuál representa un valor menor y cuándo representan el mismo valor.

- Un comparador produce tres salidas, cuyas ecuaciones booleanas son:
  - A > B: $C = A \cdot \overline{B} = 1$
  - A < B: $D = \overline{A} \cdot B = 1$
  - A = B: circuito coincidencia (complementario del OR exclusivo)

$$E = \overline{A\overline{B} + \overline{A}B} = \begin{cases} 1 & si \quad A = B \\ 0 & si \quad A \neq B \end{cases}$$

- Circuito total:



Comparadores

- Con palabras de **n** bits, hay que extender las condiciones:
  - A = B:

$A_3=B_3$, $A_2=B_2$, $A_1=B_1$ y $A_0=B_0$ ⟹ $E = E_3 \cdot E_2 \cdot E_1 \cdot E_0 = 1$

  - A > B: es el OR de las siguientes expresiones:

$$A_3 > B_3 \Rightarrow A_3 \overline{B}_3$$

$$\acute{o} \quad A_3 = B_3 \;\; y \;\; A_2 > B_2 \Rightarrow E_3 A_2 \overline{B}_2$$

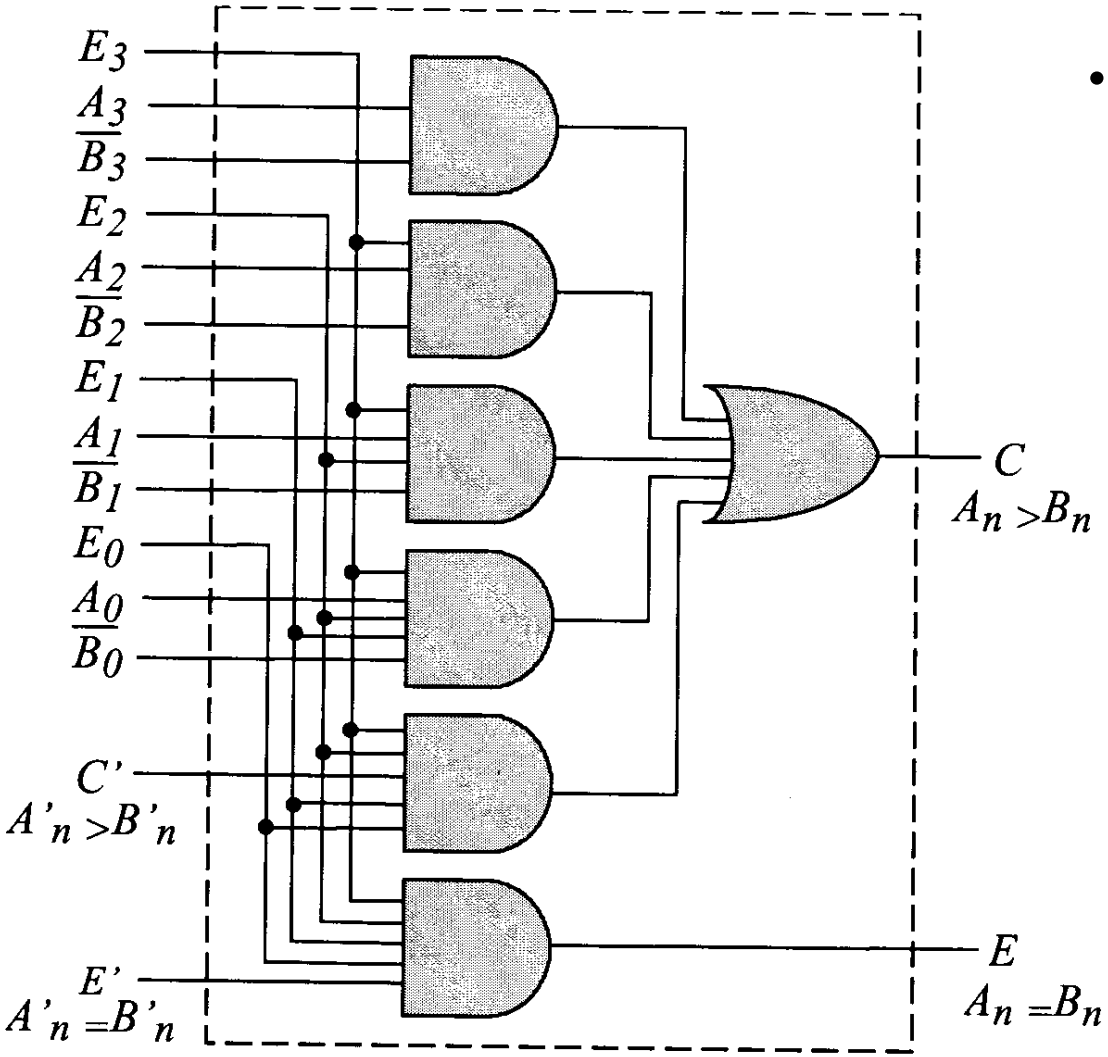
$$\acute{o} \quad A_3 = B_3 \;\; y \;\; A_2 = B_2 \;\; y \;\; A_1 > B_1 \Rightarrow E_3 E_2 A_1 \overline{B}_1$$

$$\acute{o} \quad A_3 = B_3 \;\; y \;\; A_2 = B_2 \;\; y \; A_1 = B_1 \;\; y \;\; A_0 > B_0 \Rightarrow E_3 E_2 E_1 A_0 \overline{B}_0$$

  - A < B: cuando E=0 y C(A>B) = 0.

Comparadores

- Circuito comparador para palabras de 4 bits:



- C′ y E′: entradas para conexión en cascada.
  - Proceden de otro comparador que compara bits de menor peso.

Comparadores

- Circuito comparador comercial de palabras de 8 bits: SN74AS866A.

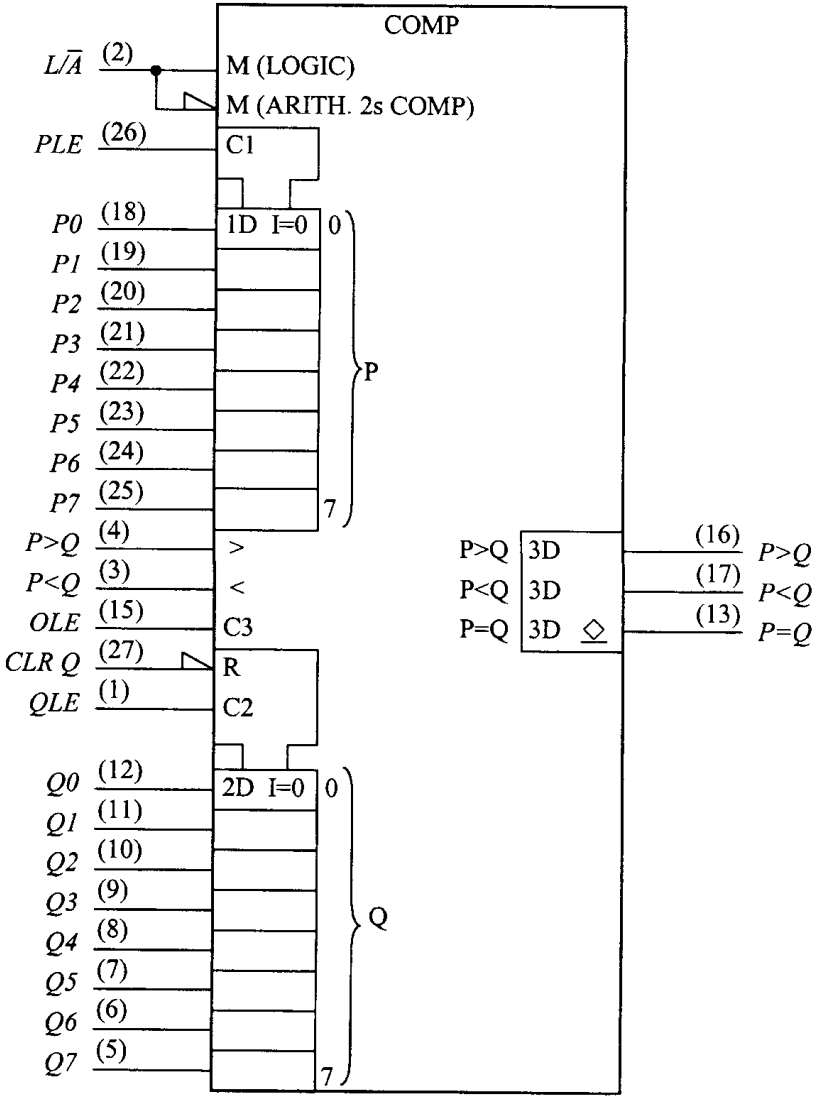


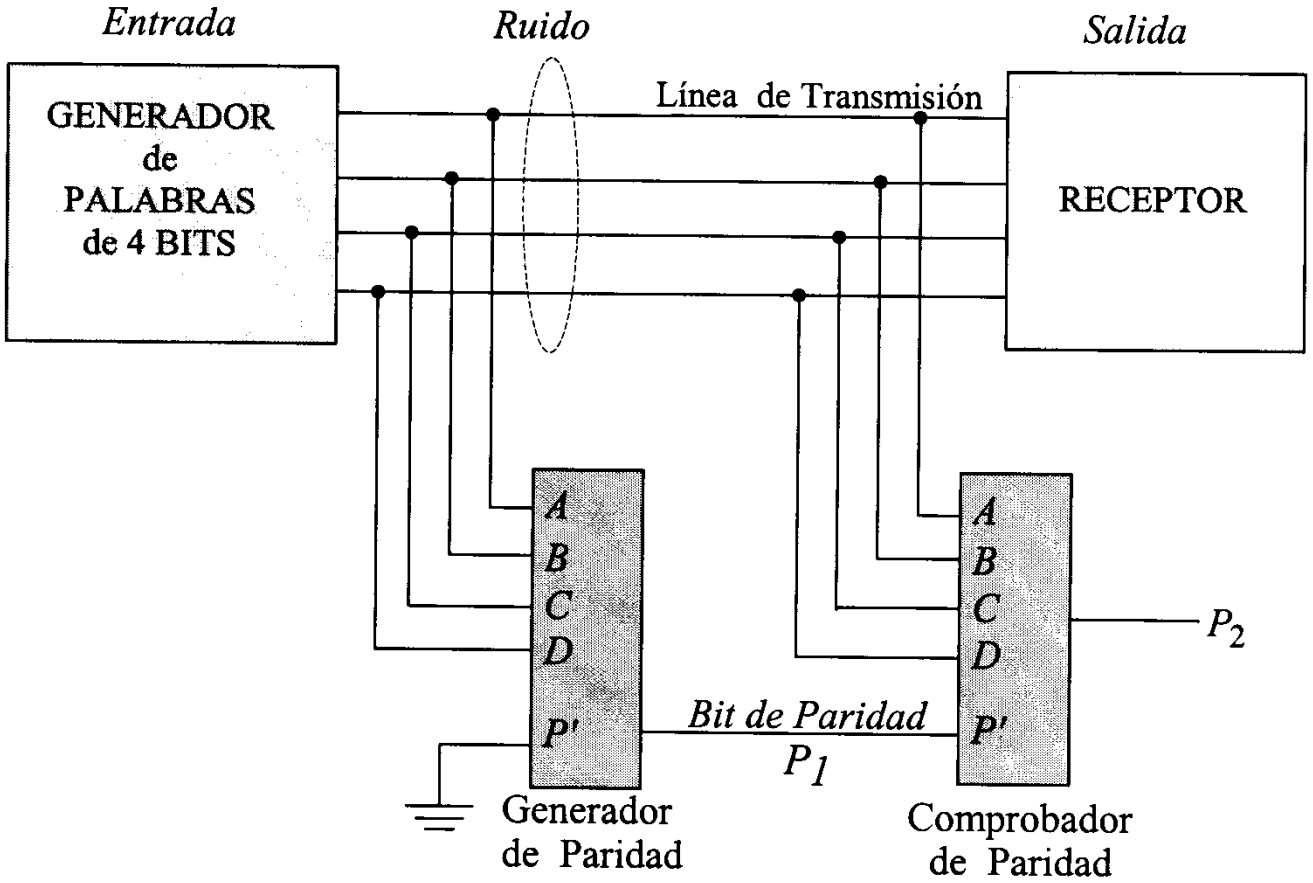
TABLA DE FUNCIÓN DEL COMPARADOR

Comparación	L/Ā	*Datos de Entrada P0-P7, Q0-Q7*	*ENTRADAS* P>Q	P<Q	*SALIDAS* P>Q	P<Q	P=Q
Lógica	H	P>Q	x	x	***H***	***L***	***L***
Lógica	H	P<Q	x	x	***L***	***H***	***L***
Lógica	H	P=Q	L	L	***L***	***L***	***H***
Lógica	H	P=Q	L	H	***L***	***H***	***L***
Lógica	H	P=Q	H	L	***H***	***L***	***L***
Lógica	H	P=Q	H	H	***H***	***H***	***L***
Aritmética	L	P AG Q	x	x	***H***	***L***	***L***
Aritmética	L	Q AG P	x	x	***L***	***H***	***L***
Aritmética	L	P=Q	L	L	***L***	***L***	***H***
Aritmética	L	P=Q	L	H	***L***	***H***	***L***
Aritmética	L	P=Q	H	L	***H***	***L***	***L***
Aritmética	L	P=Q	H	H	***H***	***H***	***L***

Generadores y detectores de paridad

- Los circuitos generadores y detectores de paridad se activan cuando la suma de los unos de la palabra de entrada es par (**paridad par**) o bien cuando la suma de los unos de dicha palabra es impar (**paridad impar**).

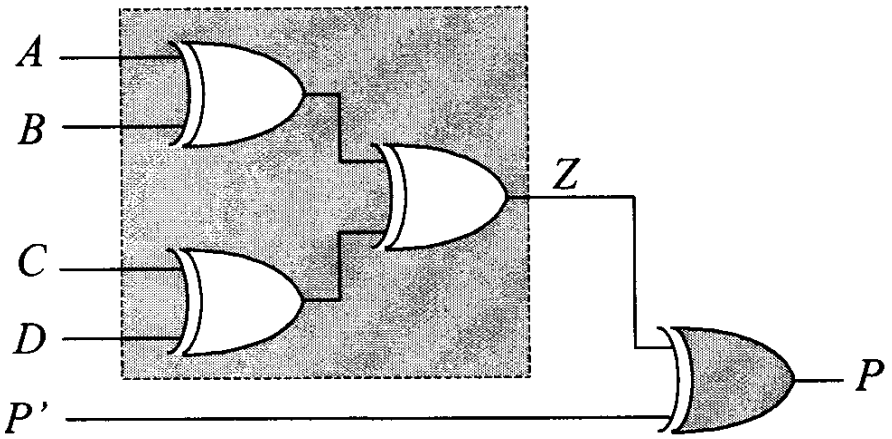
- Se suelen usar para detectar errores en transmisión de datos.



Generadores y detectores de paridad

- Síntesis de circuitos detectores de paridad: a partir de la función OR exclusivo.
  - El OR exclusivo de dos bits es 0 cuando son iguales: dos bits a 1 o dos bits a 0 (número par de unos).
  - El OR exclusivo de dos bits es 1 cuando son distintos: un 1 y un 0 (número impar de unos).
  - El OR exclusivo da la NO PARIDAD de los datos de entrada (para calcular la paridad, el generador de paridad debe negar el OR exclusivo de los bits de entrada).
  - La operación OR exclusivo es asociativa, por lo que se puede ampliar fácilmente a más bits.

- Ejemplo: comprobador de paridad de 4 bits de entrada A, B, C y D.
  - Bit Z: no paridad de A, B, C y D.
  - Se puede conectar en cascada mediante puertas XOR adicionales (P': paridad de otro módulo comprobador).

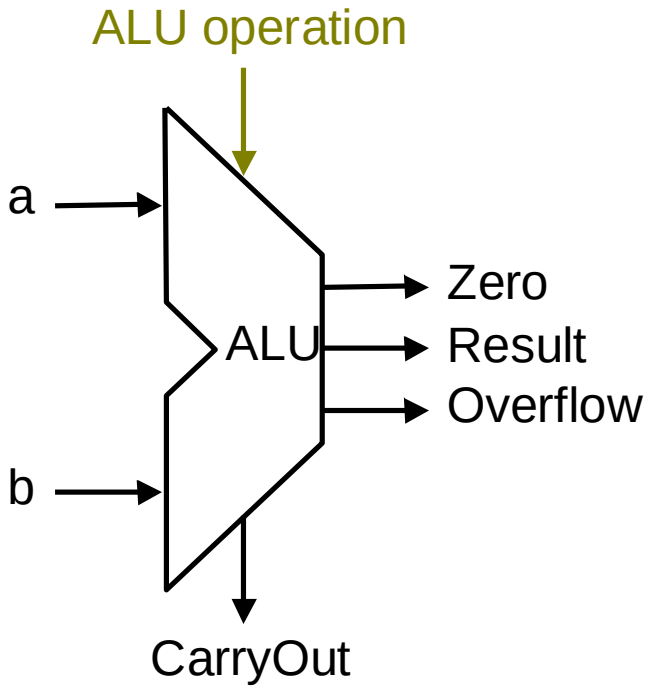


Programa

1. Representación conjunta de números positivos y negativos.
2. Sumadores y restadores.
3. Sumadores en complemento a 1.
4. Comparadores.
5. Unidades aritmético-lógicas.

Unidades aritmético-lógicas

- Una unidad aritmético-lógica (UAL, ALU en inglés) es un circuito capaz de realizar varias operaciones aritméticas y lógicas con los datos de entrada.
  - La salida de una ALU es el resultado de una de las operaciones que es capaz de hacer.
  - La salida concreta se selecciona en función de unas **entradas de control** que permiten indicar la operación deseada.
  - Aparte del resultado propiamente dicho, la ALU puede proporcionar otros bits de salida, tales como:
    - El acarreo superior.
    - El bit de rebose.
    - Un bit indicador de resultado nulo.
    - El bit de signo del resultado.



Unidades aritmético-lógicas

- Tabla de verdad de la unidad aritmético-lógica comercial SN74181.

SELECCIÓN				DATO ACTIVO EN ALTA		
				M=H	M=L Operaciones Aritméticas	
S_3	S_2	S_1	S_0	Funciones Lógicas	$\overline{c_n}$=H (sin acarreo)	$\overline{c_n}$=L (con acarreo)
L	L	L	L	$F = \overline{A}$	$F = A$	$F = A$ PLUS *1*
L	L	L	H	$F = \overline{A + B}$	$F = A + B$	$F = (A + B)$ PLUS *1*
L	L	H	L	$F = \overline{A}\,B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS *1*
L	L	H	H	$F = 0$	F = MINUS *1* (comp. a 2)	$F = Cero$
L	H	L	L	$F = \overline{A\,B}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS *1*
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS *1*
L	H	H	L	$F = A \oplus B$	$F = A$ MINUS B MINUS *1*	$F = A$ MINUS B
L	H	H	H	$F = A\,\overline{B}$	$F = A\,\overline{B}$ MINUS *1*	$F = A\,\overline{B}$
H	L	L	L	$F = \overline{A} + B$	$F = A$ PLUS AB	$F = A$ PLUS AB PLUS *1*
H	L	L	H	$F = \overline{A \oplus B}$	$F = A$ PLUS B	$F = A$ PLUS B PLUS *1*
H	L	H	L	$F = B$	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS *1*
H	L	H	H	$F = A\,B$	$F = A\,B$ MINUS *1*	$F = A\,B$
H	H	L	L	$F = 1$	$F = A$ PLUS A	$F = A$ PLUS A PLUS *1*
H	H	L	H	$F = A + \overline{B}$	$F = (A + B)$ PLUS A	$F = (A + B)$ PLUS A PLUS *1*
H	H	H	L	$F = A + B$	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS *1*
H	H	H	H	$F = A$	$F = A$ MINUS *1*	$F = A$

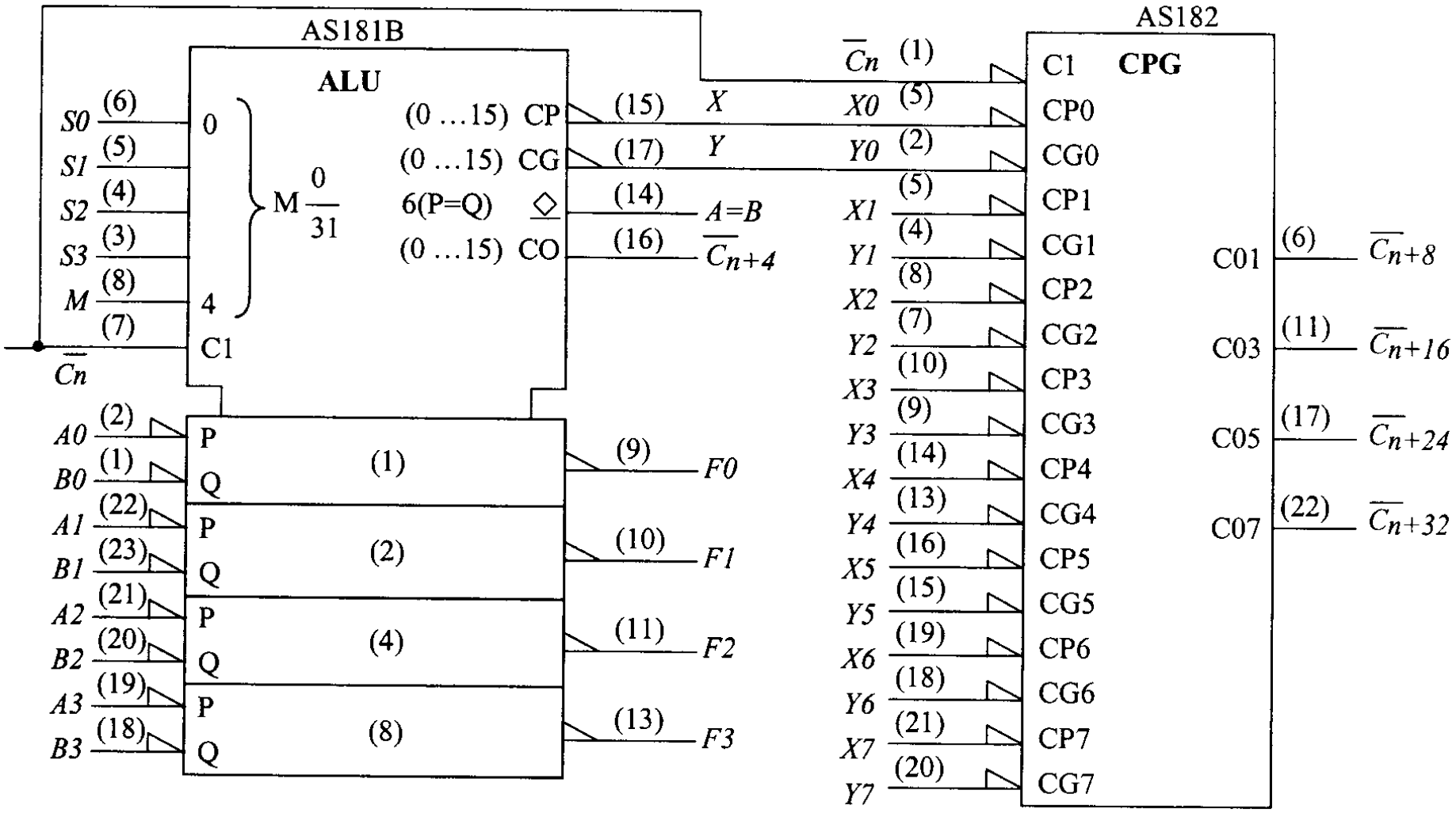
Entradas de control:

- M: distingue entre operación aritmética o lógica.
- $S_3S_2S_1S_0$: selección de operación.

- **Significado de símbolos:**
- **+** significa OR lógico.
- ⊕ significa OR exclusivo lógico.
- El producto lógico se sobreentiende.
- **PLUS** significa suma aritmética.
- **MINUS** significa resta aritmética mediante suma en complemento a 1.

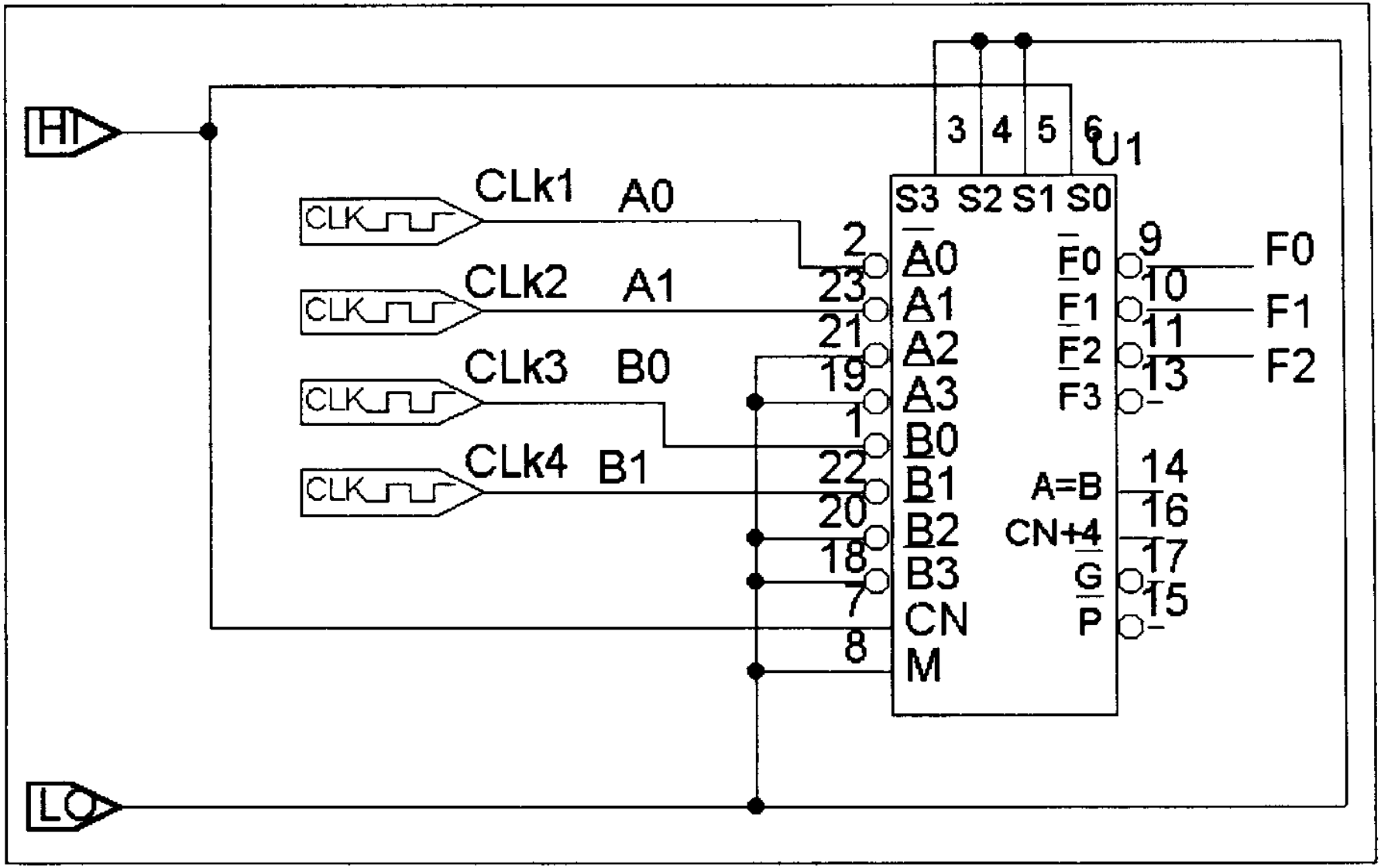
Unidades aritmético-lógicas

- Ejemplo de conexión de una ALU SN74181 con un anticipador de acarreo SN74182.



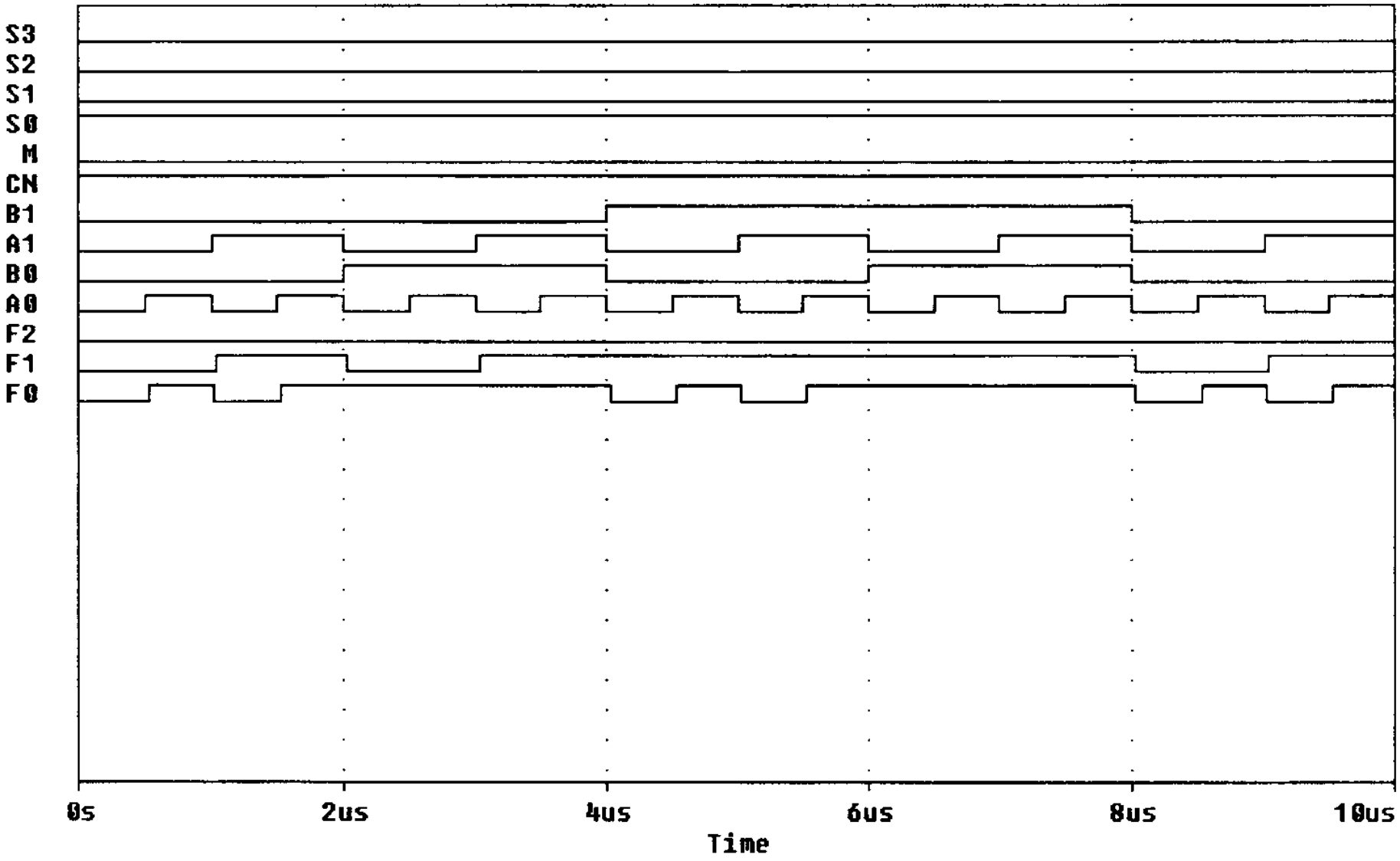
Unidades aritmético-lógicas

- Ejemplo de uso para comprobar la operación A+B (OR).



Unidades aritmético-lógicas

- Cronograma del circuito anterior:



Unidades aritmético-lógicas

- Tabla de verdad y simplificación para comprobar la función realizada:

B1	*A1*	*B0*	*A0*	*F2*	*F1*	*F0*
0	0	0	0	*0*	*0*	*0*
0	0	0	1	*0*	*0*	*1*
0	1	0	0	*0*	*1*	*0*
0	1	0	1	*0*	*1*	*1*
0	0	1	0	*0*	*0*	*1*
0	0	1	1	*0*	*0*	*1*
0	1	1	0	*0*	*1*	*1*
0	1	1	1	*0*	*1*	*1*
1	0	0	0	*0*	*1*	*0*
1	0	0	1	*0*	*1*	*1*
1	1	0	0	*0*	*1*	*0*
1	1	0	1	*0*	*1*	*1*
1	0	1	0	*0*	*1*	*1*
1	0	1	1	*0*	*1*	*1*
1	1	1	0	*0*	*1*	*1*
1	1	1	1	*0*	*1*	*1*

B1 A1 / *B0 A0*	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$\overline{F0} = \overline{A0}\ \overline{B0}$ $F0 = A0 + B0$

B1 A1 / *B0 A0*	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$F1 = A1 + B1$