66.70 Estructura del Computador

Unidades aritméticas

Operaciones aritméticas

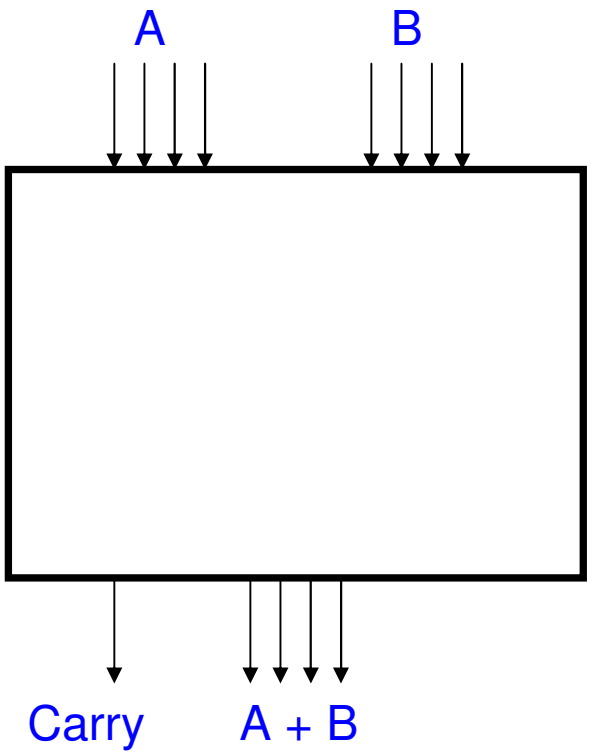
*Realizar operaciones aritméticas (+ - * /) es lo más elemental que se le puede pedir a una computadora.*

Sin embargo,

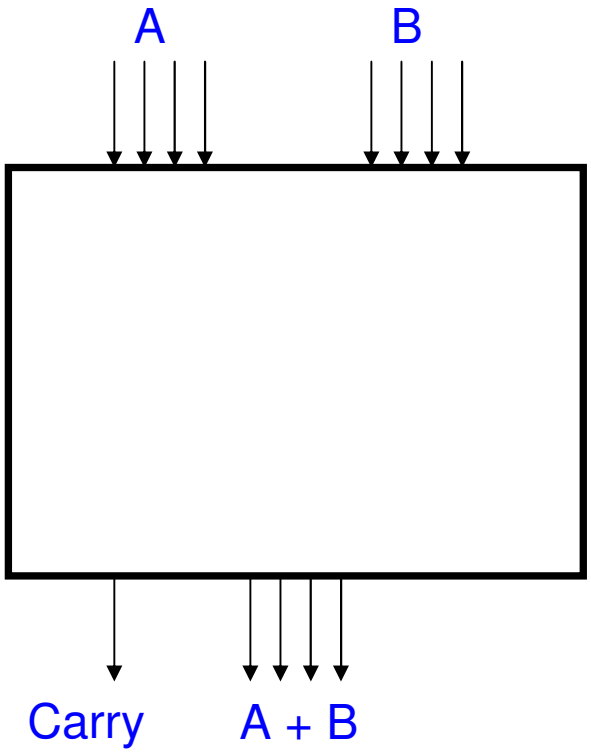
- No son tan simples de implementar como podría esperarse

- Su implementación forma parte de la Unidad Aritmético Lógica (ALU) que es el núcleo mismo del microprocesador

Sumador



Sumador



- Diseño con lógica de 2 niveles -

Cantidad de

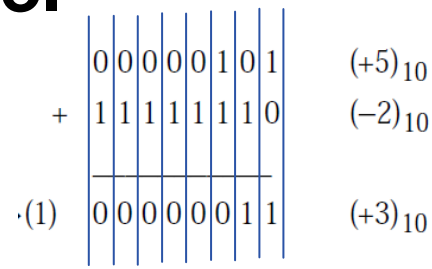
- variables de entrada
- minitérminos por cada salida
- compuertas
- entradas por compuerta
- sumadores de 32 bits

Buscar otra solución

Suma de números de N dígitos

Operar como en lápiz y papel

(dígito a dígito encolumnados)



El circuito puede **dividirse** en

sumadores simples

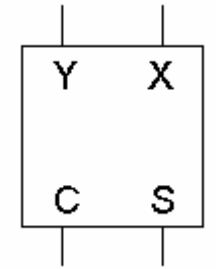
(suma de 2 bits)

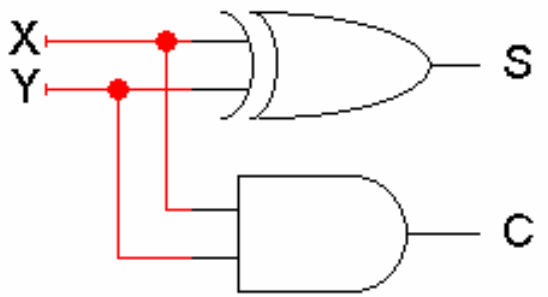
Sumador de dos bits
semisumador o half adder

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = X \; xor \; Y$$
$$C = X \,.\, Y$$





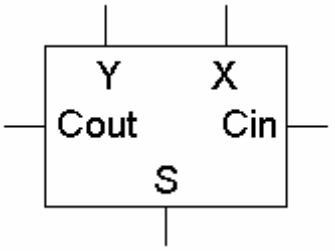
Sumador de tres bits
sumador completo o full adder

X	Y	C_{in}	C_{out}	S
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

$S = \Sigma m(1,2,4,7)$
$= X' Y' C_{in} + X' Y C_{in}' + X Y' C_{in}' + X Y C_{in}$
$= X' (Y' C_{in} + Y C_{in}') + X (Y' C_{in}' + Y C_{in})$
$= X' (Y \oplus C_{in}) + X (Y \oplus C_{in})'$
$= X \oplus Y \oplus C_{in}$

$C_{out} = \Sigma m(3,5,6,7)$
$= X' Y C_{in} + X Y' C_{in} + X Y C_{in}' + X Y C_{in}$
$= (X' Y + X Y') C_{in} + XY(C_{in}' + C_{in})$
$= (X \oplus Y) C_{in} + XY$

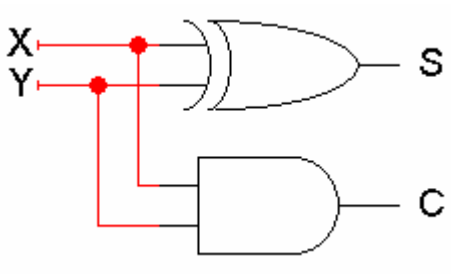
$$S = X \oplus Y \oplus Cin$$
$$Cout = (X \oplus Y)\, C_{in} + XY$$



Sumador de tres bits

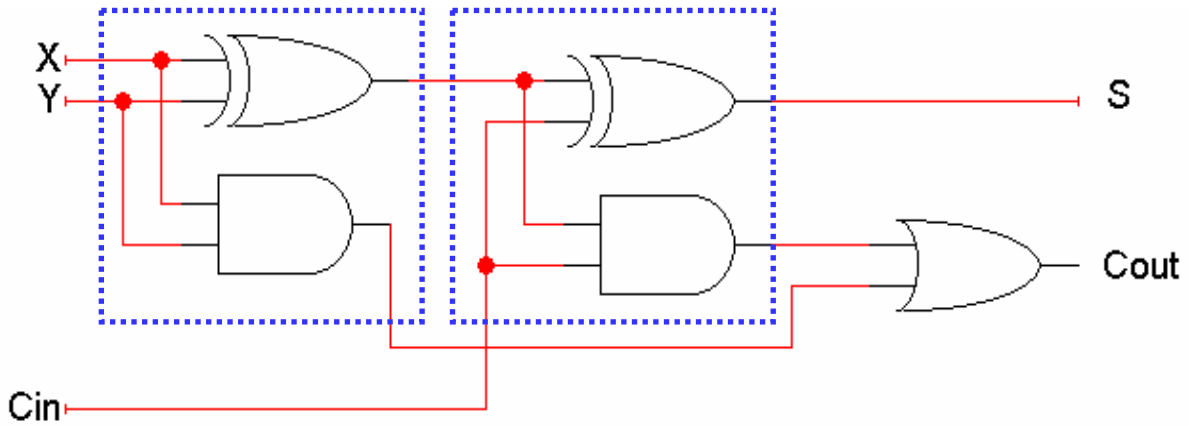
Sumador completo en base a 2 semisumadores

$$S = X \oplus Y$$
$$C = X . Y$$



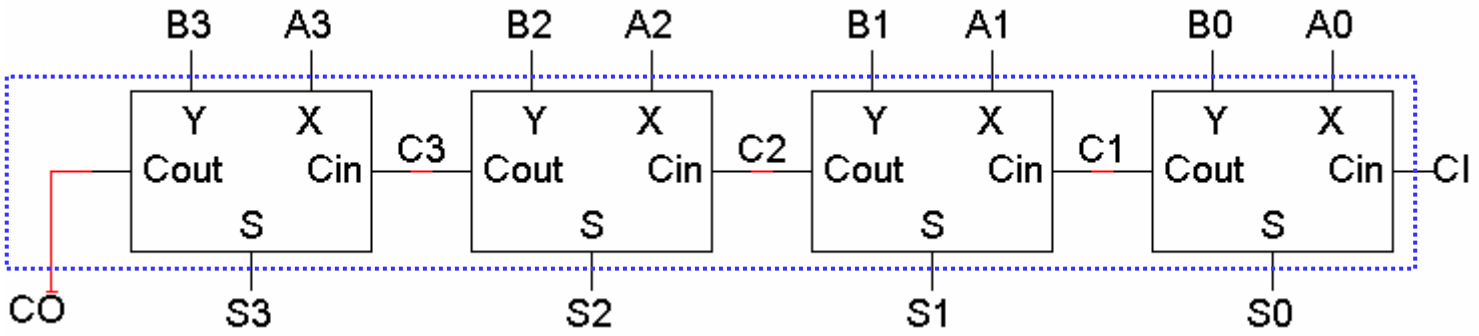
Semisumador

$$S = X \oplus Y \oplus C_{in}$$
$$C_{out} = (X \oplus Y).C_{in} + XY$$



Sumador completo

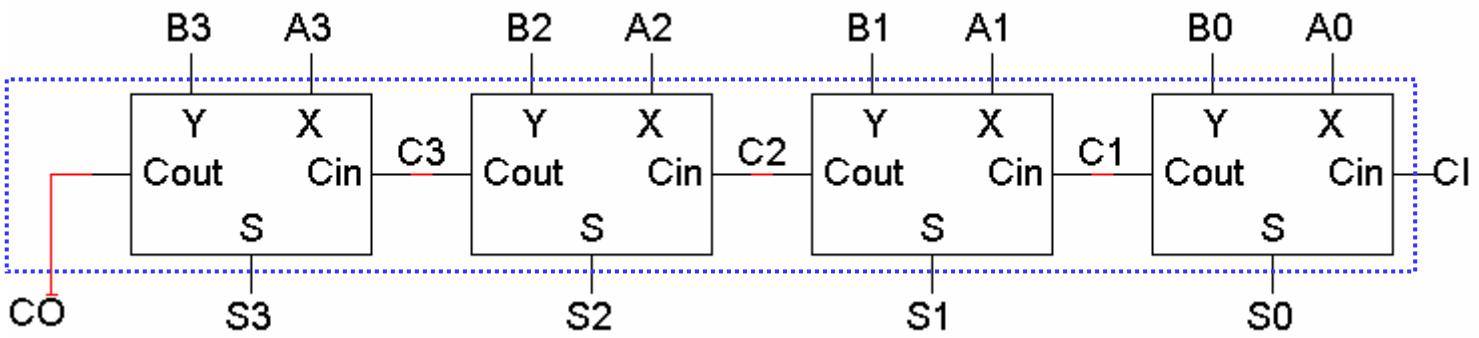
Sumador de dos números de 4 bits



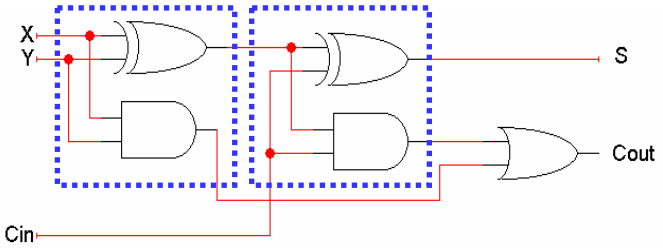
- Para qué es necesario el C_{out} ?
- Para qué es necesario el C_{in} ?

- Suma de números sin signo
- Suma de números en complemento a 2

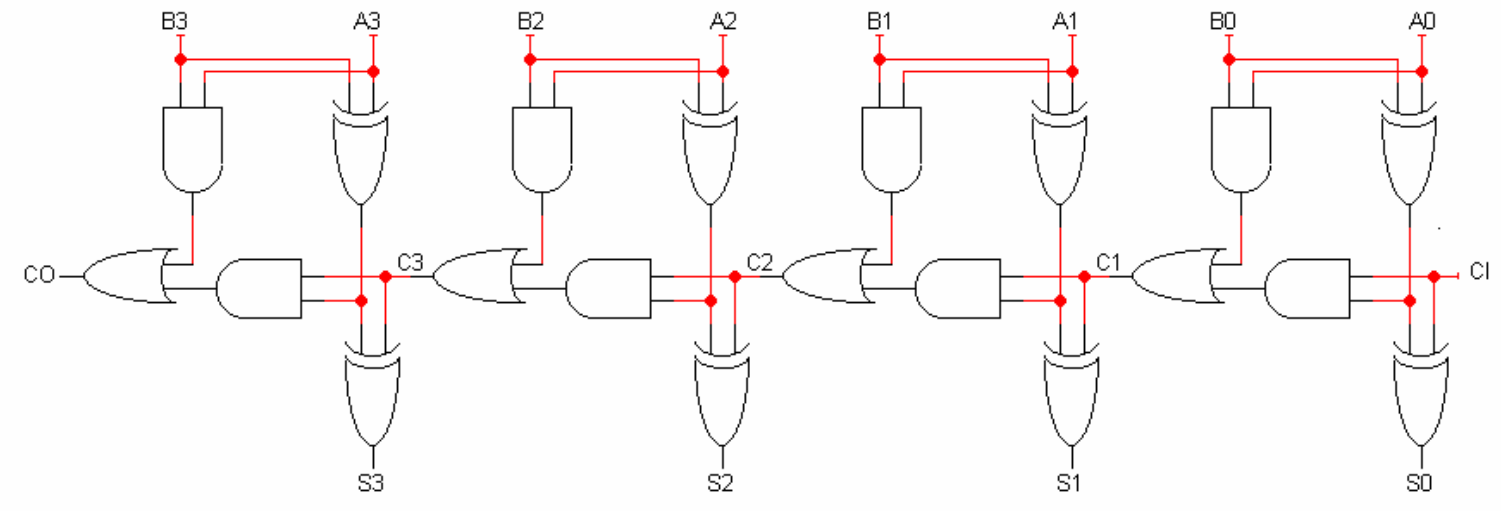
Velocidad de operación



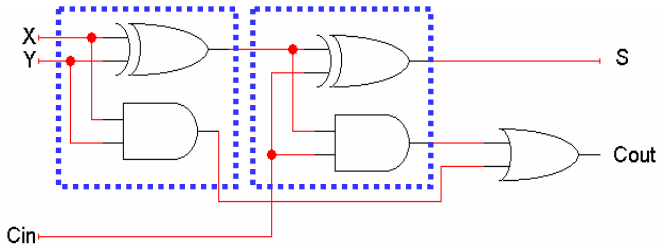
❖ Qué parte de la estructura de este sumador es el principal limitante a su velocidad?



Velocidad de operación

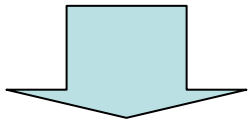


- Largo camino entre (A0, B0, CI) hasta CO y S3.
- Estimar el retardo en un sumador de 64-bit



Sumadores de alta velocidad

“Cuello de botella” = Carry



Existen soluciones a ese problema

- **“Carry look-ahead”**
- **“Carry save”**

Resta

- ✓ *Podemos diseñar un restador del mismo modo en que planteamos el circuito del sumador.*

- ✓ Pero también podemos:
  Usar complemento a 2 y convertir cualquier resta en un problema de suma A - B = A + (-B)

Basar la resta en complemento a 2 es ventajoso porque:

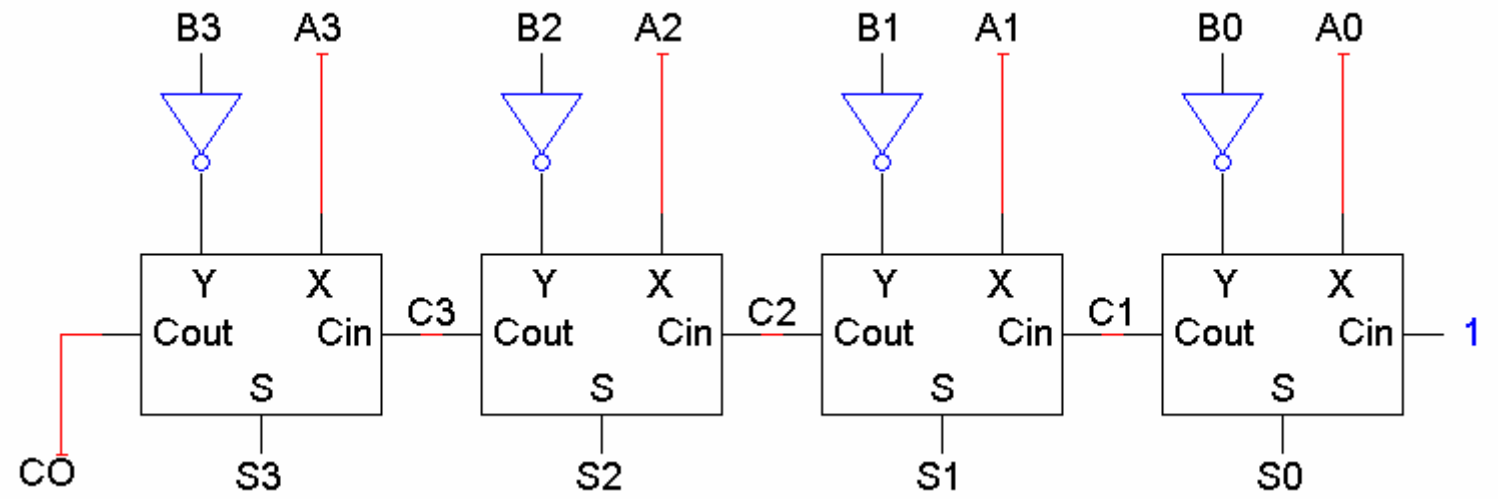
- Reutilizamos el circuito del sumador
- Es simple obtener el complemento a 2 de un número

Resta

Las únicas diferencias con el circuito sumador son:

→ *El negador complementa cada bit del sustraendo B3 B2 B1 B0.*

→ *En el restador el C_{in} es 1 en vez de 0*



Es posible obtener un circuito sumador/restador?

Multiplicación

de dos palabras de 1 bit

- **La multiplicación de dos bits es equivalente a la función *AND***

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

Valores **lógicos**

a	b	a×b
0	0	0
0	1	0
1	0	0
1	1	1

Valores **numéricos**

Multiplicación

de dos palabras de 1 bit

- **La multiplicación de dos bits es equivalente a la función *AND***

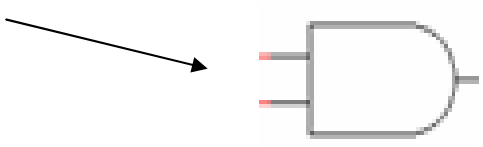
a	b	a·b
0	0	0
0	1	0
1	0	0
1	1	1

Valores **lógicos**

a	b	a×b
0	0	0
0	1	0
1	0	0
1	1	1

Valores **numéricos**

Multiplicador de
2 bits



Multiplicación
de dos palabras de 4 bits

- **La multiplicación de dos números binarios sigue el método del papel y lápiz**
- **Se basa en <u>sumas</u> y <u>desplazamientos</u>**

				1	1	0	1	Multiplicando
			×	0	1	1	0	Multiplicador
				0	0	0	0	Productos parciales
+			1	1	0	1		
		1	1	0	1			
	0	0	0	0				
	1	0	0	1	1	1	0	Producto

- **En binario los productos parciales sólo pueden dar:** *0000* *ó el multiplicando*

Multiplicación

de dos palabras de 4 bits

Inconveniente 1

- **Disponemos de sumadores de 2 sumandos**

…pero debemos sumar 4 números

Enteros sin signo

```
    1110
  x 0111
  ------
    1110
   1110
  1110
 0000
  ------
    ?010
```

solución

Por sumas parciales

```
    1110
  +1110
  ------
  101010
 +1110
  ------
 1100010
```

Inconveniente 2

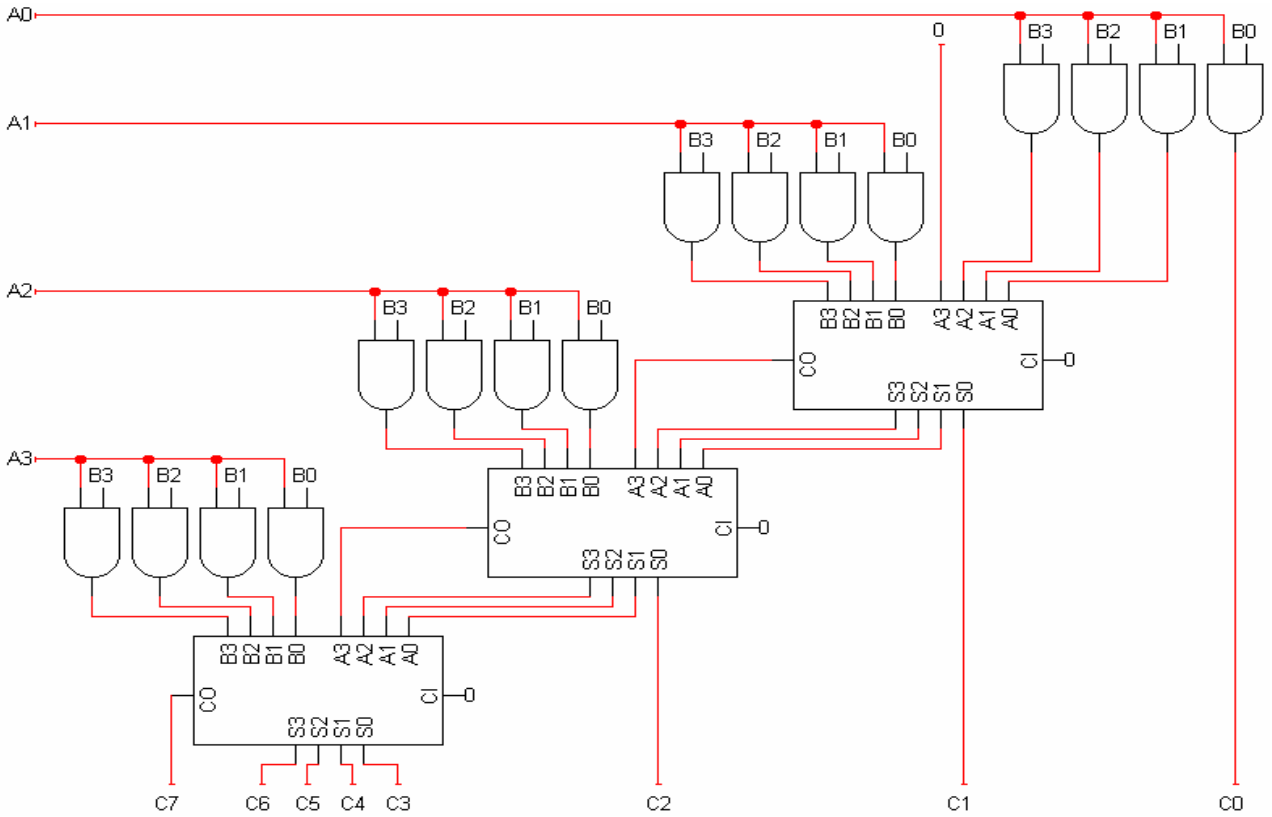
- **Tenemos sumadores de 4 bits pero la suma da un resultado de 8 bits**

solución

sumas de 4 bits y desplazamiento a izquierda en cada suma parcial

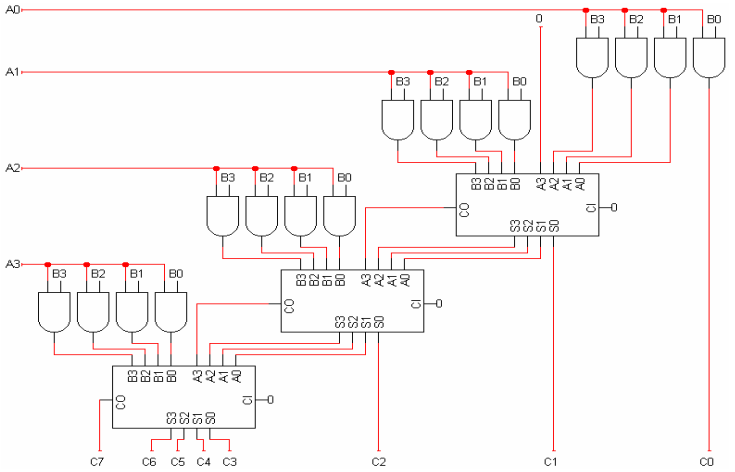
Multiplicador paralelo

de dos números de 4 bits



Multiplicador paralelo

de dos números de 4 bits



- ✓ Al multiplicar dos números de n-bits deben resolverse
  - Hasta n productos parciales (uno por cada 1 del multiplicador).
  - Hasta n-1 sumas
  - Cada sumador suma dos números de n bits (tamaño del multiplicando)

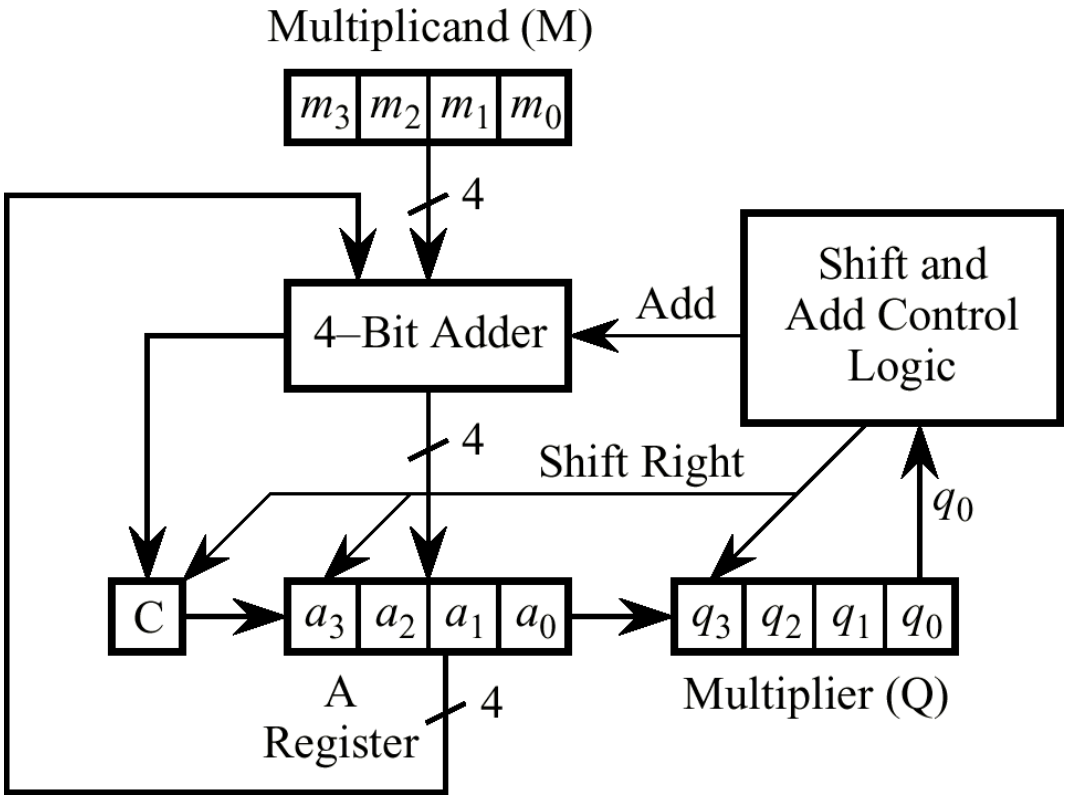
=> El circuito de un multiplicador paralelo de 32-bit o 64-bit es enorme!

➢ Una arquitectura **serie** admite hardware mucho **más simple**

…pero **más lenta**

Multiplicador serie

de dos números de 4 bits (sin signo)



Al inicio:

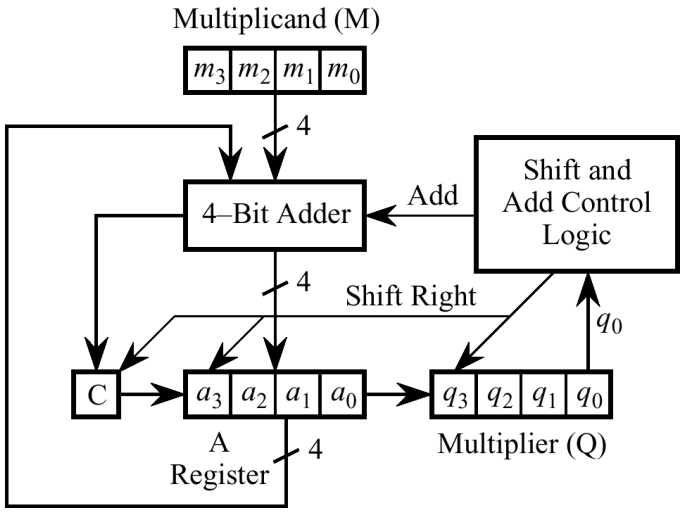
- Multiplicando en M
- Multiplicador en Q
- A=C=0

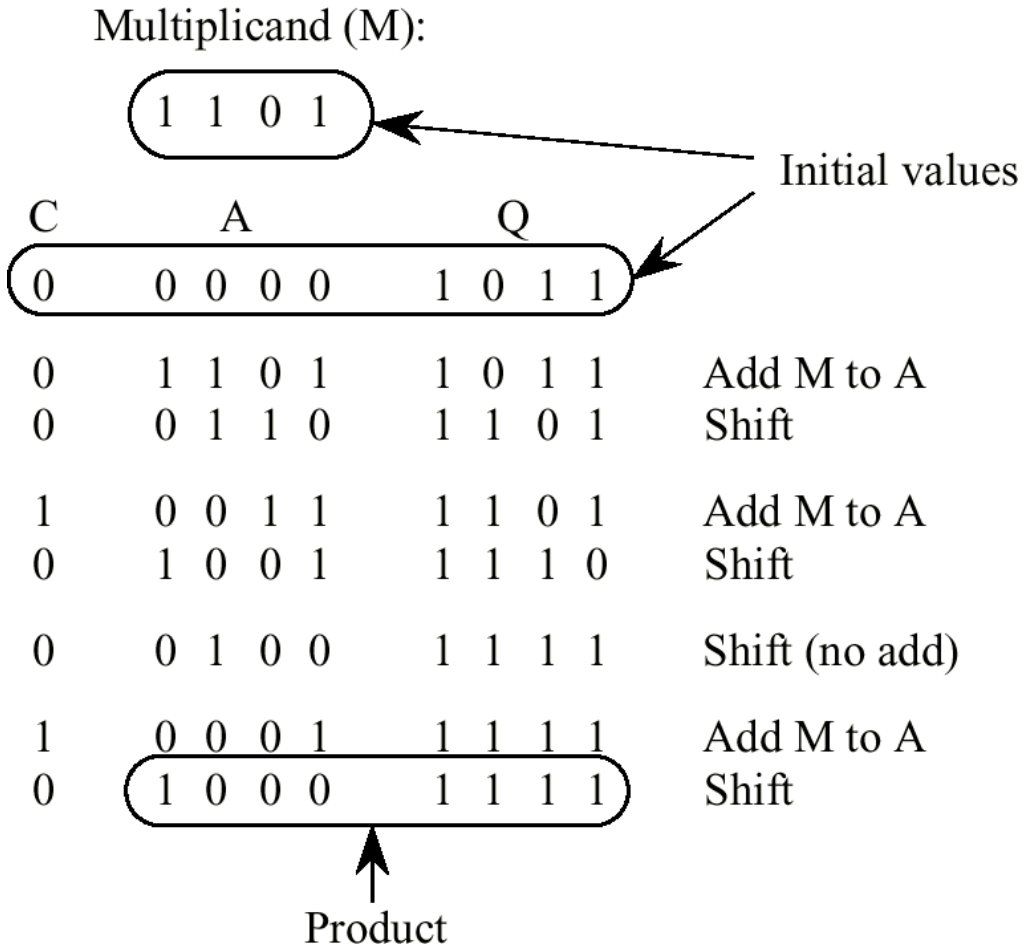
Resultado:

- A (bits más significativo)
- Q (bits menos significativo)

Multiplicador serie

de dos números de 4 bits (sin signo)





Multiplicación de enteros con signo

Multiplicando negativo, multiplicador positivo

```
    1110      (-2)10
  x 0101      (5)10
--------
    1110
   0000
  1110
 0000
--------
01110110
```

Multiplicación de enteros con signo

Multiplicando negativo, multiplicador positivo ←

```
    1110        (-2)10
  x 0101        (5)10
  ------
    1110
   0000
  1110
 0000
--------
01110110        (118)10
```

incorrecto

Extensión de signo

```
    1110
  x 0101
  ------
11111110
00000000
11111110
00000000
--------
11110110        (-10)10
```

- Se puede aplicar métodos similares con enteros sin signo y con ent. compl. a 2
- Pero en el caso de complemento a 2 es necesario extender el signo hasta completar 2 n bits

Multiplicación de enteros con signo

Multiplicador negativo ←

```
    0101        (5)10
  x 1110        (-2)10
  ------
    0000
   0101
  0101
 0101
 --------
 01000110       (70)10
```

incorrecto

Extensión de signo

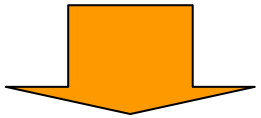
```
      0101
  x 11111110
  ----------
  00000000
    0101
   0101
  0101
 0101
0101
 0101
  0101
   0101
    0101
    --------
    01011110110      (-10)10
```

- Se debe extender el signo del multiplicador
- Si el multiplicando fuese también negativo se extiende su signo también
- Del resultado sólo tomamos los *2n* bits menos significativos

Enteros con signo
Suma vs. Multiplicación

- La **suma** binaria se realiza en forma idéntica en enteros con signo y sin signo.
  => se obtiene la misma combinación de 0`s y 1`s como resultado
- El resultado es interpretado de modo diferente en cada caso

- La **multiplicación** binaria se realiza en forma diferente en enteros con signo y sin signo.
- El resultado está formado por diferentes combinaciones de 0`s y 1's que además son interpretadas en forma diferente (complemento a 2 o sin signo)
- En ambos casos el producto tiene el doble de bits que cada operando



En el set de instrucciones de **Pentium**

- ✓ hay una única instrucción para la suma (**ADD**). El resultado es interpretado por el programador como con signo o como sin signo.
- ✓ hay dos instrucciones de multiplicación: **MUL** (sin signo) e **IMUL** (con signo)
- ✓ El resultado de ADD se guarda en **1 registro**
- ✓ El resultado de MUL o IMUL se guarda en **2 registros**

Aumentando la velocidad de los circuitos multiplicadores

- En el método clásico la velocidad de la multiplicación está condicionada por el número de sumas

Necesitamos reducir la cantidad de sumas

- multiplicar por 01110 requiere 3 productos parciales

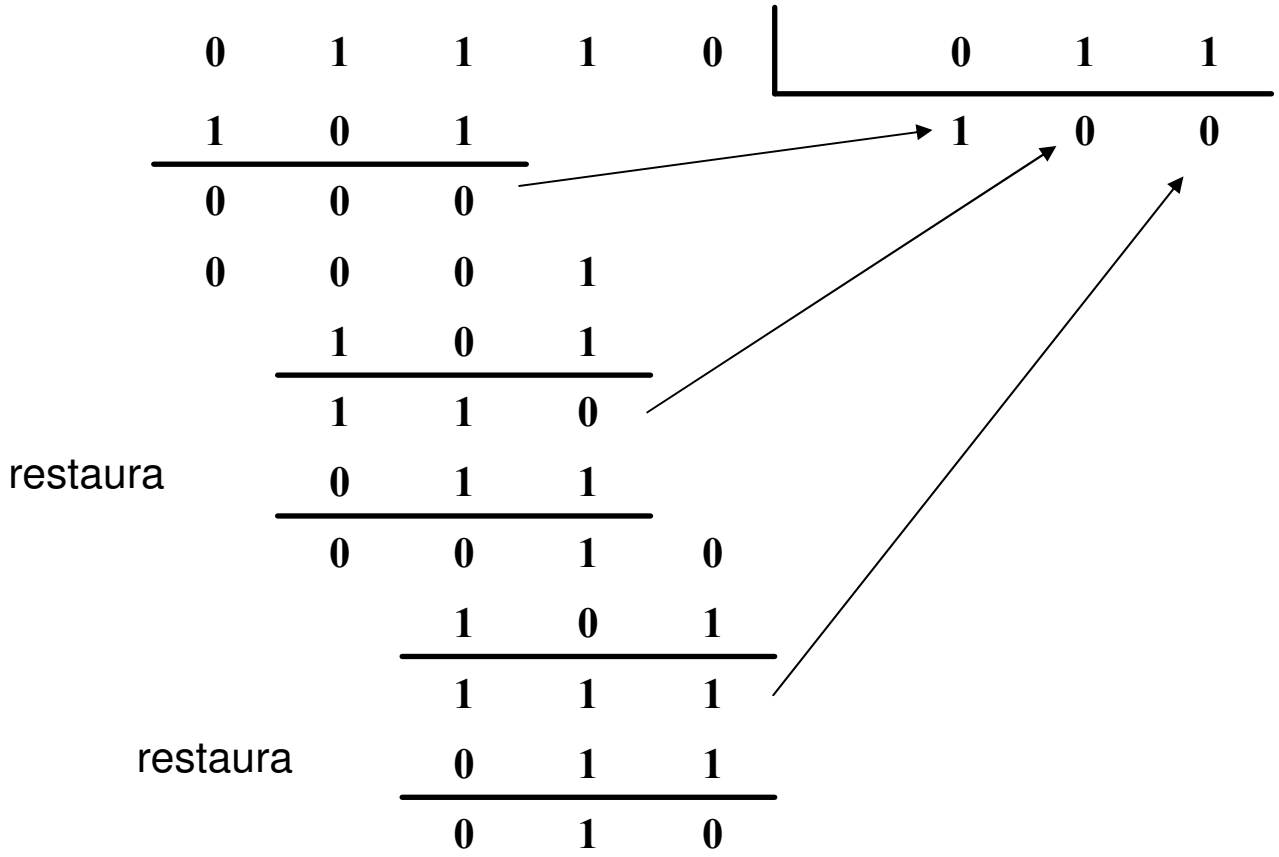
- pero si expresamos 01110 $(14)_{10}$ *como 10000 – 00010* $(16)_{10}$ - $(2)_{10}$
sólo hay 2 productos parciales (uno suma y otro resta)

(Un método formal para implementar esta idea es dado por el algoritmo de Booth)

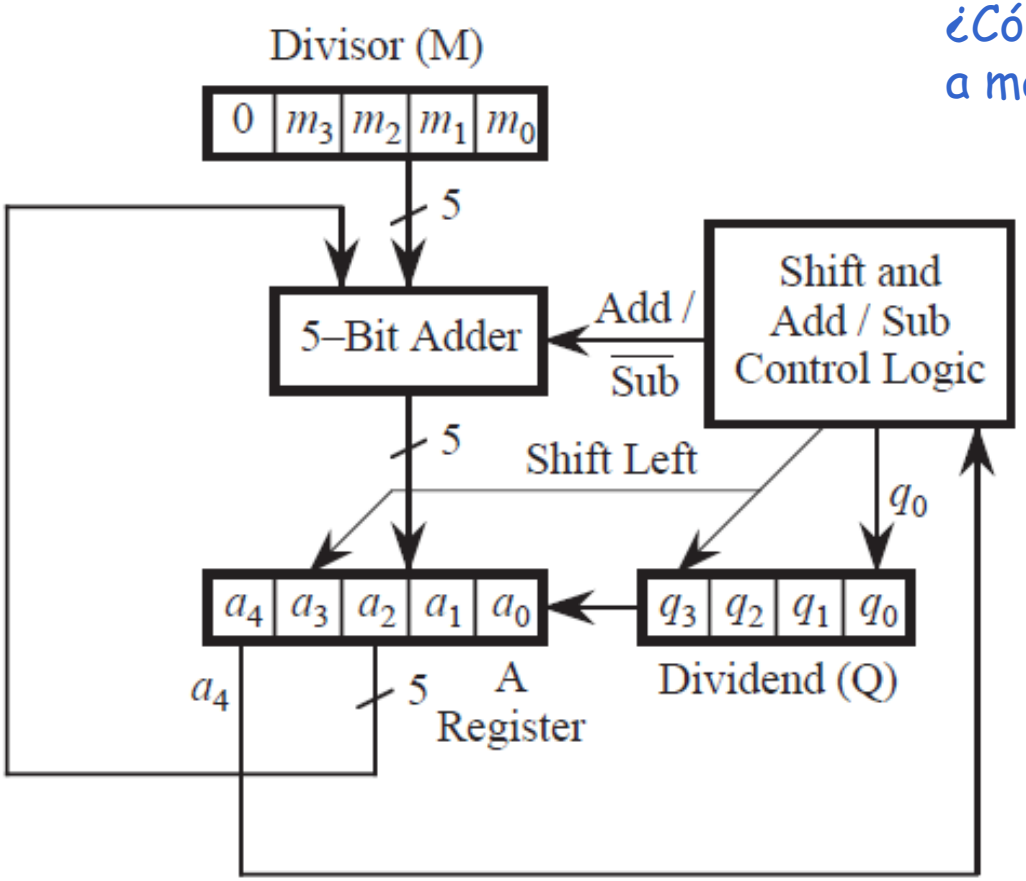
División

- *Se puede implementar en hardware siguiendo el método de papel y lápiz*
- *Se basa en restas y desplazamientos*
- *Para ver cuantos dígitos del dividendo "entran" en el divisor:*
  1. *se hace la resta del dividendo menos el divisor*
  2. *si el signo del resto es negativo se debe bajar un digito más*
- *Paso a paso:*
  1) *Se toma el primer bit del dividendo*
  2) *Se le resta el divisor (se suma su complemento)*
  3) *En caso que el resto sea negativo*
     1. *se agrega un 0 al cociente*
     2. *se restaura el valor original (suma del divisor)*
     3. *se "baja" del dividendo un dígito más y se lo agrega al resultado de la resta*
  4) *En caso de que sea nulo o positivo*
     1. *se agrega un 1 al cociente*
     2. *se "baja" del dividendo un dígito más y se lo agrega al resultado de la resta*
  5) *Vuelve a 2 en tanto queden en el dividendo bits sin procesar*

División


0 1 1 1 0
0 1 1
1 0 1
1 0 0
0 0 0
0 0 0 1
1 0 1
1 1 0
restaura
0 1 1
0 0 1 0
1 0 1
1 1 1
restaura
0 1 1
0 1 0

Implementación de un divisor serie de dos números de 4 bits



¿Cómo cambia el circuito a mayor cantidad de bits?

Divisor en M

Dividendo en Q

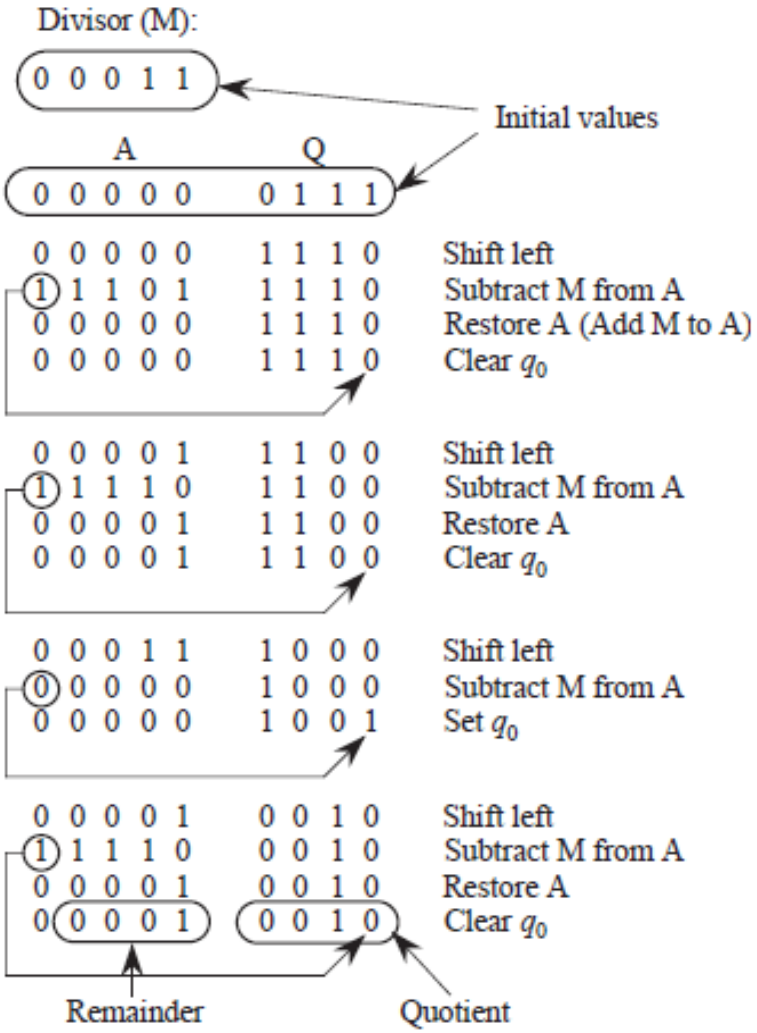
Resultado en Q

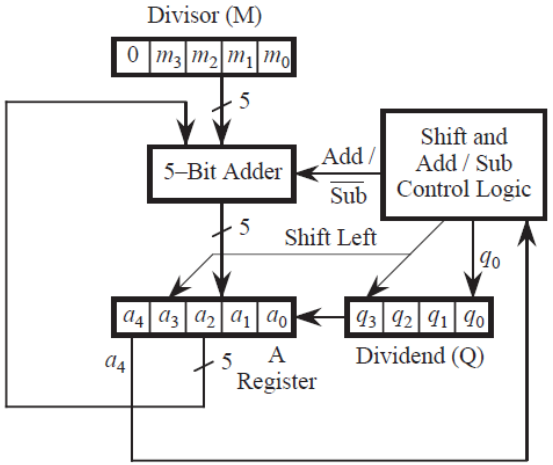
Resto final en A

a_4: signo del resto

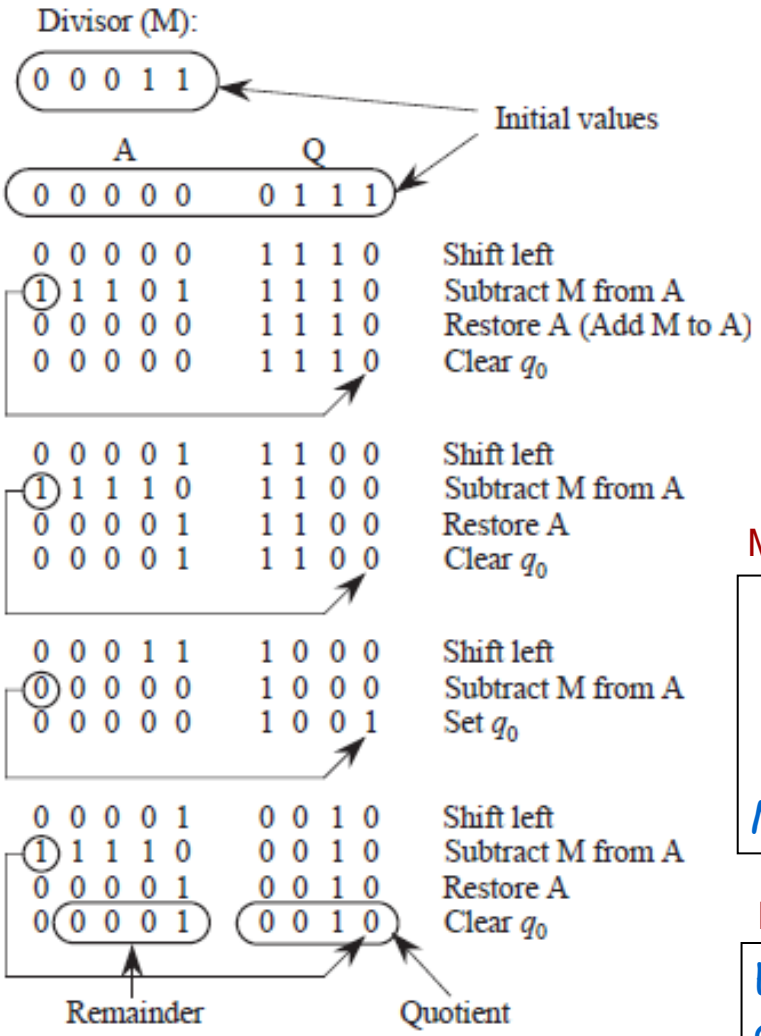
Restos parciales en A

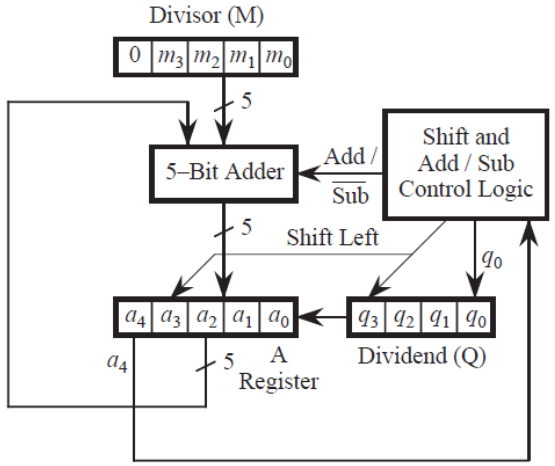
Implementación de un divisor serie de dos números de 4 bits





Implementación de un divisor serie de dos números de 4 bits





Mayor Velocidad **1**

Podemos aumentar la velocidad evitando el "·restaurar"

↓

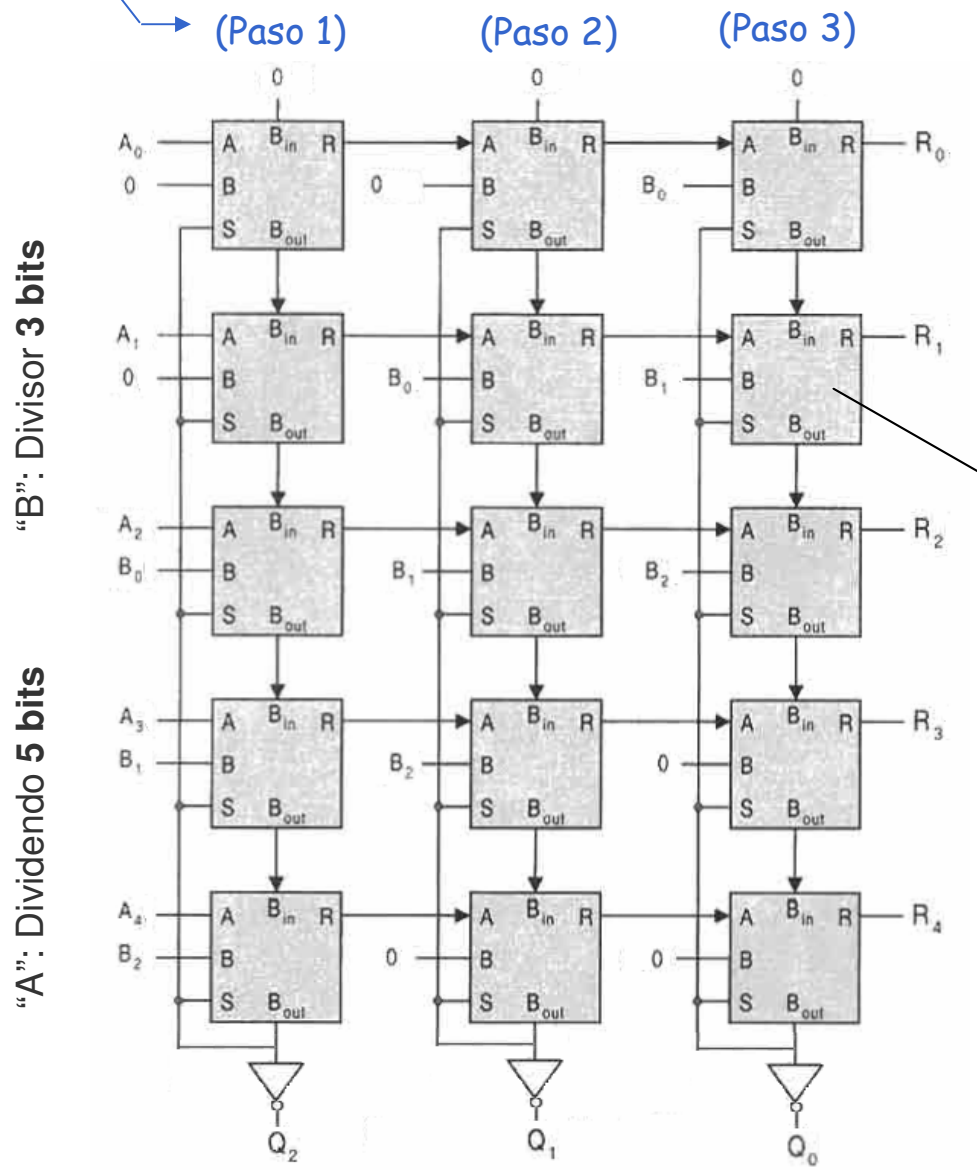
Método de "división sin restauración"

Mayor Velocidad **2**

Estructura paralelo: mayor velocidad al costo de mayor complejidad

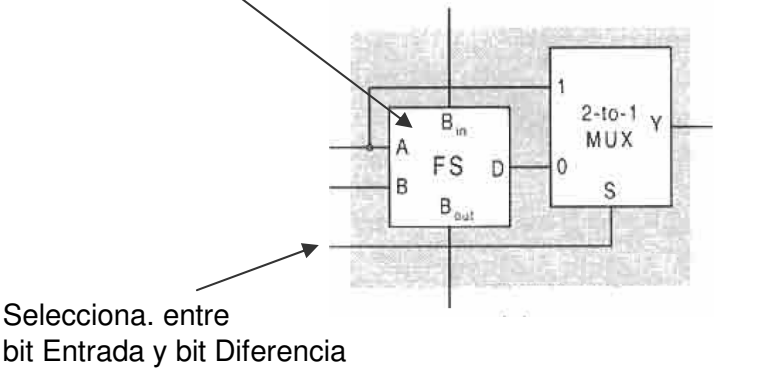
Divisor paralelo

Corresponde a:



En vez de "paso a paso"
=> "nivel" circuital

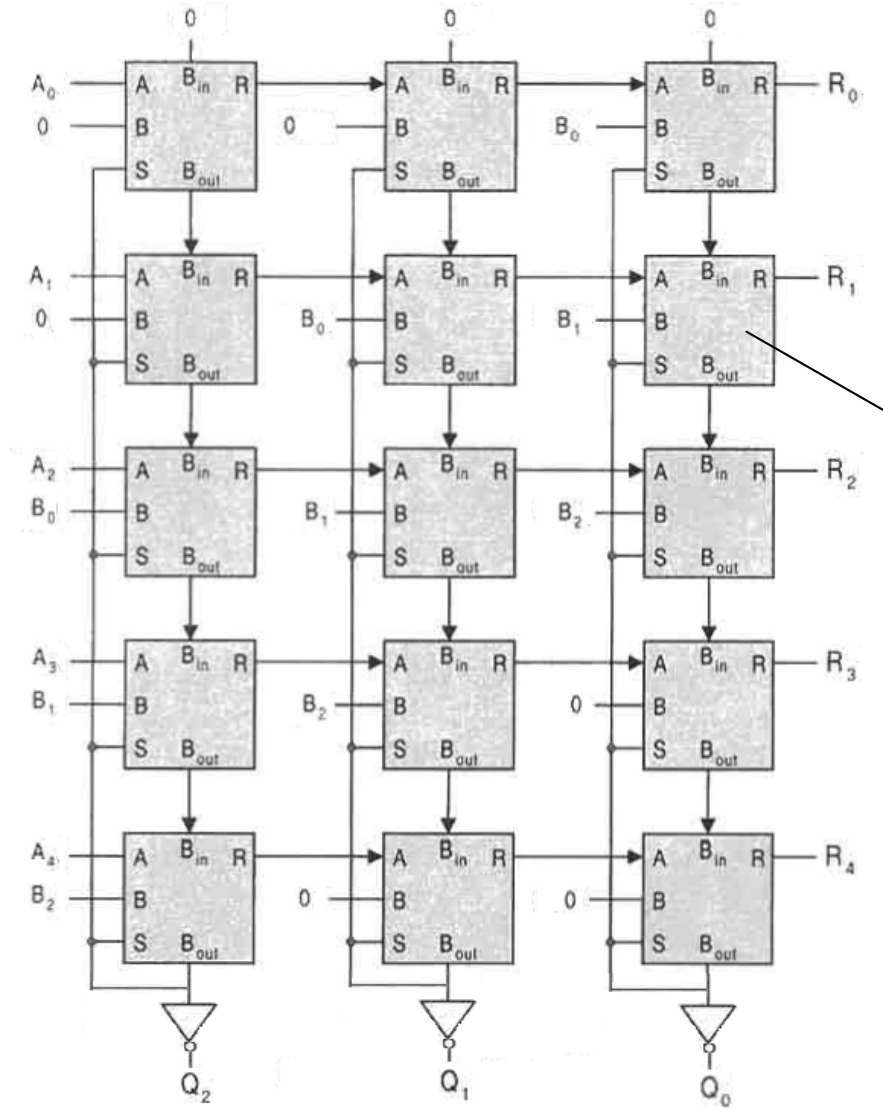
La decisión de restar o bajar otro dígito "la toma" el multiplexor de cada módulo "restador si/no"



Divisor paralelo

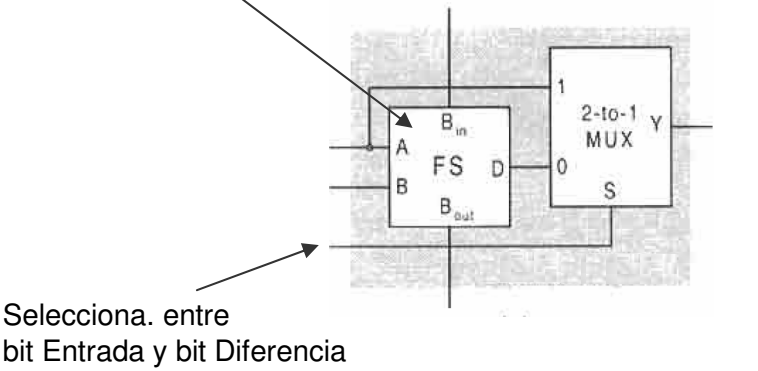
"A": Dividendo **5 bits**

"B": Divisor **3 bits**



¿Porqué debería ser más rápido que el divisor serie?

¿Cómo cambia el circuito a mayor cantidad de bits?



División de enteros con signo

Alternativas:

- División sin signo y signo obtenido según leyes del algebra

- Resolver en complemento a 2

  Resulta en circuitos demasiado lentos/complejos por lo que es raramente utilizada