

# Arquitectura de Computadoras para Ingeniería

(Cód. 7526)  
1° Cuatrimestre 2016

Dra. Dana K. Urribarri  
DCIC - UNS



# Multiplicadores

# Multiplicadores

- Enteros
  - No signados
  - Signados

# Multiplicación de enteros no signados

- Dos números de  $n$  dígitos en base  $b$ 
  - Multiplicador:  $X = x_{n-1} x_{n-2} \dots x_0 = \sum_{i=0}^{n-1} x_i b^i$
  - Multiplicando:  $A = a_{n-1} a_{n-2} \dots a_0$
- El producto  $P$  (de  $2n$  dígitos) será
  - $P = X A = (\sum_{i=0}^{n-1} x_i b^i) A$
- El producto  $P$  (de  $2n$  dígitos) en binario
  - $P = X A = (\sum_{i=0}^{n-1} x_i 2^i) A$
- Se resuelve con  $n$  sumas de operandos de  $n$  bits.  
Por lo tanto, sería de  $O(n^2)$

# Algoritmo secuencial

- Parte de la expresión

$$P = X \cdot A = \left( \sum_{i=0}^{n-1} x_i 2^i \right) A = \sum_{i=0}^{n-1} A x_i 2^i$$

Donde  $x_i = 0$  o  $x_i = 1$

- Arranca con el producto parcial en 0.
- Consiste de  $n$  pasos, donde el paso  $j$  multiplica el bit  $x_j$  por  $A$  y lo suma al producto parcial acumulado.

# Enteros no signados

- El producto de dos enteros no signados de  $n$  bits, puede dar como resultado máximo  $P_{\max}$

$$(2^n - 1)(2^n - 1) = 2^{2n} - \underbrace{2^{2n+1} + 1}_{<0} = 2^{2n-1} + \underbrace{(2^{2n-1} - 2^{n+1} + 1)}_{>0, n \geq 3}$$

- Luego

$$2^{2n-1} < P_{\max} < 2^{2n}$$

- El resultado tiene como máximo  $2n$  bits

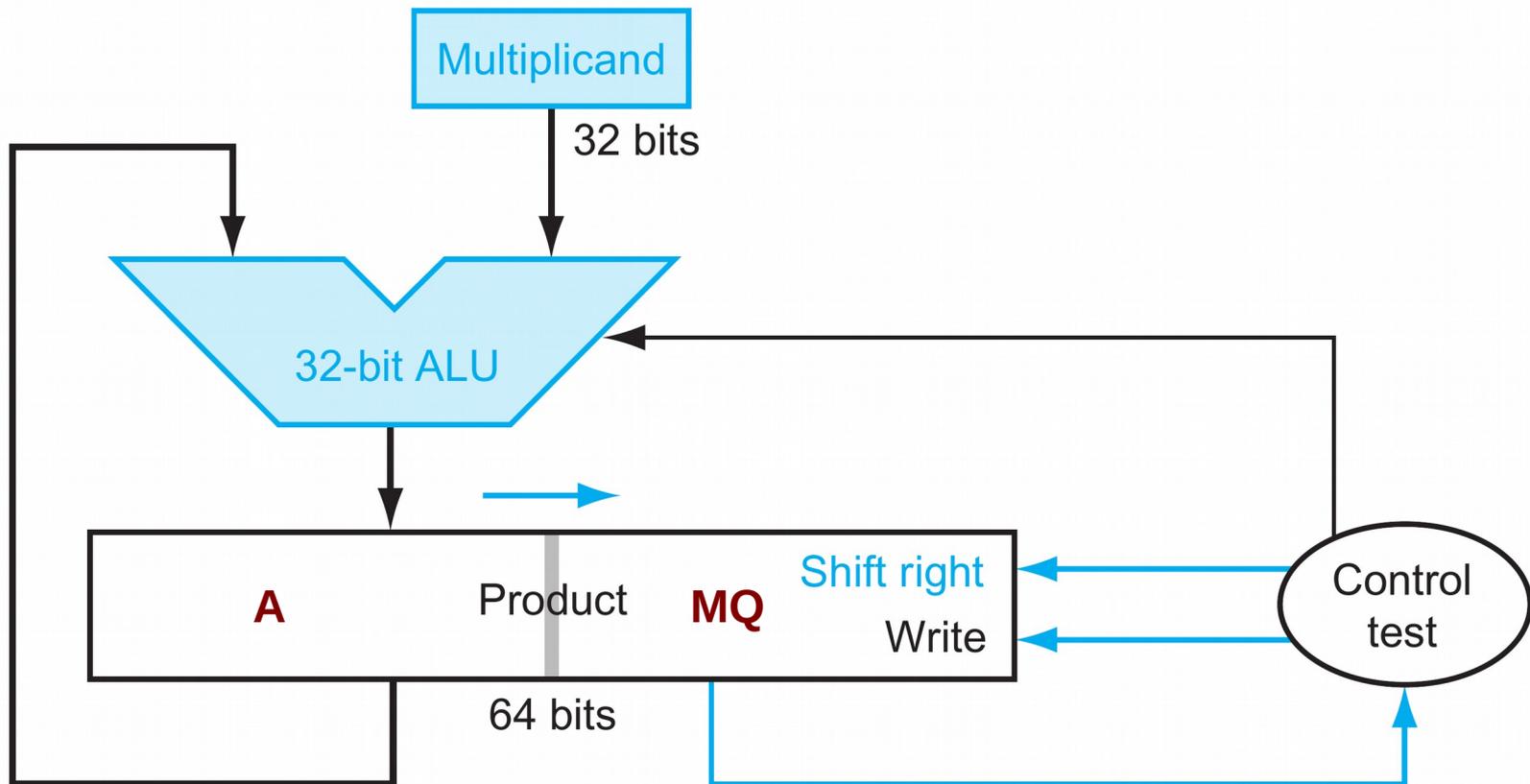
# Algoritmo secuencial

- $X = 13 = 1101$
- $A = 6 = 0110$

					0	1	1	0	
				×	1	1	0	1	
					0	1	1	0	Suma parcial:
					0	0	0	0	00000110
		0			0	1	1	0	00000110
					0	1	1	0	00011110
	+	0	1	1	0				01001110
	0	1	0	0	1	1	1	0	Producto

# Hardware del algoritmo secuencial

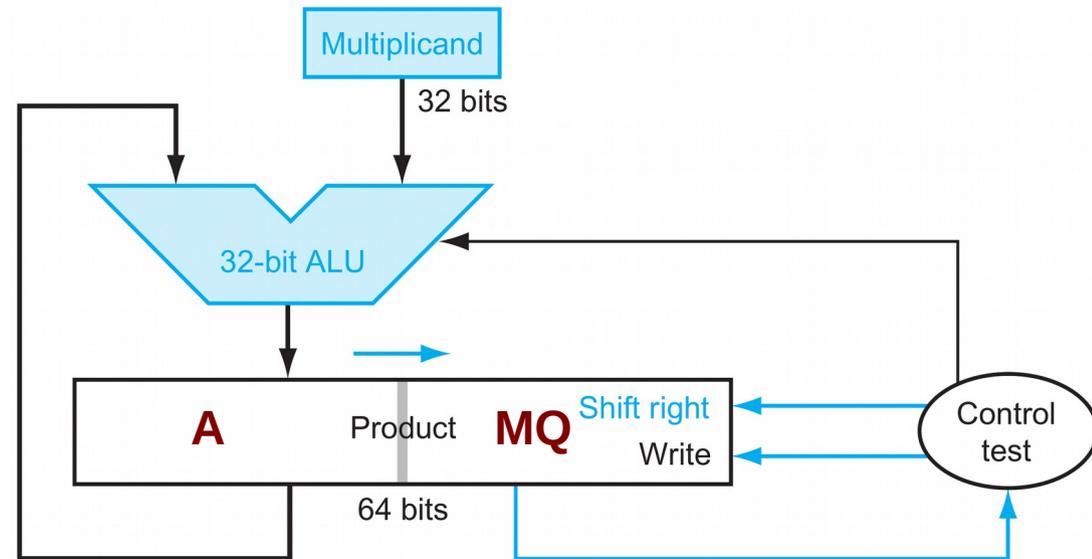
- El registro Producto P es un registro doble A | MQ
  - Donde A y MQ son registros de  $n$  bits
- La ALU suma los registros de  $n$  bits A y Multiplicando



# Hardware del algoritmo secuencial

- Inicio: Copiar el multiplicador en el registro MQ (parte derecha de P). Inicializar A (parte izq.) en 0.
- Repetir n veces

- Paso 1:  
Si  $P_0 = 0$ , ir al paso 3
- Paso 2:  
 $A \leftarrow A + \text{Multiplicando}$
- Paso 3: Desplazar P 1 bit a derecha.

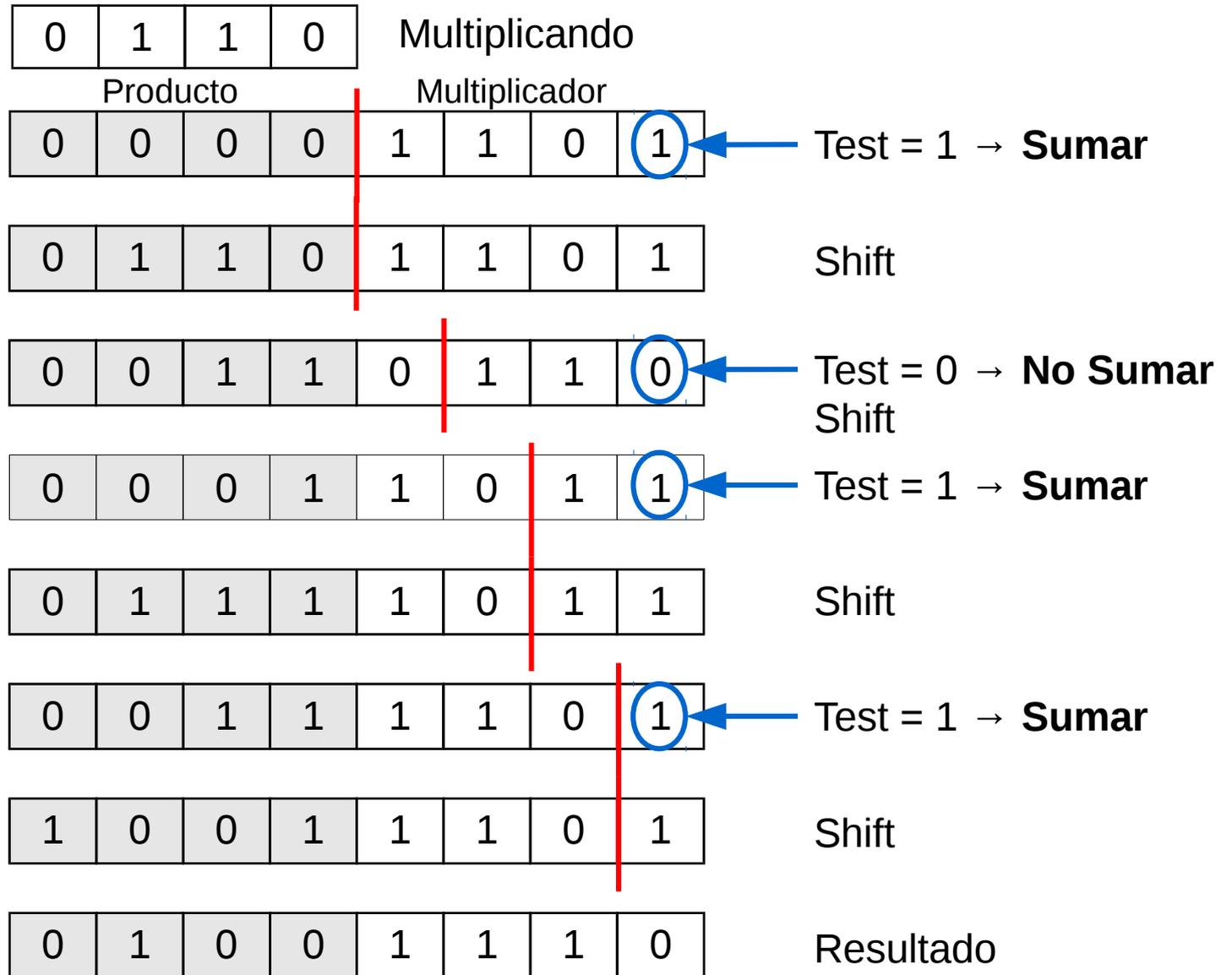


- El registro doble P contiene el resultado

# Ejemplo

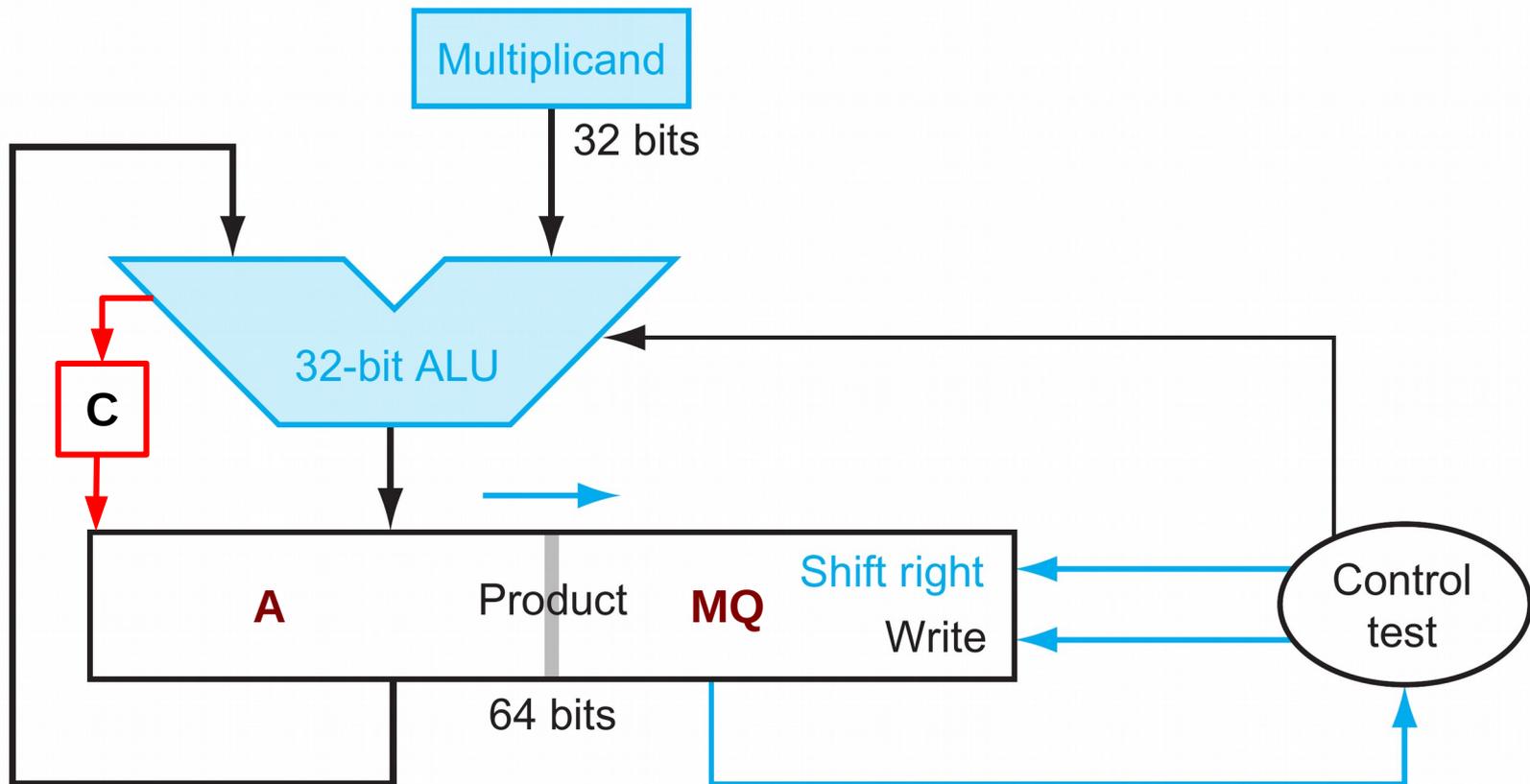
X = 13

A = 6



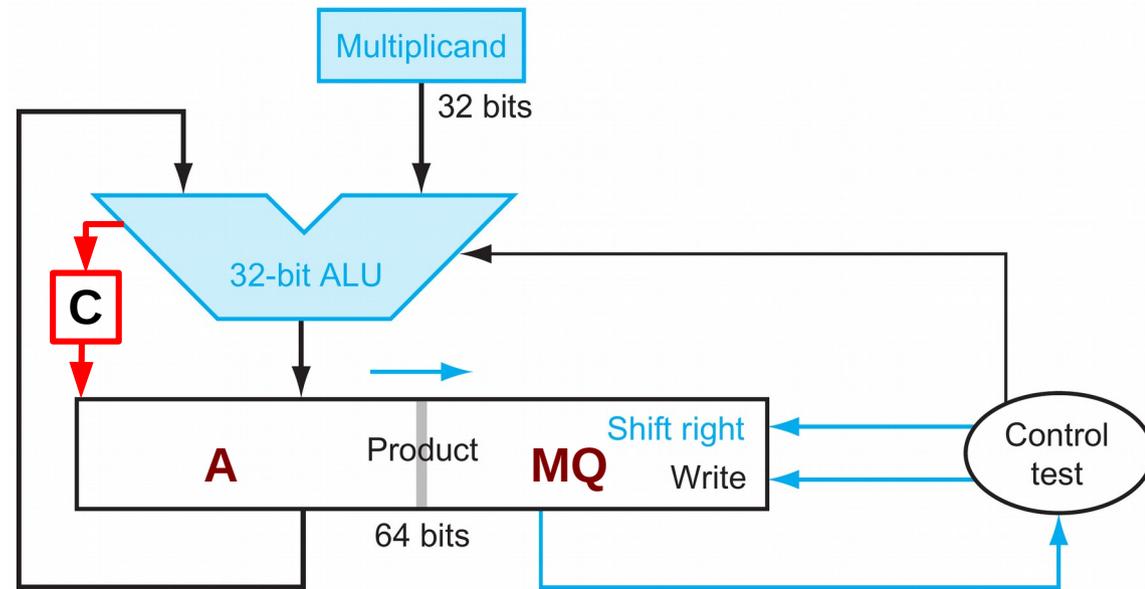
# Hardware del algoritmo secuencial + carry

- Las sumas parciales pueden generar carry-out.
- Agregar el carry en el hw básico



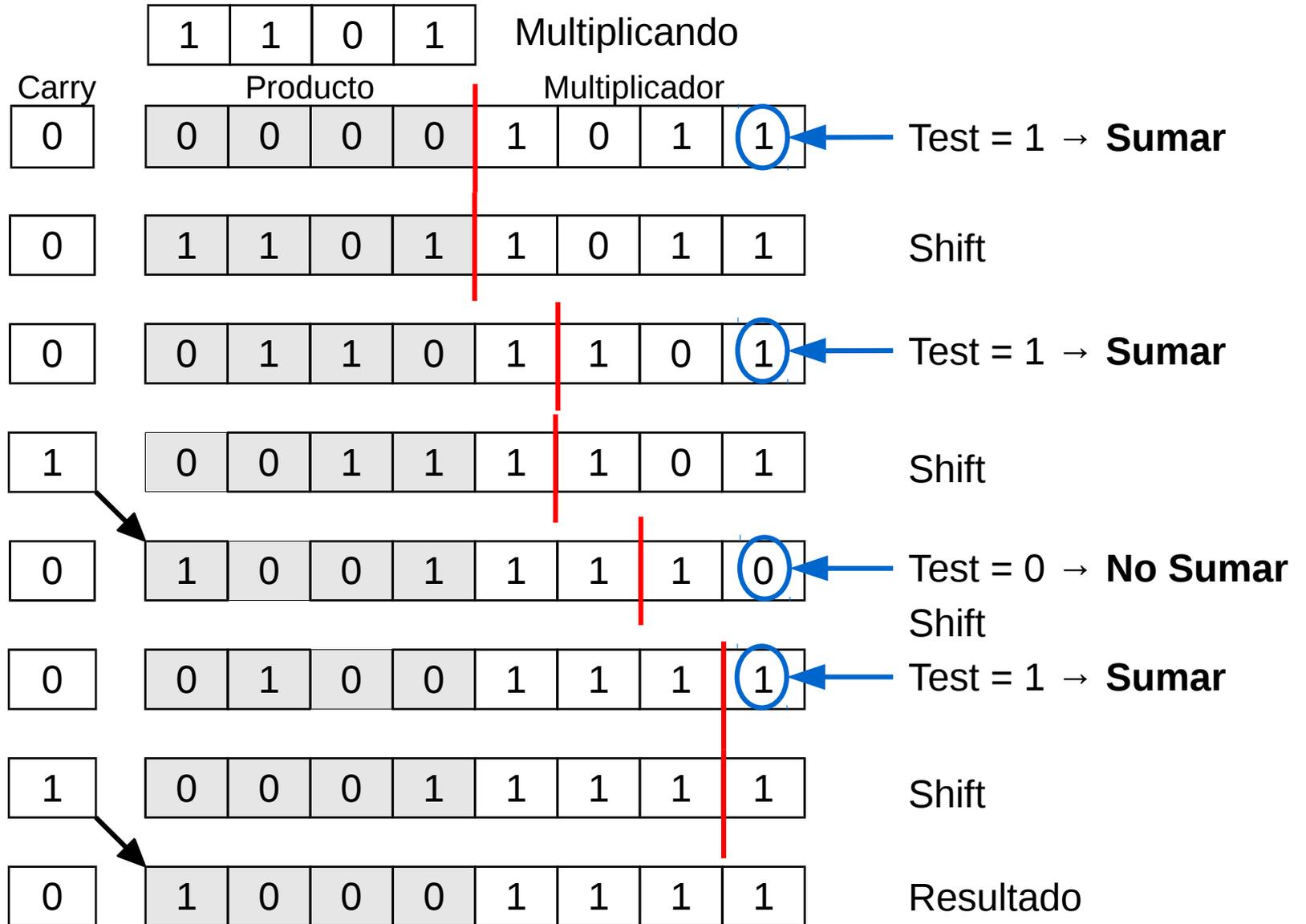
# Hardware del algoritmo secuencial

- Inicio: Copiar el multiplicador en el registro MQ. Inicializar A en 0.
- Repetir n veces
  - Paso 1:  
Si  $P_0 = 0$ , poner el carry en cero e ir al paso 3
  - Paso 2:  
 $A \leftarrow A + \text{Multiplicando}$
  - Paso 3: Desplazar P 1 bit a derecha incluyendo el carry.



# Ejemplo

X = 11  
A = 13



# Multiplicación de enteros signados

- Si los números están en signo-magnitud
  - Calcular el producto sin signo
$$|p| = |X| \cdot |A|$$
  - Calcular el signo de forma separada
$$p_{2n-1} = x_{n-1} \oplus a_{n-1}$$
- En el caso de complemento a 2 y complemento a uno.

Distinguir entre multiplicando A negativo y multiplicador X negativo

# Multiplicación de enteros signados

- Multiplicando negativo
  - $A = 2^n - |A|$
  - $P' = X \cdot A = X \cdot (2^n - |A|) = 2^n X - X \cdot |A|$
  - $P = -X \cdot |A|$

# Multiplicación de enteros signados

- Solución 1:
  - La diferencia entre  $P$  y  $P'$  es  $2^n X \therefore P = P' - 2^n X$
  - Como  $P$  es un registro de  $2n$  bits,  $-2^n X$  en 2 complemento es  $2^{2n} - 2^n X$
  - Corregir el resultado del algoritmo restando  $X$  de la parte más significativa del registro  $P$  (esto no requiere ALU de  $2n$  bits).

# Multiplicación de enteros signados

- Solución 2:

- Considerar A de doble precisión:  $A = 2^{2n} - |A|$

- $P' = X \cdot A = X \cdot (2^{2n} - |A|) = 2^{2n} X - X \cdot |A|$

- $2^{2n} X$  es mayor que la longitud del registro P  $\square$  es carry que se descarta.

- $P' = X \cdot A = 2^{2n} X - X \cdot |A|$   
 $\equiv 2^{2n} - X \cdot |A|$

						1	0	0	1	1	(-13)
					×	0	1	0	1	1	(+11)
<hr/>											
1	1	1	1	1	1	1	0	0	1	1	
1	1	1	1	1	1	0	0	1	1		
0	0	0	0	0	0	0	0	0			
1	1	1	0	0	1	1					
0	0	0	0	0	0						
<hr/>											
1	1	0	1	1	1	0	0	0	1		(-143)

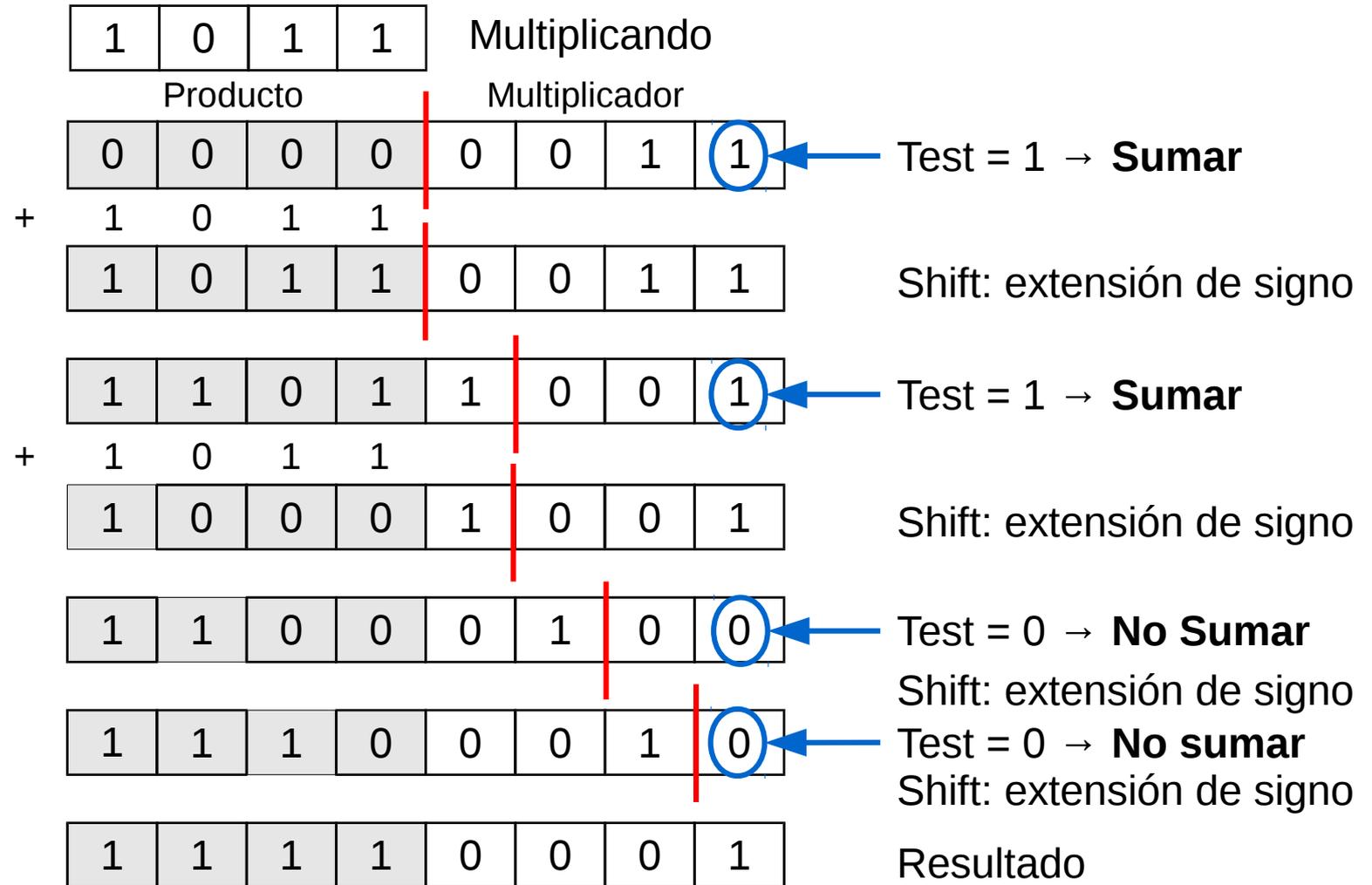
- No implica ALU de  $2n$  bits.

# Multiplicación de enteros signados

- Multiplicando negativo: Solución 2 ✓
  - Si solo el multiplicando es negativo, no hay necesidad de cambiar el algoritmo.
  - Se suma un número negativo.
  - El hardware debe extenderse de forma tal que provea extensión de signo en el producto parcial.
    - Antes de la primera suma, en la extensión de signo ingresa 0.
    - Luego de la primera suma, en la extensión de signo ingresa  $a_{n-1}$

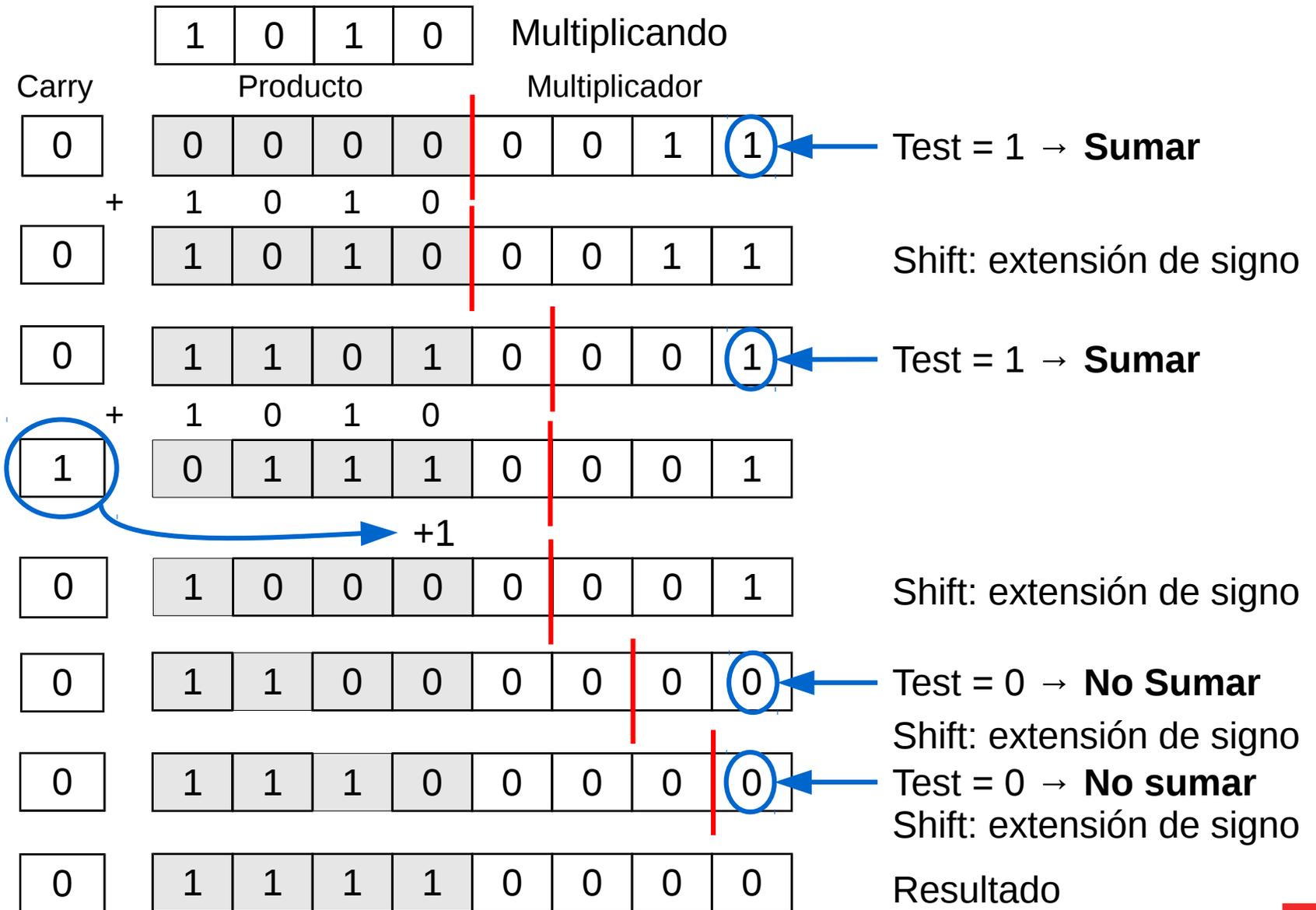
# Ejemplo: complemento a 2

$X = 3$   
 $A = -5$   
 $a_{n-1} = 1$



# Ejemplo: complemento a 1

$X = 3$   
 $A = -5$   
 $a_{n-1} = 1$



# Multiplicación de enteros signados

- Multiplicador negativo
  - $X = 2^n - |X|$
  - $P' = X \cdot A = (2^n - |X|) \cdot A = 2^n A - |X| \cdot A$
  - $P = -|X| \cdot A$

# Multiplicación de enteros signados

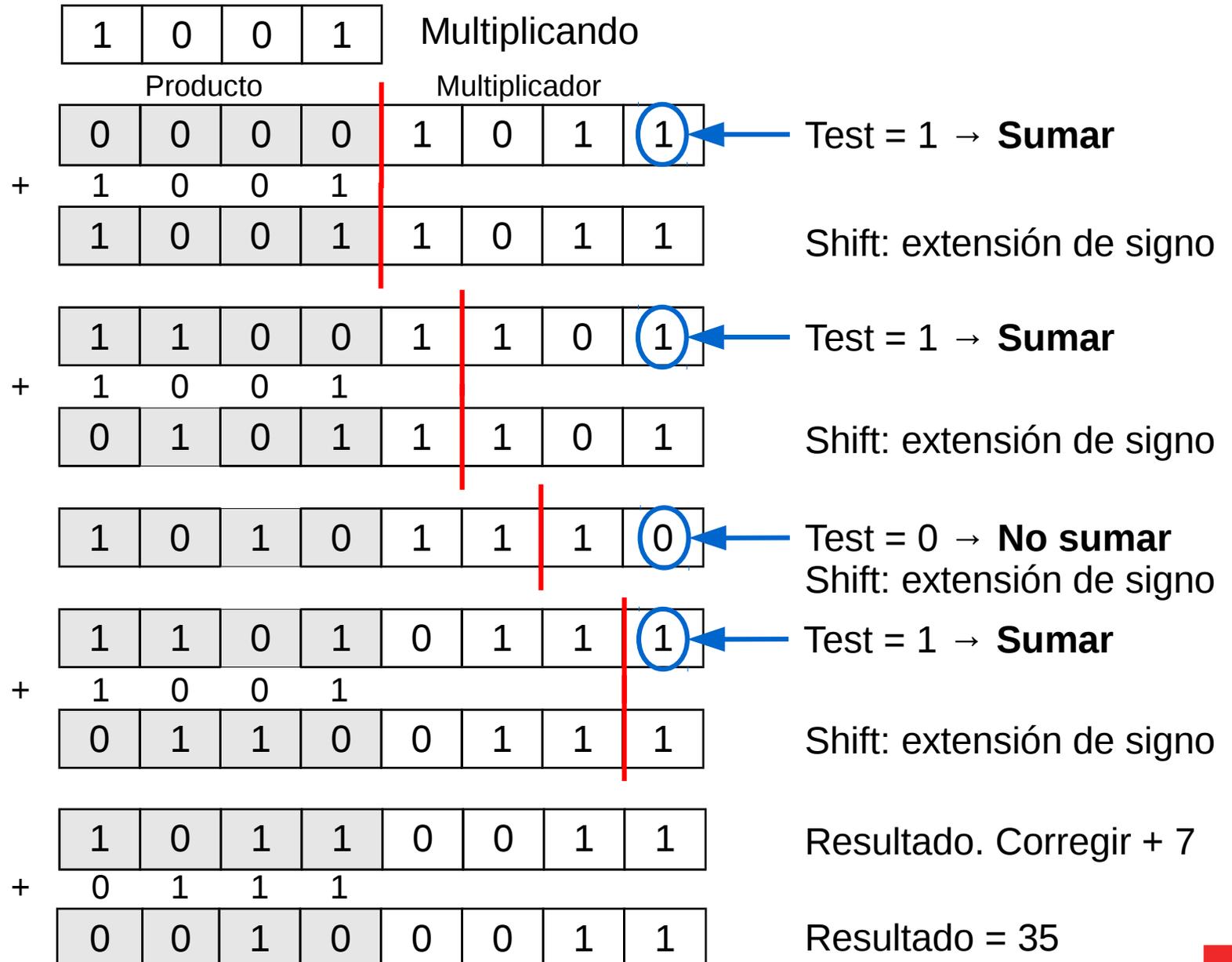
- Solución 1:
  - La diferencia entre  $P$  y  $P'$  es  $2^n A \therefore P = P' - 2^n A$
  - Como  $P$  es un registro de  $2n$  bits,  $-2^n A$  en 2 complemento es  $2^{2n} - 2^n A$
  - Corregir el resultado del algoritmo restando  $A$  de la parte más significativa del registro  $P$  (esto no requiere ALU de  $2n$  bits).

# Multiplicación de enteros signados

- Solución 2:
  - Asumir  $X$  de doble precisión:  $X = 2^{2n} - |X|$
  - $P' = X \cdot A = (2^{2n} - |X|) \cdot A = 2^{2n} A - |X| \cdot A$   
 $\equiv 2^{2n} - |X| \cdot A$
  - ¿Problema?
    - Multiplicador de doble precisión implica el doble de iteraciones.
- Multiplicador Negativo: Solución 1 ✓

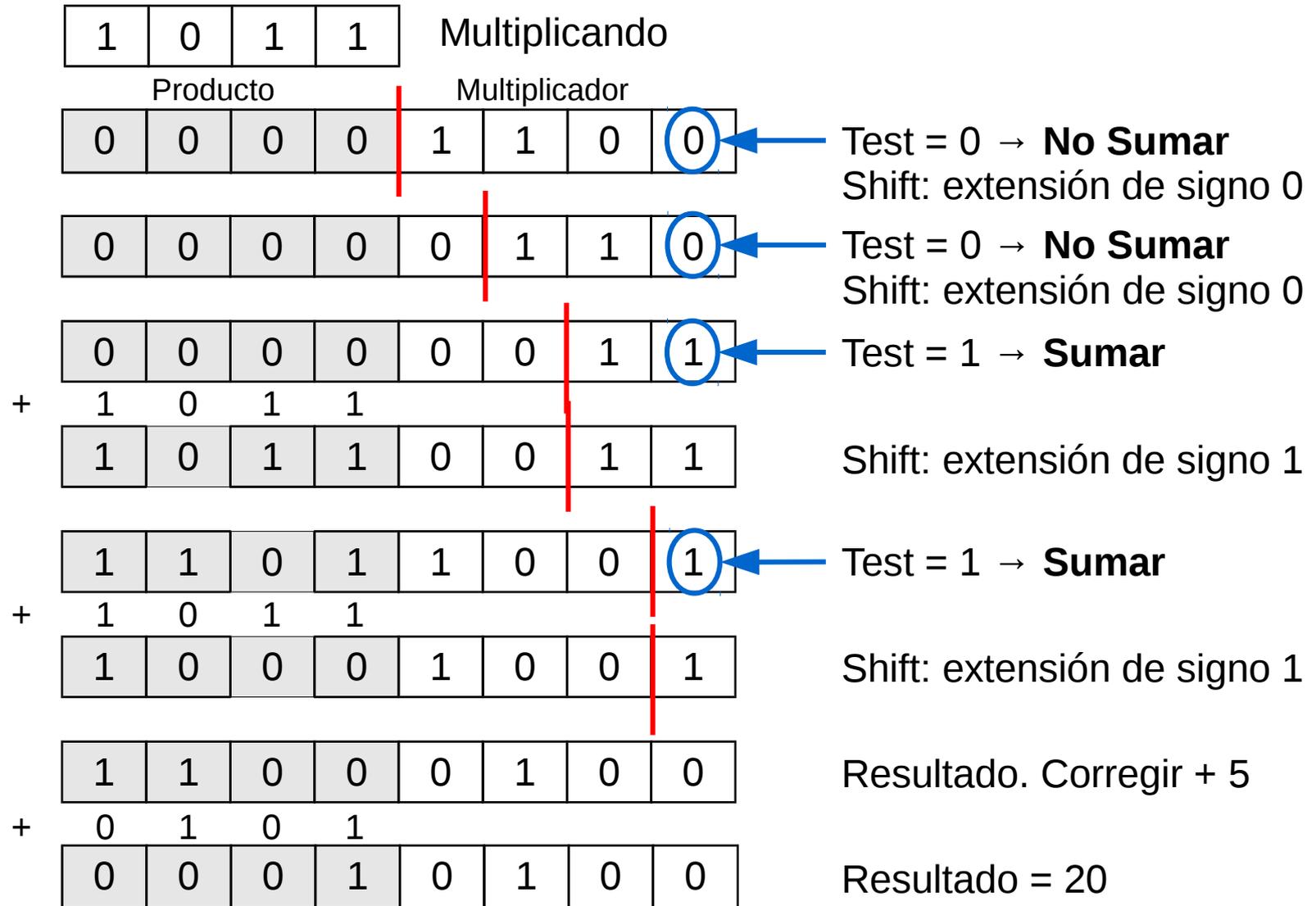
# Ejemplo: complemento a 2

$X = -5$   
 $A = -7$   
 $a_{n-1} = 1$



# Ejemplo: complemento a 2

$X = -4$   
 $A = -5$   
 $a_{n-1} = 1$



# Algoritmo secuencial

- Casos especiales en función del multiplicador o multiplicando negativo.
- Grandes secuencias de 1s en el multiplicador generan sumas sucesivas.

# Reducir el número de productos parciales

- Examinar de a varios bits a la vez.
- Requiere generar múltiplos del multiplicando
- Examinar de a 2 bits:

$$\underbrace{1101}_{\text{2 bits}} \times 0110 = 11 \times 0110 \times 100 + 01 \times 0110$$

- Reduce en 2 las sumas parciales
- Implica generar  $0A, A, 2A, 3A, \dots$
- Examinar de a  $c$  bits reduce en  $c$  los productos parciales.
  - Implica generar  $0A, 1A, 2A, \dots, (2^c - 1)A$

# Algoritmo de Booth

- Reduce la cantidad de productos parciales.
- Se basa en que

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

- Genera secuencias de 1s como la resta entre dos operandos de un solo 1 cada uno.

$$\begin{array}{r} 1000 \quad (8) \\ - 0001 \quad (1) \\ \hline 0111 \quad (7) \end{array}$$

$$\begin{array}{r} 01000000 \quad (64) \\ - 00001000 \quad (8) \\ \hline 00111000 \quad (56) \end{array}$$

# Algoritmo de Booth

$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
0	0	1	1	1	0	0	0
0	+1	0	0	-1	0	0	0

$$56 = 2^5 + 2^4 + 2^3$$

$$56 = 2^6 - 2^3$$

$$56 = 0100\bar{1}000$$

# Recodificación de Booth

- Recorrer el multiplicador  $X$  de derecha a izquierda ( $i = 0, 1 \dots n - 1$ )
- Para cada par de bits consecutivos  $x_i x_{i-1}$  generar el multiplicando recodificado  $Y$ , de tal forma que

$x_i$	$x_{i-1}$	Operation	Comments	$y_i$
0	0	shift only	string of zeros	0
1	1	shift only	string of ones	0
1	0	subtract and shift	beginning of a string of ones	$\bar{1}$
0	1	add and shift	end of a string of ones	1

Asumir  $x_{-1} = 0$

# Recodificación de Booth

- Para multiplicar  $X \cdot |A|$
- Recodificar  $X$  y obtener  $Y$
- Multiplicar  $Y \cdot |A|$
  
- Los productos parciales se calculan:
  - Sumando multiplicando por cada dígito  $+1$
  - Sumando el complemento a 2 del multiplicando por cada dígito  $-1$

# Ejemplo: complemento a 2

$X = -5$

$A = -7$

$X = 1011$

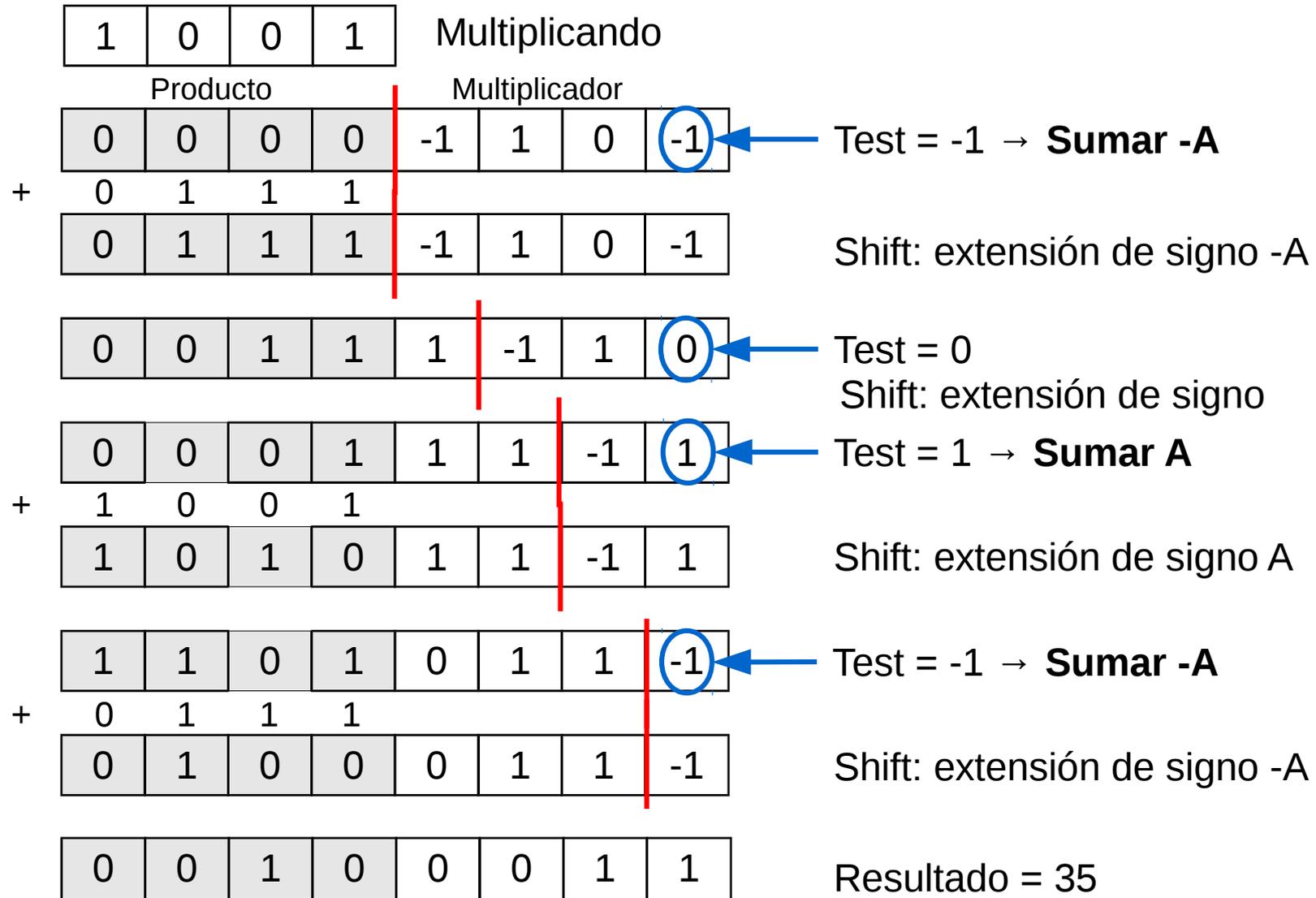
$Y = \overline{1101}$

$A = 1001$

$a_{n-1} = 1$

$\overline{A} = 0111$

$\overline{a}_{n-1} = 0$



# Ejemplo: complemento a 2

$X = -5$

$A = 7$

$X = 1011$

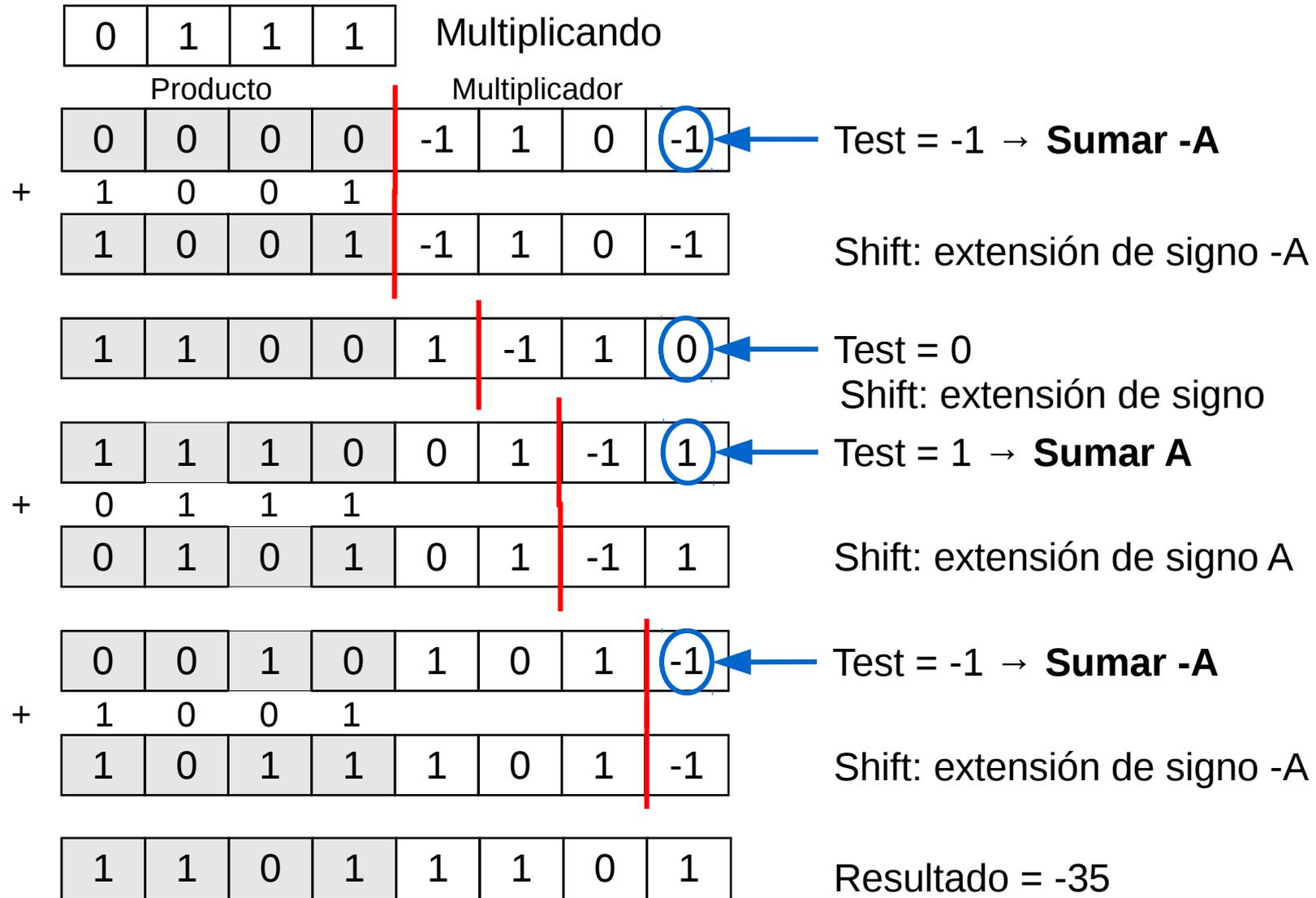
$Y = \overline{1101}$

$A = 0111$

$a_{n-1} = 0$

$\overline{A} = 1001$

$\overline{a}_{n-1} = 1$



# Recodificación de Booth

- El algoritmo secuencial con recodificación de Booth:
  - Funciona correctamente con números en complemento a 2.
  - En el caso de números signados, se debe agregar un cero a la izquierda del multiplicador ( $x_n = 0$ )

# Algoritmo secuencial + Rec. de Booth

## Desventajas

- Complican el diseño de un multiplicador sincrónico:
  - La cantidad de sumas/restas es variable.
  - La cantidad de desplazamientos entre sumas/restas es variable.
- Se vuelve ineficiente con unos aislados.
  - 0010101 se recodifica como  $01\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$ . Requiere 3 sumas y tres restas en vez de solo tres sumas.

# Recodificación de Booth

- Recorrer el multiplicador  $X$  de derecha a izquierda ( $i = 0, 1 \dots n - 1$ )
- Para cada par de bits consecutivos  $x_i x_{i-1}$  generar el multiplicando recodificado  $Y$ , de tal forma que

$x_i$	$x_{i-1}$	Operation	Comments	$y_i$
0	0	shift only	string of zeros	0
1	1	shift only	string of ones	0
1	0	subtract and shift	beginning of a string of ones	$\bar{1}$
0	1	add and shift	end of a string of ones	1

Asumir  $x_{-1} = 0$

# Recodificación de Booth de varios bits

- Combinar la recodificación de Booth con la recodificación de más de a un bit.
- Existen dos formas de recodificación
  - Mirando al futuro
  - Mirando al pasado
- En cualquiera de los dos casos hay que mirar  $c + 1$  bits.

# Recodificación de Booth de a 2 bits

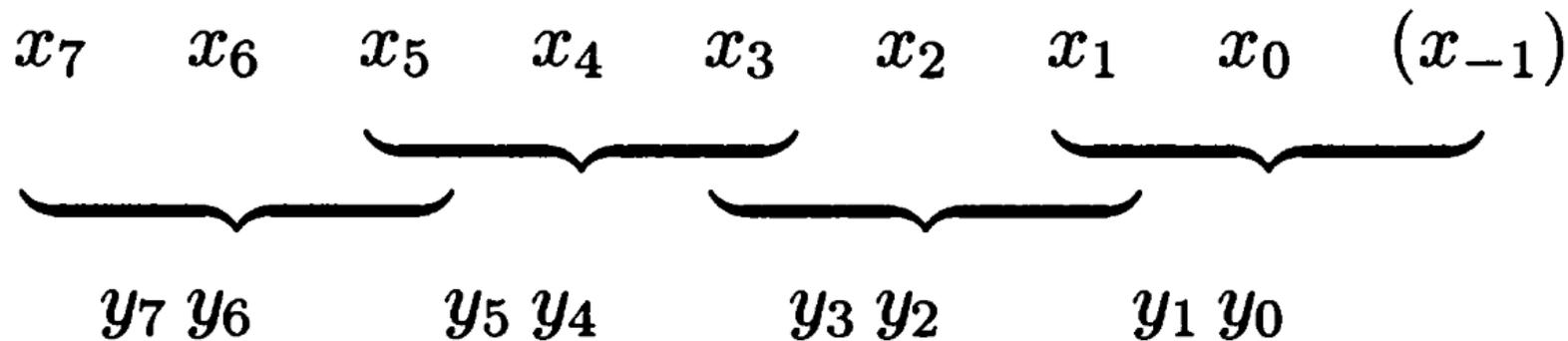
- Mirando al pasado:

- Se dividen en grupos de 2

$$x_7x_6 \mid x_5x_4 \mid x_3x_2 \mid x_1x_0$$

- Y el anterior se usa de referencia.

- Para codificar  $x_{i+1}$  y  $x_i$ , se usa  $x_{i-1}$  como referencia  
( $i = 0, 2, 4, \dots$ )



# Recodificación de Booth de a 2 bits

	$x_{i+1}$	$x_i$	$x_{i-1}$	$y_{i+1}y_i$	Múltiplo
Entre dos cadenas	0	0	0	00	0A
Final de cadena de 1s	0	0	1	01	A
1 aislado	0	1	0	01	A
Final de cadena de 1s	0	1	1	10	2A
Principio de cadena de 1s	1	0	0	$\bar{1}0$	- 2A
0 aislado	1	0	1	$0\bar{1}$	- A
Principio de cadena de 1s	1	1	0	$0\bar{1}$	- A
Medio de una cadena	1	1	1	00	0A

# Recodificación de Booth de a 2 bits

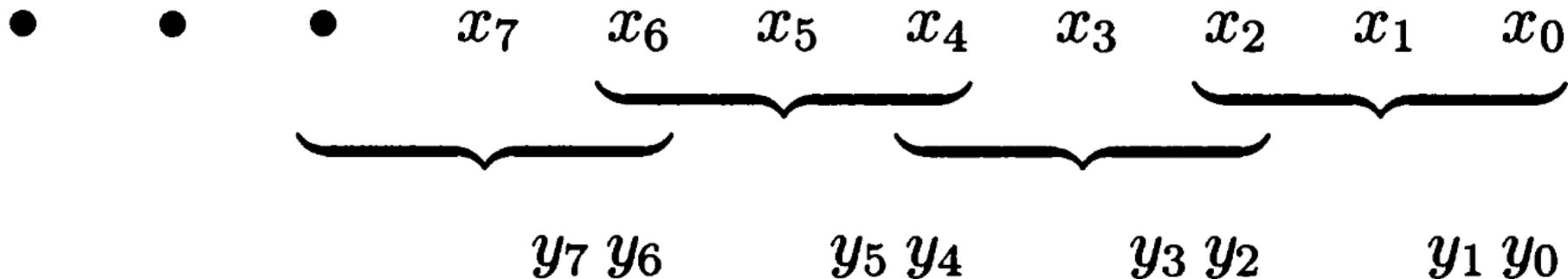
- Mirando al futuro:

- Se dividen en grupos de 2

$$x_7x_6 \mid x_5x_4 \mid x_3x_2 \mid x_1x_0$$

- Y el siguiente bit se usa de referencia.

- Para codificar  $x_{i+1}$  y  $x_i$ , se usa  $x_{i+1}$  como referencia  
( $i = 0, 2, 4, \dots$ )



# Recodificación de Booth de a 2 bits

	$x_{i+2}$	$x_{i+1}$	$x_i$		Múltiplo
Entre dos cadenas	0	0	0	00	0A
Final de cadena de 1s	0	0	1	10	2A
1 aislado	0	1	0	10	$4A - 2A = 2A$
Final de cadena de 1s	0	1	1	100	4A
Principio de cadena de 1s	1	0	0	$\bar{1}00$	-4A
0 aislado	1	0	1	$\bar{1}0$	$-4A + 2A = -2A$
Principio de cadena de 1s	1	1	0	$\bar{1}0$	-2A
Medio de una cadena	1	1	1	00	0A

No podemos generar una recodificación.  
 Sí se generan los múltiplos de A que hay que sumar.

# Recodificación de Booth de varios bits

En la recodificación de a 2 bits se necesitan los múltiplos A:

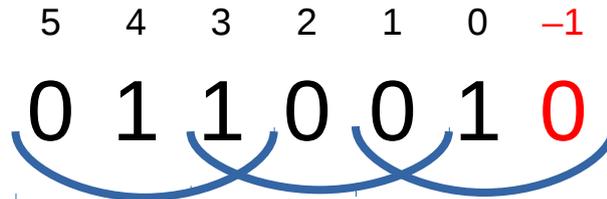
- Sin recodificación de Booth
  - $0A$ ,  $A$ ,  $2A$  y  $3A$
- Con recodificación de Booth mirando al pasado
  - $0A$ ,  $A$  y  $2A$
- Con recodificación de Booth mirando al futuro
  - $0A$ ,  $2A$  y  $4A$

# Ejemplo

$$X = 25 = 011001$$

Recodificación mirando al pasado.

- Se necesitan considerar 1 bit antes del LSB.



$$\begin{aligned} & A \cdot 2^0 & = A \\ -2A \cdot 2^2 & = -8A \\ 2A \cdot 2^4 & = 32A \end{aligned}$$

$$X \cdot A = A - 8A + 32A$$

# Ejemplo

$$X = 25 = 011001$$

Recodificación mirando al futuro.

- Se necesitan considerar 2 bits antes del LSB y un bit después de MSB.

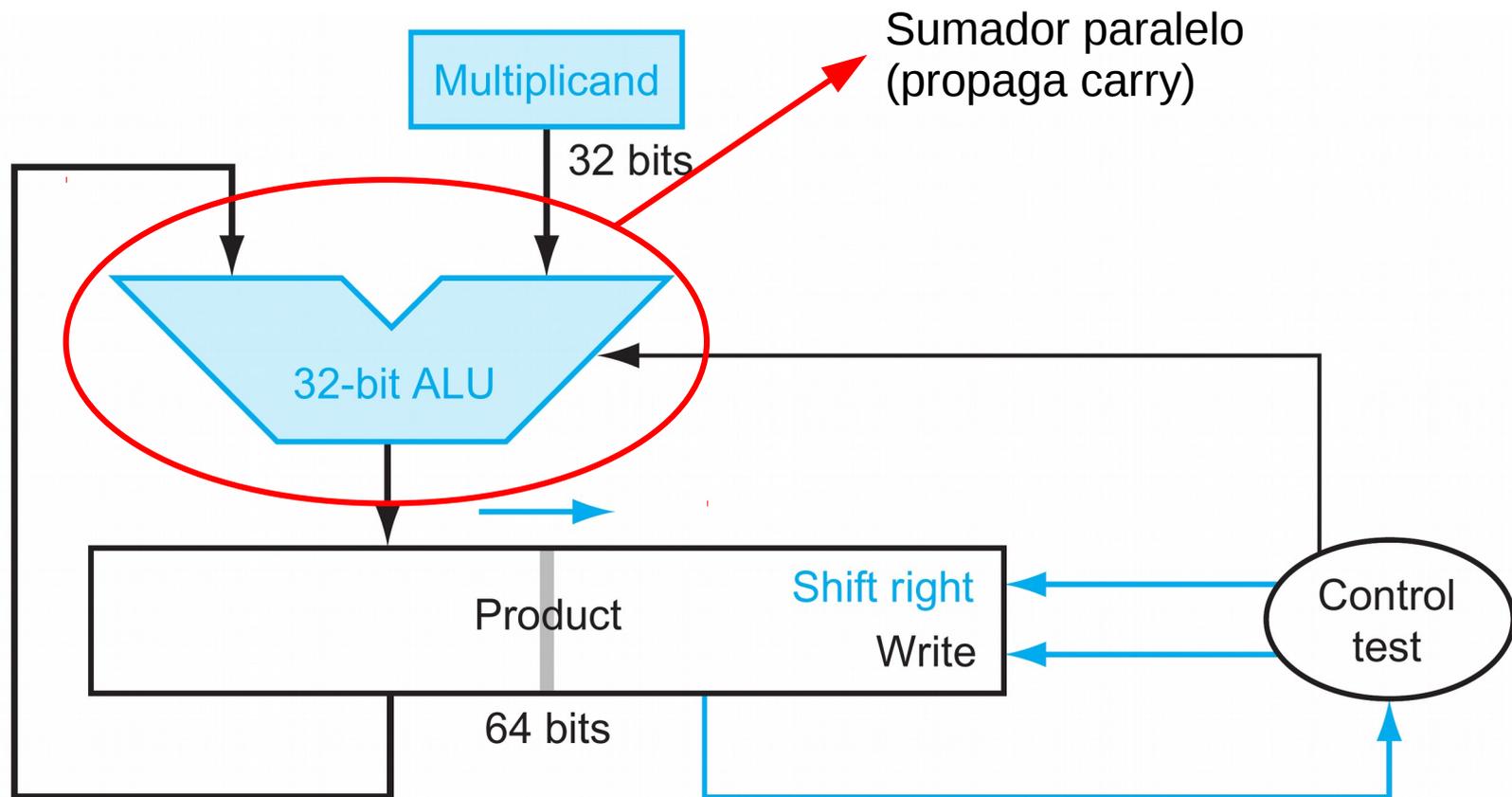
$$\begin{array}{cccccccccc} 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}$$
$$\begin{array}{r} 2A \cdot 2^4 \\ -2A \cdot 2^2 \\ 2A \cdot 2^0 \\ -4A \cdot 2^{-2} \end{array} \quad \begin{array}{l} = 32A \\ = -8A \\ = 2A \\ = -A \end{array}$$

$$X \cdot A = -A + 2A - 8A + 32A$$

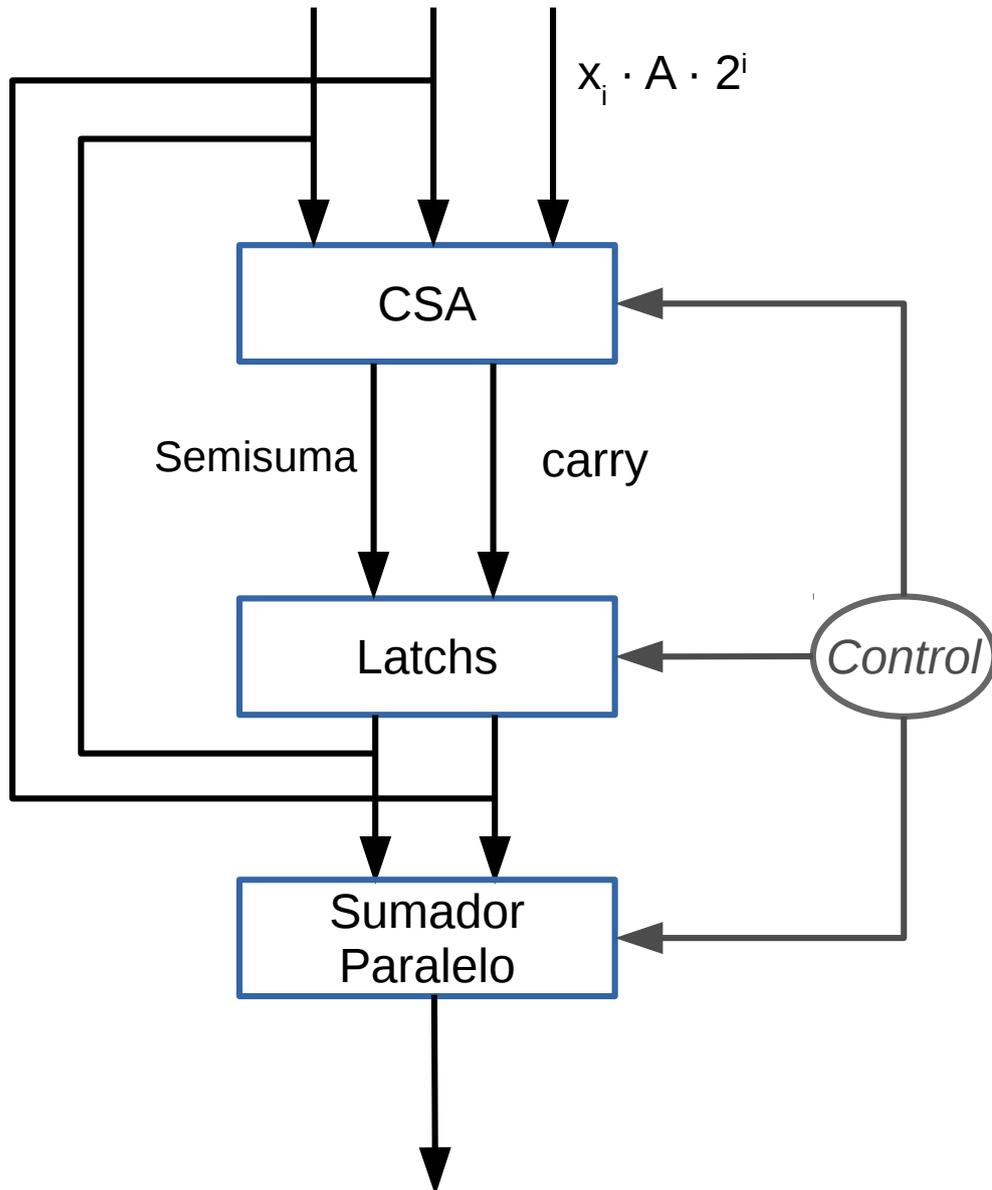


# Hardware de multiplicación

# Hardware algoritmo secuencial

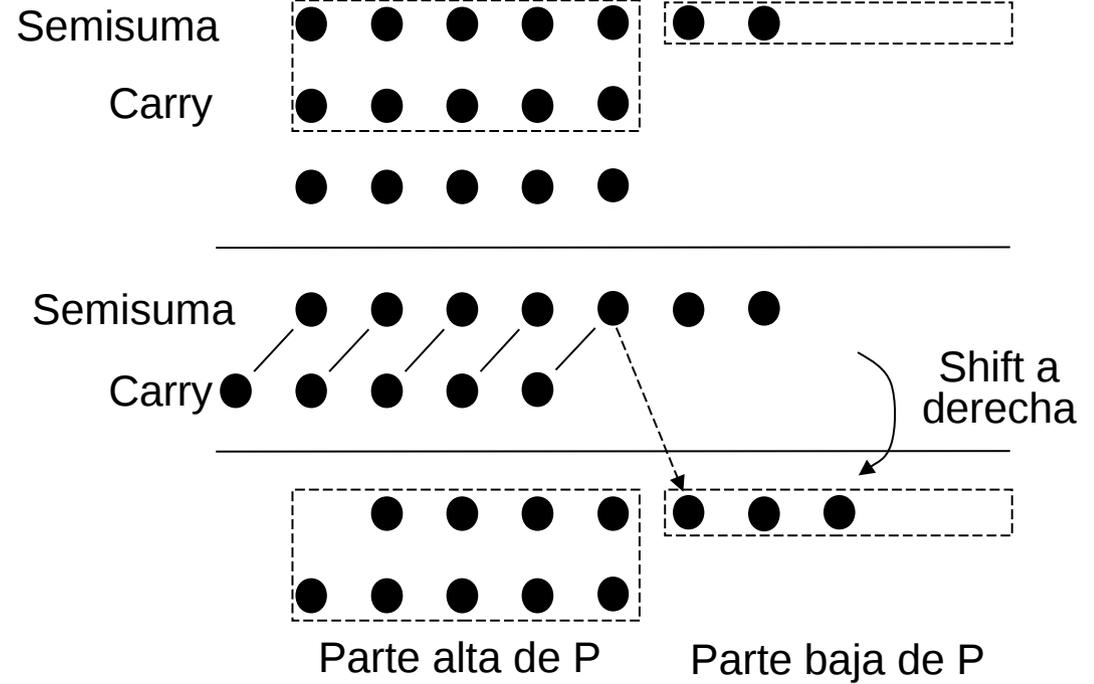
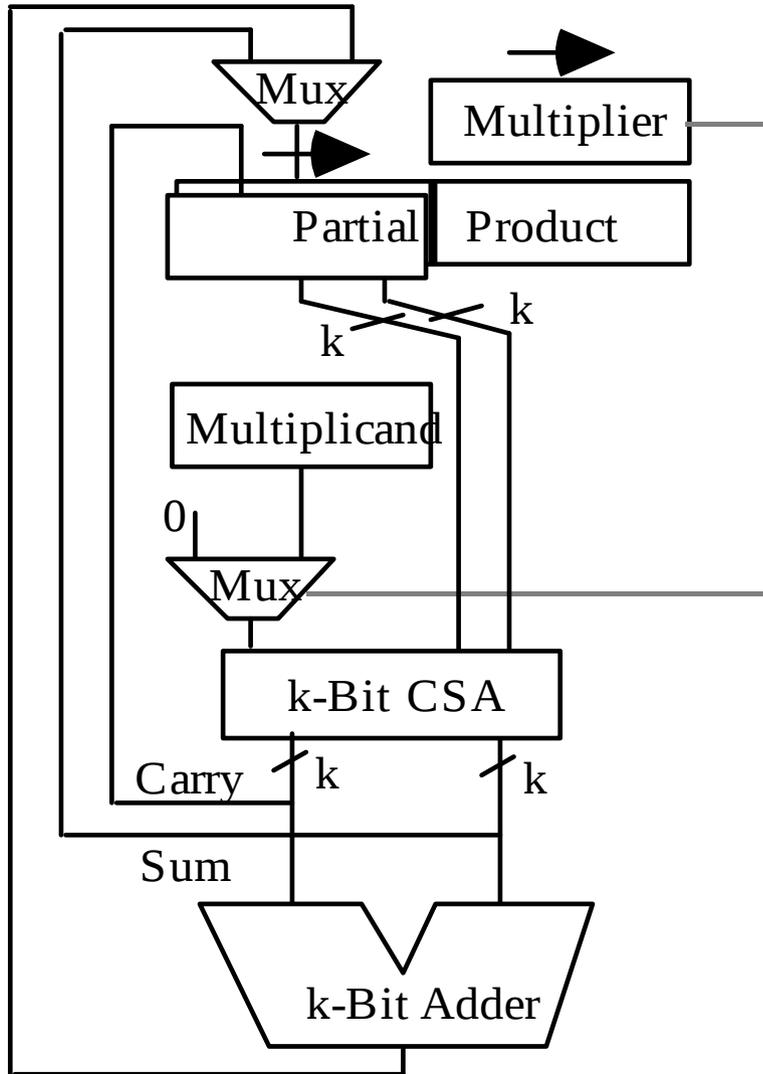


# Hardware usando un CSA

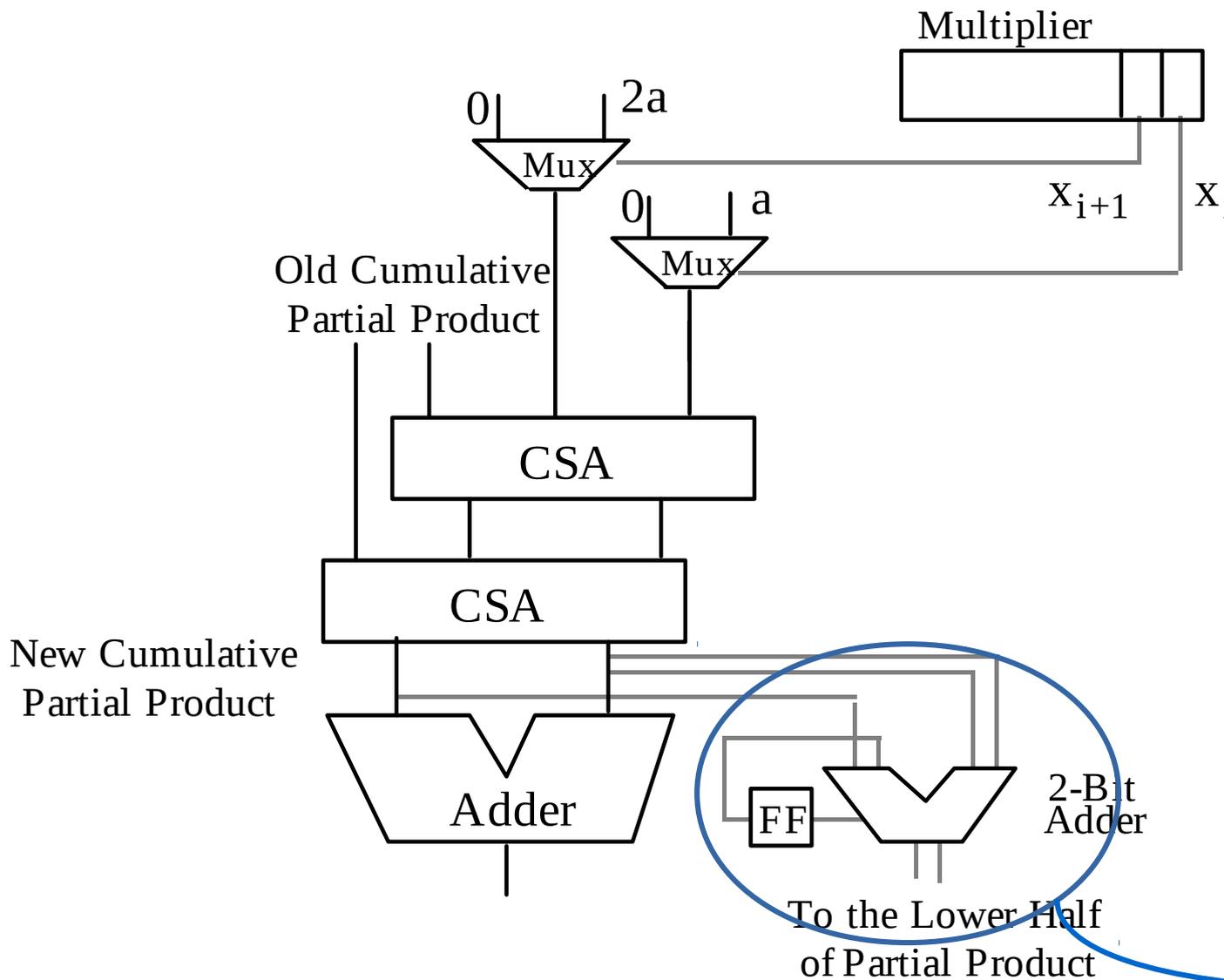


- Se realizan las sumas intermedias hasta obtener una única semisuma y un único carry.
- En el último paso se realiza la suma propagando carry.
- Realiza  $n$  iteraciones para multiplicar operandos de  $n$  bits.

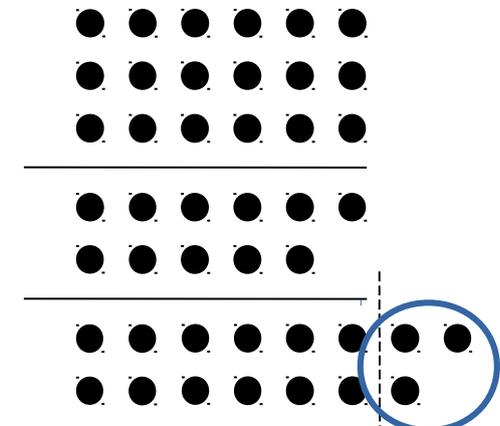
# Hardware usando un CSA



# Niveles de CSA



Combina la semisuma y el carry del CSA con los múltiplos  $x_i A$  y  $2x_{i+1} A$

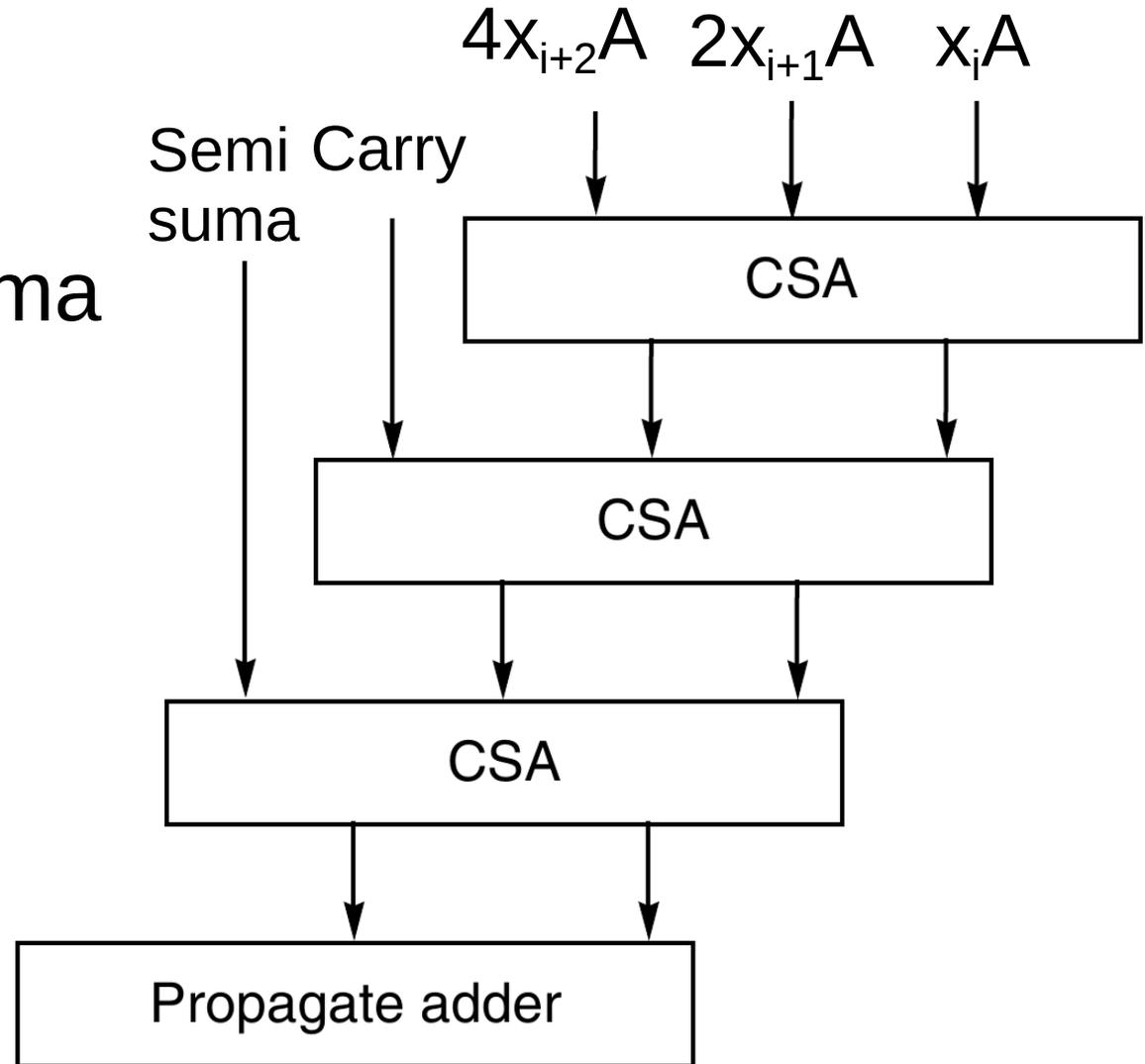


Shift de dos bits a derecha

# Niveles de CSA

- Con tres niveles de CSA se puede combinar la semisuma y el carry con tres múltiplos del multiplicando:

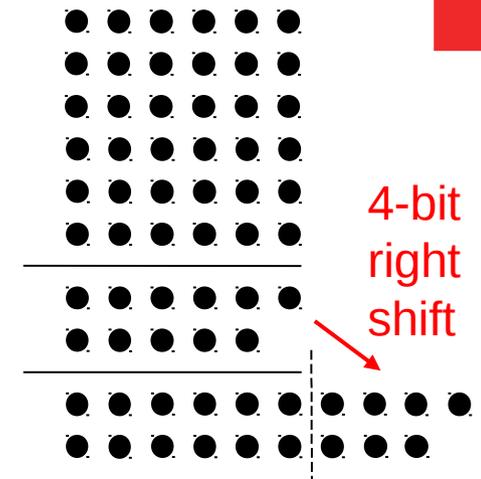
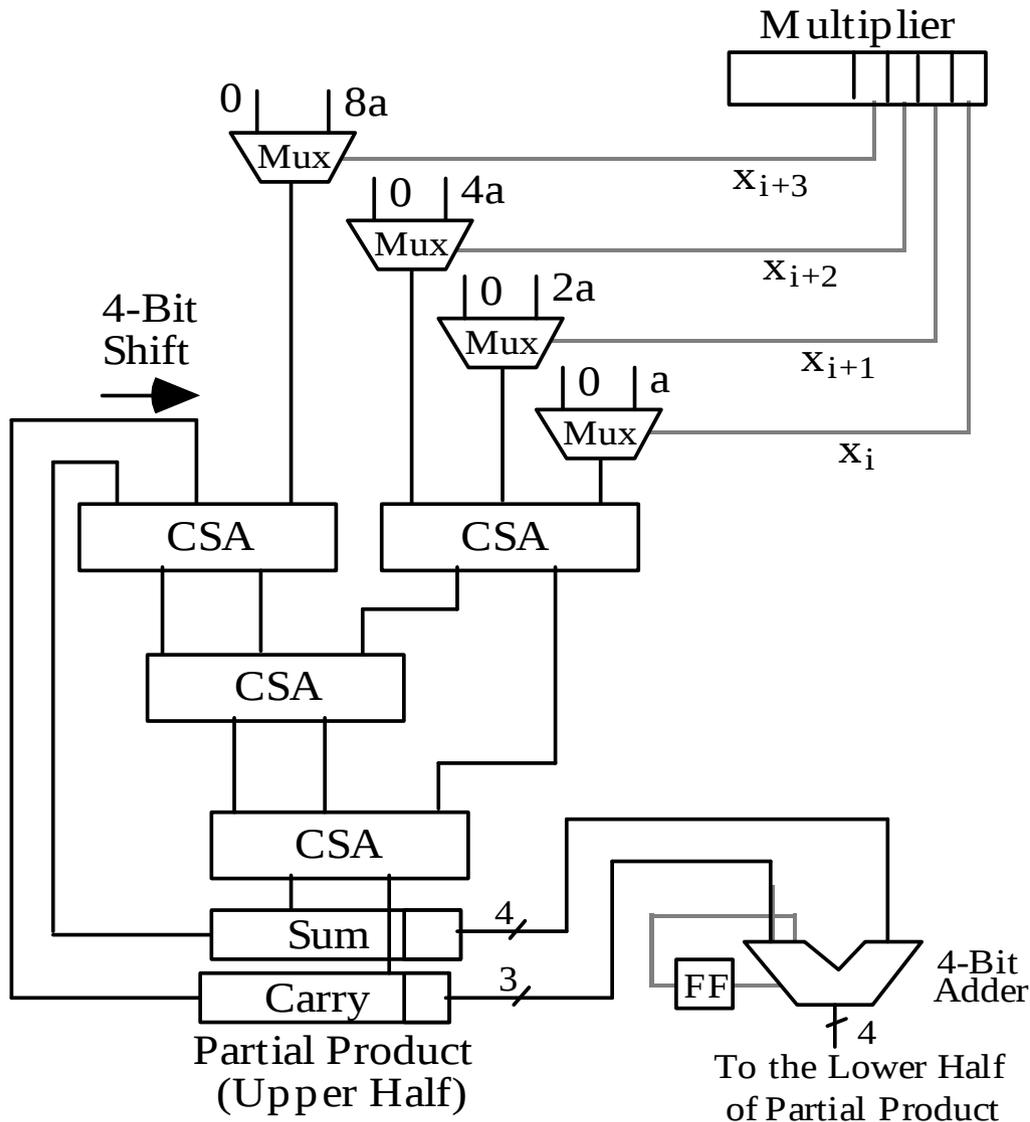
- $x_i A$
- $2x_{i+1} A$
- y  $4x_{i+2} A$



# Niveles de CSA

- Con 2 niveles de CSA reduce en 2 a la mitad la cantidad de iteraciones ( $n$ ).
- En cada iteración el camino más largo atraviesa los dos CSA.
- El retardo por los CSA es de  $2n \Delta_{\text{CSA}}$ .
- Ocurre algo análogo para 3 CSA.
- Con 4 o más CSA se puede lograr paralelismo (Wallace tree).

# Niveles de CSA



- La cantidad de iteraciones se reduce en 4.
- En cada iteración se atraviesan hasta 3 CSA.
- En total el retardo por CSA es de  $3/4 n \Delta_{CSA}$ .

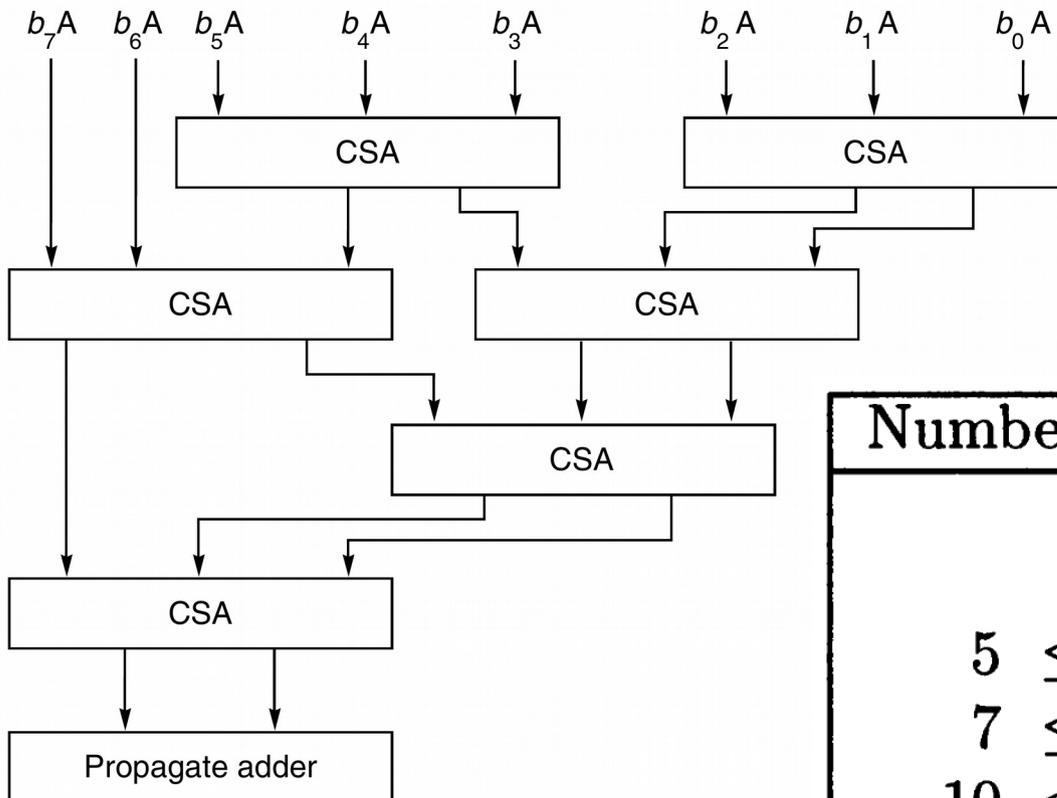
# Árbol de CSA

- Para maximizar el paralelismo se puede armar un Wallace Tree de  $n$  operandos de  $n$  bits.
- Para  $k = n$  operandos, los niveles de un Wallace Tree son aprox.

$$\frac{\log \frac{2}{n}}{\log \frac{2}{3}} = \frac{\log \frac{n}{2}}{\log \frac{3}{2}}$$

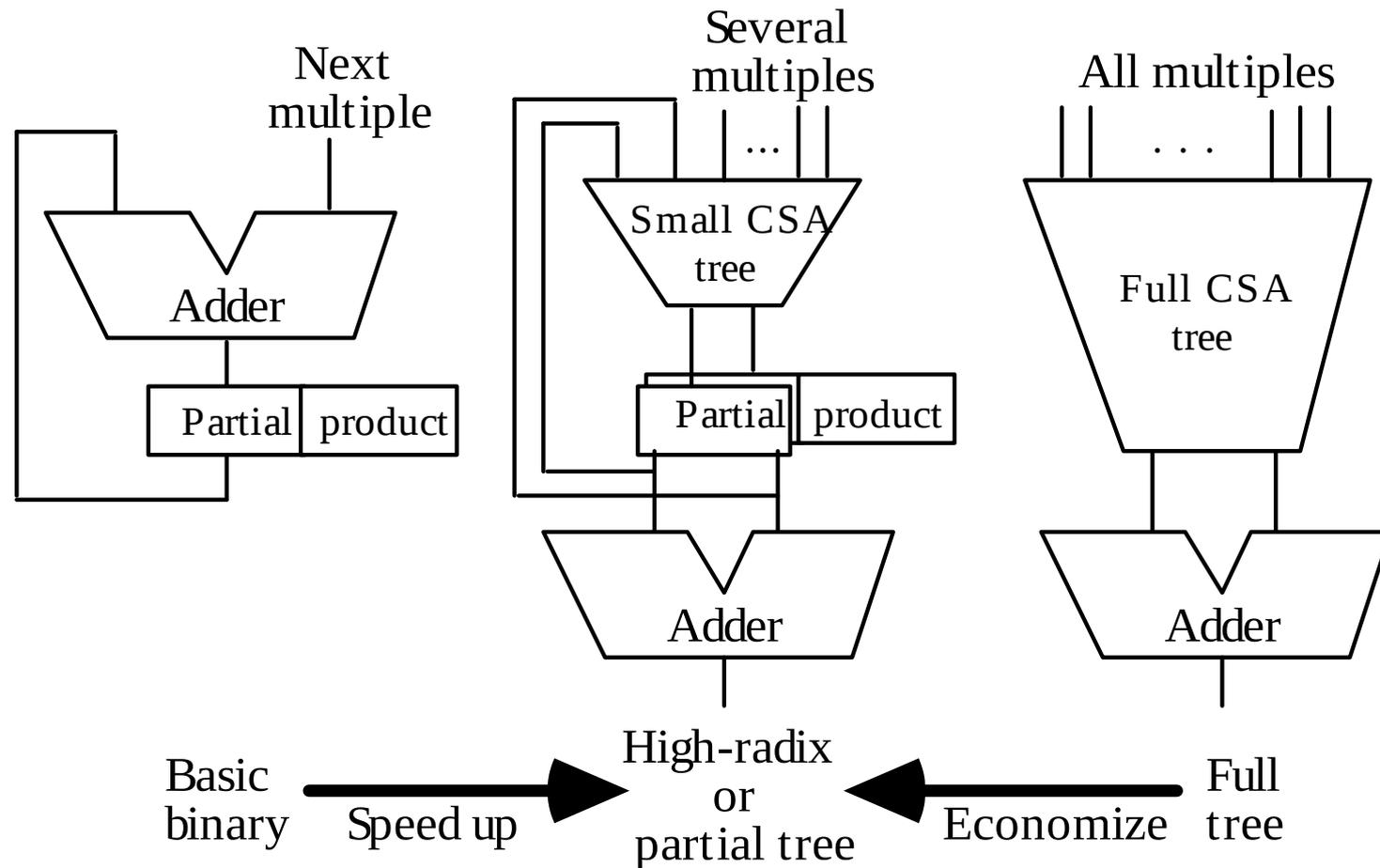
- El resultado final requiere atravesar  $O(\log(n))$  CSA.

# Wallace Tree



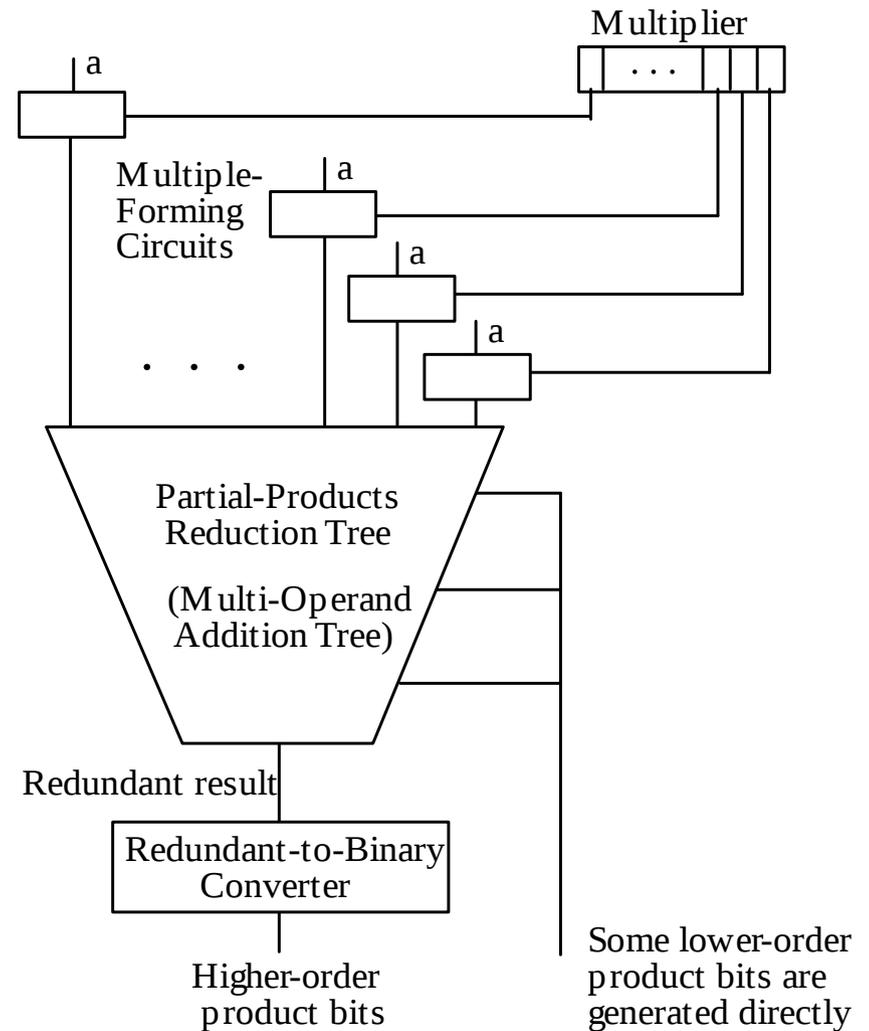
Number of operands	Number of levels
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9

# Velocidad vs. Tamaño



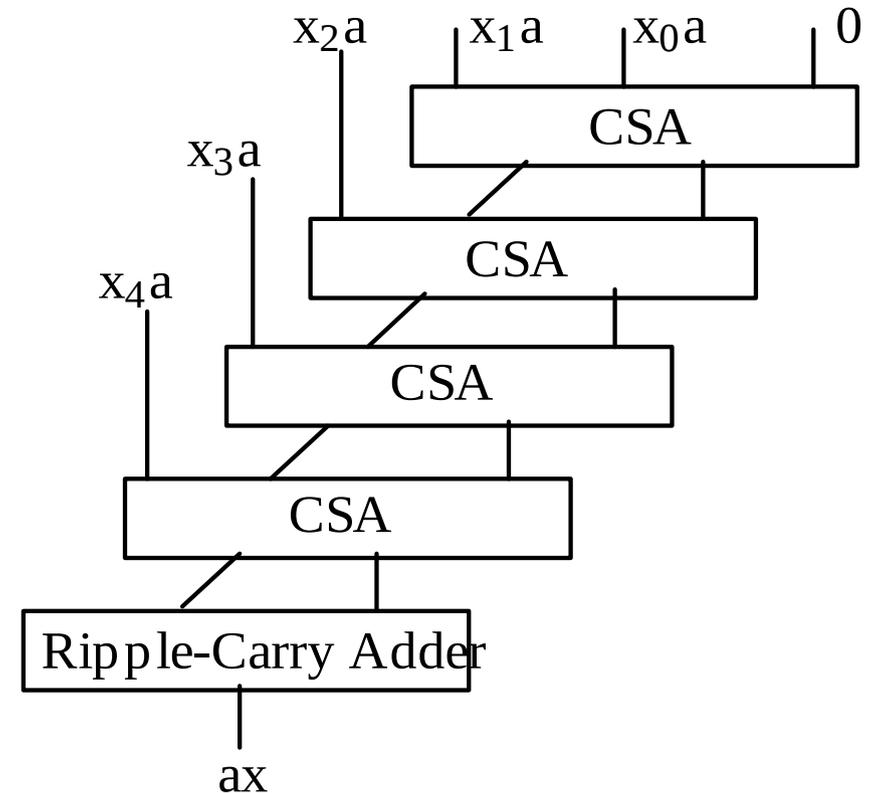
# Árbol multiplicador

- En su forma más general, varios múltiplos de  $A$ 
  - En binario
  - Recodificación de varios bits
  - Recodificación de Booth
- Un árbol de reducción (combinacional) genera los productos parciales.
- El resultado está en notación redundante.
- El resultado se convierte a binario.



# Array multiplier

- El árbol de reducción es una línea de CSA.
- Al final un ripple adder.
- ¿Por qué es interesante este multiplicador tan lento?

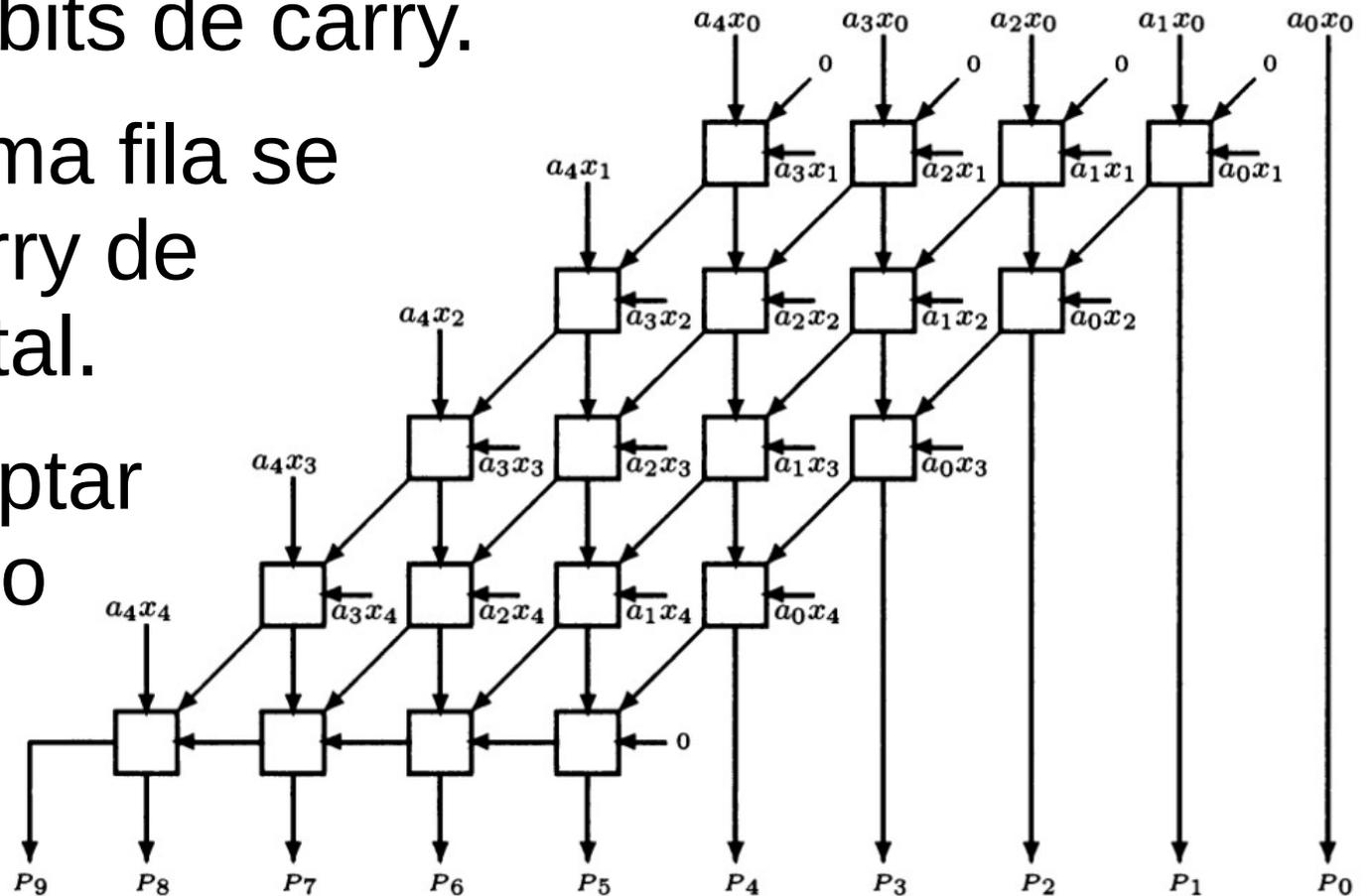


# Array multiplier

- Es un layout simple y eficiente para diseño VLSI.
  - Estructura muy regular.
  - Las conexiones son cortas (sólo hay que conectarse con el full adder adyacente vertical, horizontal o diagonal).

# Array multiplier

- Ninguna fila propaga carry.
- Cada producto parcial consiste de sumas intermedias y bits de carry.
- Sólo en la última fila se propaga el carry de forma horizontal.
- Se puede adaptar a complemento a 2



# Árbol binario redundante

- Se generan los múltiplos del multiplicando y se suman de a pares en dígito signado.
- La última suma, que es el producto en dígito signado, se convierte a binario.
- Dígito signado  $\{\bar{1}, 0, 1\}$  es notación redundante:
  - $5 = 0101 = 011\bar{1} = 1\bar{1}01 = 1\bar{1}1\bar{1} = 10\bar{1}\bar{1}$
- Esta redundancia permite sumar dos números binarios sin propagación de carry.

# Árbol binario redundante

- $S = A + B = a_{n-1}a_{n-2}\dots a_0 + b_{n-1}b_{n-2}\dots b_0 = s_n s_{n-1}\dots s_0$
- La suma sin propagación de carry se hace en dos pasos:
  - 1) Primero, se determinan un carry y suma intermedios, de tal forma que ( $a_i, b_i, c_{i+1}, s_i$  y  $d_i$  son dígitos signados):

$$a_i + b_i = 2c_{i+1} + d_i$$

- Se eligen  $c_{i+1}$  y  $d_i$  de forma que tal que no se genere carry en el próximo paso.

- 2) El el segundo paso,  $s_i$  se calcula sumando  $d_i$  y  $c_i$  del nivel anterior:  $s_i = d_i + c_i$

# Árbol binario redundante

- En  $a_i + b_i = 2c_{i+1} + d_i$  se eligen  $c$  y  $d$  de forma que no haya carry en función de los  $a$  y  $b$  anteriores:

$$\begin{array}{rcccccc}
 & a_i & 1 & 1 & \bar{1} & 0 \\
 + & b_i & 1 & \bar{1} & \bar{1} & 0 \\
 \hline
 c_{i+1}d_i & 10 & 00 & \bar{1}0 & 00 & 
 \end{array}$$

	$a_i$	1		$a_{i-1}$		$a_i$	$\bar{1}$		$a_{i-1}$
+	$b_i$	0		$b_{i-1}$		+	$b_i$	0	$b_{i-1}$
	$c_{i+1}d_i$	$1\bar{1}$	$a_{i-1} \geq 0$ y $b_{i-1} \geq 0$			$c_{i+1}d_i$	$0\bar{1}$	$a_{i-1} \geq 0$ y $b_{i-1} \geq 0$	
	01	Otro caso				$\bar{1}1$	Otro caso		

# Árbol binario redundante

Para multiplicar operandos de  $n$  bits:

- No requiere convertir los operandos a dígito signado.
- El primer nivel tiene  $n/2$  sumadores. El nivel  $k$  tendrá  $n/2^k$  sumadores.
- Todos los sumadores operan en dígito signado.
- En la última etapa, se debe convertir el dígito signado a binario restando la parte negativa a la parte positiva.

# Árbol binario redundante

- Tiene  $O(\log(n))$
- Es apto para implementación VLSI
  - Aunque tiene mayor lógica por bit (dígito signado requiere dos registros).
  - Es regular y también tiene conexiones cortas (cada sumador se conecta solo con dos del nivel siguiente)

# Bibliografía

- Capítulo 3 y 6. *Computer Arithmetic Algorithms.* Israel Koren, 2da Edición, A K Peters, Natick, MA, 2002.  
*Adapted from Koren, UMass. Copyright 2008 Koren, UMass and A.K. Peters.*
- Capítulo 10 y 11. *Computer Arithmetic: Algorithms and Hardware Designs.* Behrooz Parhami, 2nd edition, Oxford University Press, New York, 2002.
- Apéndice J. J. Hennessy & D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers INC. 2011, 5ta Ed.

## Suplementaria

- Capítulo 42. Editor Wai-Kai Chen. *The VLSI Handbook.* CRC Press. (2da Ed. 2007)