Estructura de Computadores

Tema 3
Aritmética de enteros

DISCA

Aritmética de enteros

- Objetivos:
  - ✓ Hacer la correspondencia entre los tipos de datos numéricos de alto nivel (p. ej, Java y C/C++) y los tipos nativos del procesador
  - ✓ Traducir a ensamblador expresiones aritméticas (cálculos y guardas) expresadas en alto nivel
  - ✓ Distinguir entre operadores combinacionales y secuenciales y calcular su tiempo de operación y la productividad en casos sencillos a partir de los retardos de los componentes
  - ✓ Relacionar manipulación de bits con operaciones de alto nivel (p. ej, operar con los campos del formato de coma flotante)
  - ✓ Distinguir las diferentes respuestas del computador ante las operaciones que no se pueden completar (por desbordamiento o por indeterminación)
  - ✓ Entender el soporte que da el juego de instrucciones a las singularidades del cálculo (excepciones, indicadores de la norma IEEE)

Índice

Índice

+ Archivo Tema 3. Introduccion.pdf

Tema 3. Aritmética de enteros

Repaso visto en vídeos 1 y 2

1. *La representación de los enteros*
2. *Tipos de datos en alto y bajo nivel*
3. *Suma y resta en el MIPS*
4. *Comparación de cantidades en el MIPS*

Tema 3. Aritmética de enteros

Repaso

1. La representación de los enteros

- ✓ C.B.N: código binario natural (NATURAL)
  - N bits = 2^N valores = +0 a $+2^N-1$
- ✓ C2: Complemento a dos (ENTEROS)
  - N bits = 2^{N-1} valores positivos = +0 a $+(2^{N-1})-1$

    2^{N-1} valores negativos = -1 a $-(2^{N-1})$

Tema 3. Aritmética de enteros

La representación de cantidades:

Con 4 bits:

1. *C.B.N: código binario natural (NATURAL)*
   - *N bits = 2^N valores = +0 a $+2^{N-1}$*
   - *Ejemplo: $1001 = 2^3 + 2^0 = 9$*

2. *C2: Complemento a dos (ENTEROS)*
   - *N bits = 2^{N-1} valores positivos = +0 a $+(2^{N-1}) -1$*
     2^{N-1} valores negativos = -1 a $-(2^{N-1})$
   - *Ejemplo: $1001 = -2^3 + 2^0 = -7$*

Según la instrucción que utilice el mismo dato puede ser una cantidad u otra

Tema 3. Aritmética de enteros

- ✓ Rangos de representación con *n* bits
  - Para N: C.B.N. Rango $[0 \ldots +2^n-1]$
  - Para Z: codificación en complemento a 2: $[-2^{n-1} \ldots +2^{n-1}-1]$

n	Sin signo	Con signo
8	0 … 255	–128 … +127
16	0 … 65.535	–32.768 … +32.767
32	0 … 4.294.967.295	–2.147.483.648 … +2.147.483.647
64	0 … $1.84 \cdot 10^{19}$ (aprox)	$-9.2 \cdot 10^{18}$ … $+9.2 \cdot 10^{18}$ (aprox)

Tema 3. Aritmética de enteros

Repaso visto en vídeos 1 y 2

1. La representación de los enteros
2. Tipos de datos en alto y bajo nivel
3. Suma y resta en el MIPS
4. Comparación de cantidades en el MIPS

Tema 3. Aritmética de enteros

- Tipos de datos básicos

lb/lbu sb
lh/lhu sh
lw/sw

	bits	Java	C/C++ (32 bits)	MIPS	x64/IA-64
carácter	8	—	char	byte	byte
	16	char	wchar_t	halfword	word
entero sin signo	8	—	unsigned byte	byte	byte
	16	—	unsigned short	halfword	word
	32	—	unsigned int (long)	word	dword
entero con signo	8	byte	byte	byte	byte
	16	short	short	halfword	word
	32	int	int /long	word	dword
	64	long		—	qword
coma flotante	32	float	float	float	float
	64	double	double	double	double

Unsigned significa C.B.N (o sea valor NATURAL sin signo)

Tema 3. Aritmética de enteros

- Tipos de datos básicos

lb/lbu sb
lh/lhu sh
lw/sw

U = unsigned

	bits	Java	C/C++ (32 bits)	MIPS	x64/IA-64
carácter	8	—	char	byte	byte
	16	char	wchar_t	halfword	word
entero sin signo CBN	8	—	unsigned byte	byte	byte
	16	—	unsigned short	halfword	word
	32	—	unsigned int (long)	word	dword
entero con signo C2	8	byte	byte	byte	byte
	16	short	short	halfword	word
	32	int	int /long	word	dword
	64	long		—	qword
coma flotante	32	float	float	float	float
	64	double	double	double	double

Unsigned significa C.B.N (o sea valor NATURAL sin signo)

Tema 3. Aritmética de enteros

Lh versus lhu:

1. **Lh $reg, dirección: es load half**
   - El valor es un entero en C2 en 16 bits almacenado en memoria, en dirección
   - Se va a almacenar en un registro de 32 bits, $reg
   - Se tiene que extender el signo a la parte superior del registro $reg para seguir conservando el formato
2. **Lhu $reg, dirección: es load half unsigned**
   - El valor es natural (CBN) en 16 bits almacenado en memoria, en dirección
   - Se va a almacenar en un registro de 32 bits, $reg
   - Se tiene que extender ceros a la parte superior del registro $reg para seguir conservando el formato

Tema 3. Aritmética de enteros

Lh versus lhu: Ejemplo

```
.data
var:  .half  -1  # -1_10 = 1111 1111 1111 1111_C2
```

1. **Lh $reg, dirección: es load half**
   - Lh $t0, var
   - $t0 = 111.....1 = 0xFFFFFFFF = -1_{10}
2. **Lhu $reg, dirección: es load half unsigned**
   - Lhu $t0,var
   - $t0 = 0 … 0 1…1 = 0x0000FFFF = $2^{16}-1 = 65535_{10}$

Tema 3. Aritmética de enteros

lb versus lbu:

1. **lb $reg, dirección: es load half**
   - El valor es un entero en C2 en 8 bits almacenado en memoria, en dirección
   - Se va a almacenar en un registro de 32 bits, $reg
   - Se tiene que extender el signo a la parte superior del registro $reg para seguir conservando el formato
2. **lbu $reg, dirección: es load half unsigned**
   - El valor es natural (CBN) en 8 bits almacenado en memoria, en dirección
   - Se va a almacenar en un registro de 32 bits, $reg
   - Se tiene que extender ceros a la parte superior del registro $reg para seguir conservando el formato

Tema 3. Aritmética de enteros

Repaso

1. La representación de los enteros

2. Tipos de datos en alto y bajo nivel

3. Suma y resta en el MIPS

- ✓ add (addi) : enteros (C2) :
  - genera excepción si hay desbordamiento aritmético
- ✓ addu(addiu): naturales (CBN)

Ejemplos

- Dada la siguiente declaración de variables en lenguaje C++,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

```
unsigned short a1 = 10;
unsigned int b1 = 1;
unsigned byte c1 = 128;

b1 = b1 + a1 + c1;
```

```
.data
a1: .half 10
b1: .word 1
c1: .byte 128
```

```
. text
lhu $t0, a1          #  Parte alta de $t0 será CERO
lw $t1,b1
lbu $t2,c1           # los 24 bits de más peso de $t2 serán CERO
addu $t1, $t1, $t2
addu $t0,$t0,$t1
sw $t0, b1
```

Ejemplos

- Dada la siguiente declaración de variables en lenguaje C++,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

b) ¿Qué cambiaría en el código si no fueran unsigned?

```
unsigned short a1 = 10;
unsigned int b1 = 1;
unsigned byte c1 = 128;

b1 = b1 + a1 + c1;
```

```
.data
a1: .half 10
b1: .word 1
c1: .byte 128
```

```
. text
lhu $t0, a1          #  Parte alta de $t0 será CERO
lw $t1,b1
lbu $t2,c1           # los 24 bits de más peso de $t2 serán CERO
addu $t1, $t1, $t2
addu $t0,$t0,$t1
sw $t0, b1
```

Ejemplos

- Dada la siguiente declaración de variables en lenguaje C++,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

b) ¿Qué cambiaría en el código si no fueran unsigned?

```
unsigned short a1 = 10;
unsigned int b1 = 1;
unsigned byte c1 = 128;

b1 = b1 + a1 + c1;
```

```
.data
a1: .half 10
b1: .word 1
c1: .byte 128

. text
lh $t0, a1          #  Parte alta de $t0 será SIGNO
lw $t1,b1
lb $t2,c1           # los 24 bits de más peso de $t2 serán SIGNO
add $t1, $t1, $t2
add $t0,$t0,$t1
sw $t0, b1
```

Además con 8 bits no se puede representar el 128

-128 a +127

Solo eliminar la palabra unsigned puede dar lugar a errores de cálculo

Ejemplos

- Dada la siguiente declaración de variables en lenguaje C++,

c) ¿Qué diferencia en rango de representación de enteros tiene una variable de tipo unsigned byte frente a una de tipo byte? Unsigned -128 a 127 byte 0 a 255

```
unsigned short a1 = 10;
unsigned int b1 = 1;
unsigned byte c1 = 128;

b1 = b1 + a1 + c1;
```

```
.data
a1: .half 10
b1: .word 1
c1: .byte 128
```

```
. text
lh $t0, a1
lw $t1,b1
lb $t2,c1
add $t1, $t1, $t2
add $t0,$t0,$t1
sw $t0, b1
```

Parte alta de $t0 será SIGNO

los 24 bits de más peso de $t2 serán SIGNO

Además con 8 bits no se puede representar el 128

-128 a +127

Solo eliminar la palabra unsigned puede dar lugar a errores de cálculo

Ejemplos

- Dada la siguiente declaración de variables en lenguaje Java,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

```
long a=10,b=2,c;

c = a ;
```

Ejemplos

- Dada la siguiente declaración de variables en lenguaje Java,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

```
long a=10,b=2,c;

c = a ;
```

```
.data 0x10000000
a: .word 10,0   #  en dos palabras consecutivas el 10
b: .word 2,0
c: .space 8     # solo se reservan 8 bytes
```

00 00 00 00 00 00 00 0A = 10_{10}

0x00	0x00	0x00	0x0A
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x02
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00

0x10000000 = a

0x10000004

0x10000008 = b

0x1000000C

0x10000010 = c

0x10000014

Ejemplos

- Dada la siguiente declaración de variables en lenguaje Java,

a) indique cómo se traducirían por un compilador a lenguaje del MIPS

```
long a=10,b=2,c;

c = a ;
```

```
.data
a: .word 10,0   #  en dos palabras consecutivas el 10
b: .word 2,0
c: .space 8     # solo se reservan 8 bytes
```

```
.text
        la $t0, a  #  $t0 es la dirección de variable a
        lw $t1,0($t0) # se lee la parte baja de a
        lw $t2,4($t0) # se lee la parte alta de a
        la $t0,c
        sw $t1, 0($t0)
        sw $t2,4($t0)
```

$t2||$t1 = long a

Tema 3. Aritmética de enteros

Repaso

1. Tipos de datos en alto y bajo nivel
2. La representación de los enteros
3. Suma y resta en el MIPS
4. Comparación de cantidades

Tema 3. Aritmética de enteros

Comparación de cantidades: Set on Less Than

1. Slt (slti) para enteros (C2)
2. Sltu (sltiu) para naturales (CBN)

Tema 3. Aritmética de enteros

Comparación de cantidades

1. Slt (slti) para enteros (C2)
2. Sltu (sltiu) para naturales (CBN)

SLT $rd, $r1, $r2

$rd = ¿ $r1 < $r2 ? (Si cierto es UNO, si falso es CERO)

O sea;

Si $rd = 1 entonces $r1 < $r2

Si $rd = 0 entonces $r1 ≥ $r2

Tema 3. Aritmética de enteros

Comparación de cantidades

1. Slt (slti) para enteros (C2)
2. Sltu (sltiu) para naturales (CBN)

SLTI $rd, $r1, valor

$rd = ¿ $r1 < valor? (Si cierto es UNO, si falso es CERO)

O sea;

Si $rd = 1 entonces $r1 < valor

Si $rd = 0 entonces $r1 ≥ valor

Tema 3. Aritmética de enteros

Comparación de cantidades

1. Slt (slti) para enteros (C2)
2. Sltu (sltiu) para naturales (CBN)

¿Qué importancia tienen estas instrucciones?

Tema 3. Aritmética de enteros

Comparación de cantidades

1. Slt (slti) para enteros (C2)
2. Sltu (sltiu) para naturales (CBN)

¿Qué importancia tienen estas instrucciones?

- Instrucciones de salto condicional en MIPS son:
  - Beq, bne, beqz, bnez, bgtz, etc…
    - Solo comparan un registro con $zero, o
    - si dos registros son iguales o diferentes
- Para comparar registros hay que utilizarlas

Ejemplo: Pseudoinstrucciones salto

Blt $r1, $r2, etiqueta Branch Less Than

si $r1 < $r2 entonces salto a etiqueta

sino no salto

El ensamblador traduce por:

```
Slt  $1, $r1,  $r2    #  $1 = (¿$r1 < $r2?)
bnez $1,etiqueta  #  Salta a etiqueta si $1 = 1 ($1 ≠ 0)
```

NOTA: $r1 y $r2 SON ENTEROS (con signo)

Ejemplo: Pseudoinstrucciones salto

Blt $r1, $r2, etiqueta Branch Less Than

si $r1 < $r2 entonces salto a etiqueta

sino no salto

El ensamblador traduce por:

```
Slt  $1, $r1,  $r2    #  $1 = (¿$r1 < $r2?)
bnez $1,etiqueta  #  Salta a etiqueta si $1 = 1 ($1 ≠ 0)
```

O sea SALTO si CIERTO

NOTA: $r1 y $r2 SON ENTEROS (con signo)

Ejemplo: Pseudoinstrucciones salto

¿ y si los valores son naturales (CBN)?

Bltu $r1, $r2, etiqueta Branch Less Than unsigned

si $r1 < $r2 entonces salto a etiqueta

sino no salto

El ensamblador traduce por:

```
Sltu $1, $r1, $r2    # $1 = (¿$r1 < $r2?)
Bnez $1,etiqueta  # Salta a etiqueta si $1 = 1 ($1 ≠ 0)
                        ( o sea SALTO si CIERTO)
```

NOTA: $r1 y $r2 SON NATURALES (sin signo, CBN)

Tema 3. Aritmética de enteros

Ejemplo: Pseudoinstrucciones salto

¿ y si la comparación es menor igual?

Ble $r1, $r2, etiqueta *Branch Less Equal*

si $r1 ≤ $r2 entonces salto a etiqueta

sino no salto (aquí $r1 > $r2)

NOTA: $r1 y $r2 SON ENTEROS

Ejemplo: Pseudoinstrucciones salto

¿ y si la comparación es menor igual?

Ble $r1, $r2, etiqueta Branch Less Equal

si $r1 ≤ $r2 entonces salto a etiqueta

sino no salto (aquí $r1 > $r2)

Le doy la vuelta a la comparación:

Si $r2 < $r1 entonces No salto

sino salto  Es decir ahora salto SI FALSO (CERO)

NOTA: $r1 y $r2 SON ENTEROS

Ejemplo: Pseudoinstrucciones salto

¿ y si la comparación es menor igual?

Ble $r1, $r2, etiqueta Branch Less Equal

si $r1 ≤ $r2 entonces salto a etiqueta

sino no salto (aquí $r1 > $r2)

Le doy la vuelta a la comparación:

Si $r2 < $r1 entonces No salto

sino salto ⇦ Es decir ahora salto SI FALSO (CERO)

```
slt $1, $r2, $r1    # $1 = (¿$r2< $r1?)
beqz $1,etiqueta  # Salta a etiqueta si $1 = 0
                                Ahora SALTO si FALSO
```

NOTA: $r1 y $r2 SON ENTEROS

Ejemplo: Pseudoinstrucciones salto

¿ y si la comparación es menor igual?

Ble $r1, $r2, etiqueta Branch Less Equal

si $r1 ≤ $r2 entonces salto a etiqueta

sino no salto (aquí $r1 > $r2)

Le doy la vuelta a la comparación:

Si $r2 < $r1 entonces No salto

sino salto  Es decir ahora salto SI FALSO (CERO)

```
slt $1, $r2, $r1    # $1 = (¿$r2< $r1?)
beqz $1,etiqueta  # Salta a etiqueta si $1 = 0
```

Ahora SALTO si FALSO

NOTA: $r1 y $r2 SON ENTEROS

Ejemplos

Programe el código equivalente de las siguientes pseudoinstrucciones

```
Bgt $t0, $t1, etiqueta

si  $t0 >  $t1  entonces   salto a etiqueta
sino    no salto  ( aquí  $t0 ≤ $t1)
```

Ejemplos

Programe el código equivalente de las siguientes pseudoinstrucciones

Hacer para casa:

bgeu

bleu

Ejemplos

¿Qué hace este código? Con NATURALES

```
    if:     sltu  $t0, $a1, $a0
            bnez $t0, then
else:       move $v0, $a1
            b fif

then:       move $v0, $a0
  fif:
```

Tema 3. Aritmética de enteros

RESUMEN de primeros puntos

- *En MIPS R2000 las instrucciones de salto condicional son beq, bne y las que comparan un registro con cero (bnez, bltz, etc)*
- *Con valores enteros se pueden emplear las pseudoinstrucciones de salto condicional: bge, bgt, blt …*
- *Con valores naturales se debe emplear bgeu, bgtu, bltu…*
- *Se debe comprender el uso de slt/slti, sltu/sltiu para comprender el comportamiento del código (sobre todo para prácticas)*

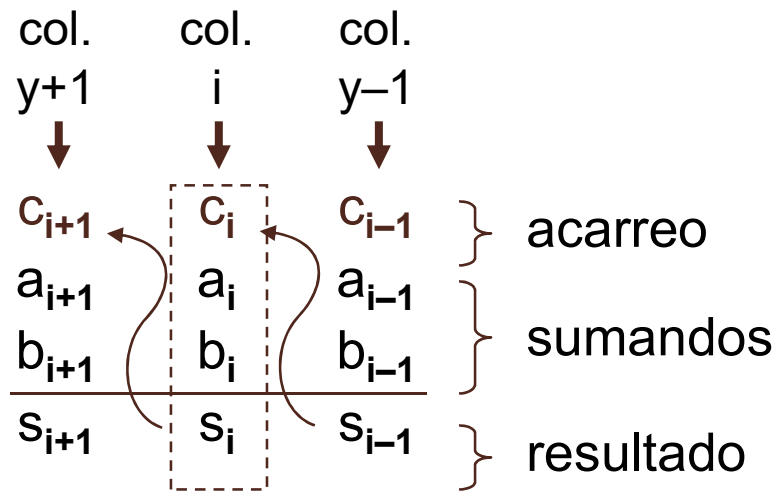
Índice

Índice

Índice

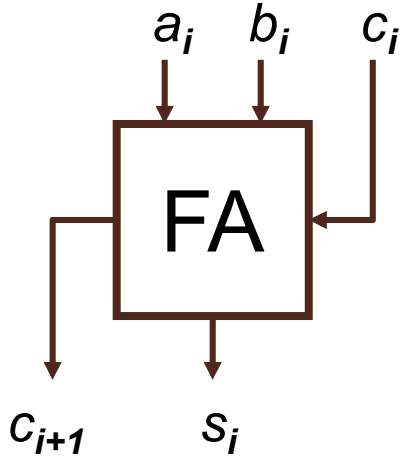
Operadores de suma

- La suma en serie
  - ✓ La suma en serie reproduce el procedimiento humano de suma
  - ✓ La suma se hace por orden, desde el LSB hacia el MSB
  - ✓ En cada columna *i*, hay que sumar los bits de los sumandos a_i, b_i y el acarreo c_i que llega de la columna $i-1$ para obtener el bit del resultado s_i y generar el acarreo c_{i+1} hacia la columna $i+1$ siguiente
  - ✓ Transporte de entrada $c_0 = 0$



Operadores de suma

- El sumador completo de un bit *(Full adder)*:
  - ✓ Implementa los cálculos de una columna de la suma en serie
  - ✓ Admite 3 entradas de un bit y produce dos salidas:



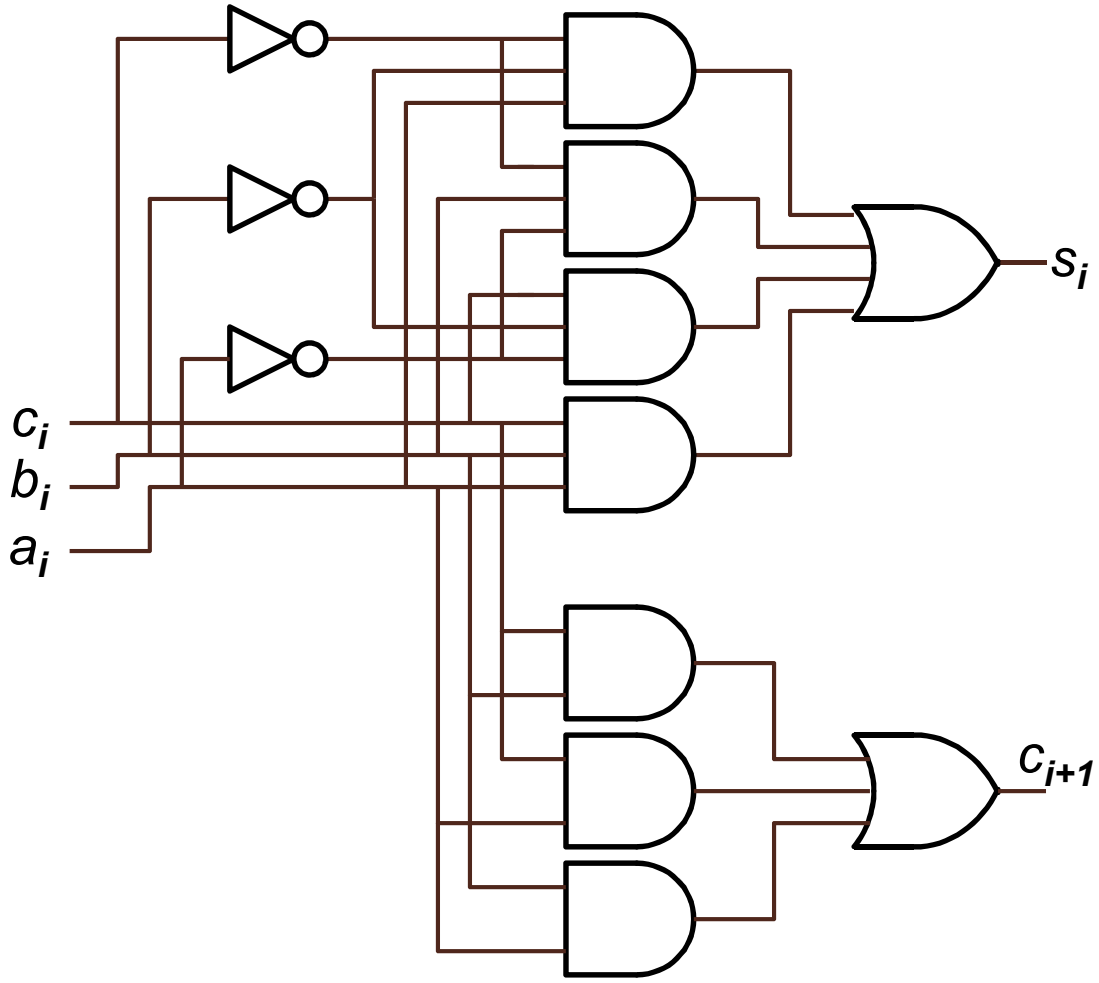
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s_i = \overline{a_i} \cdot \overline{b_i} \cdot c_i + \overline{a_i} \cdot b_i \cdot \overline{c_i} + a_i \cdot \overline{b_i} \cdot \overline{c_i} + a_i \cdot b_i \cdot c_i$$

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$

Operadores de suma

- El sumador completo de un bit: Implementación
  - ✓ Implementación a partir de las funciones lógicas

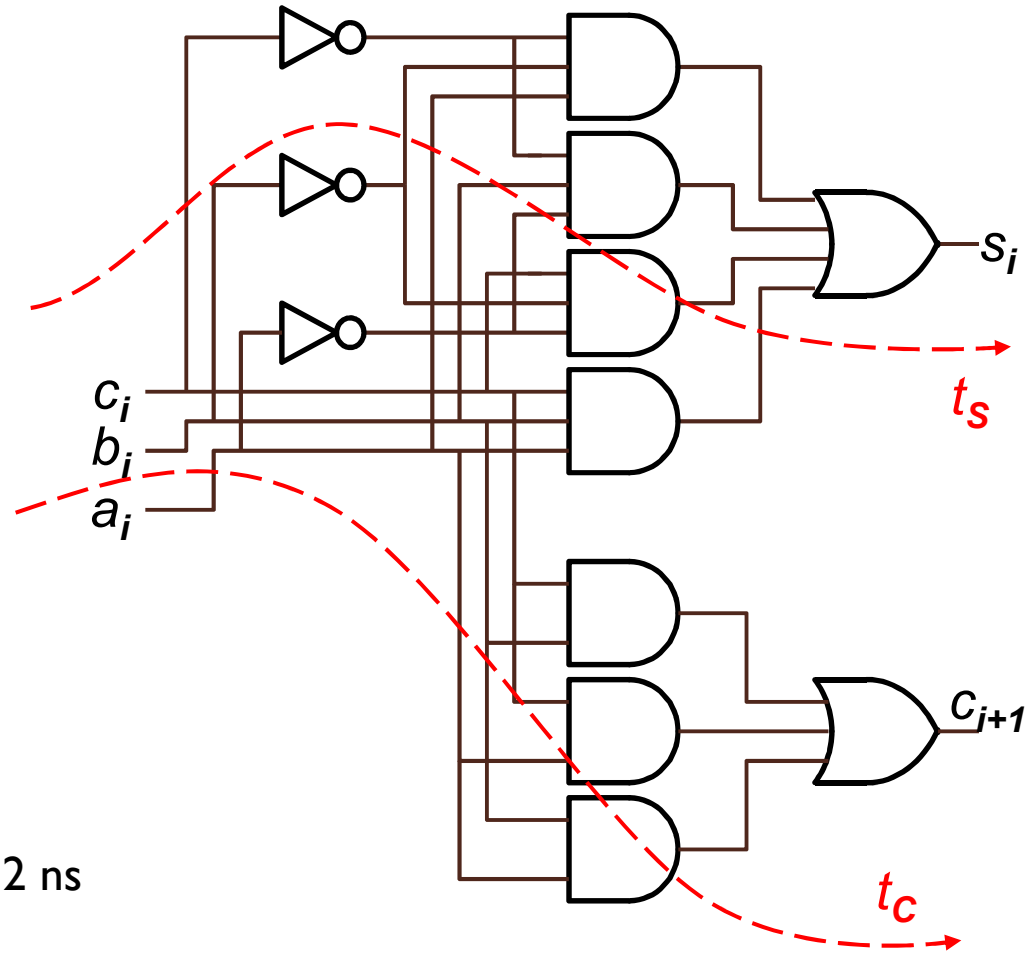


Operadores de suma

- El sumador completo de un bit: Prestaciones
  - ✓ Tiempo de retardo:

$$t_S = t_{NOT} + t_{AND} + t_{OR}$$

$$t_C = t_{AND} + t_{OR}$$

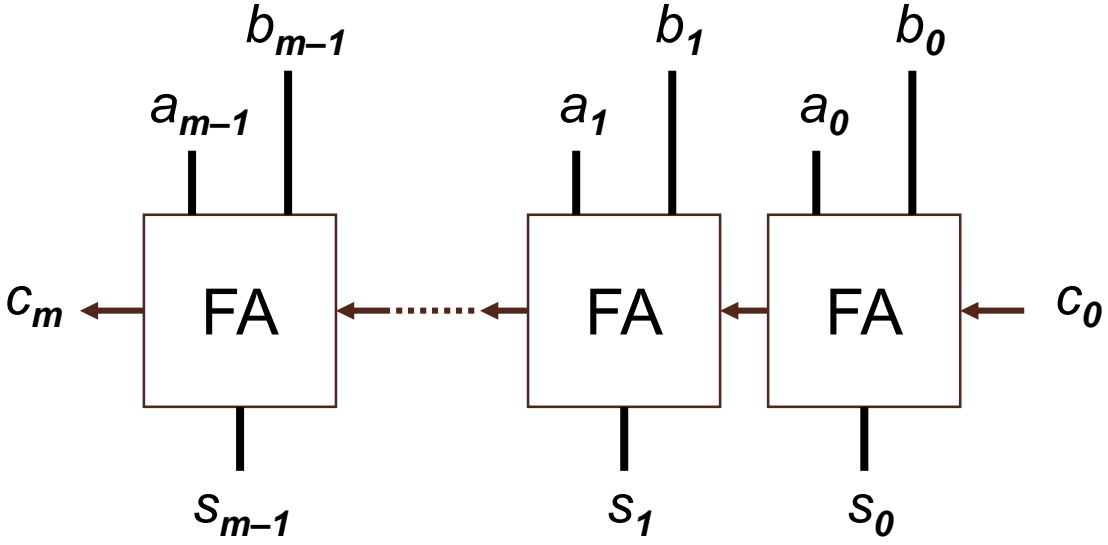


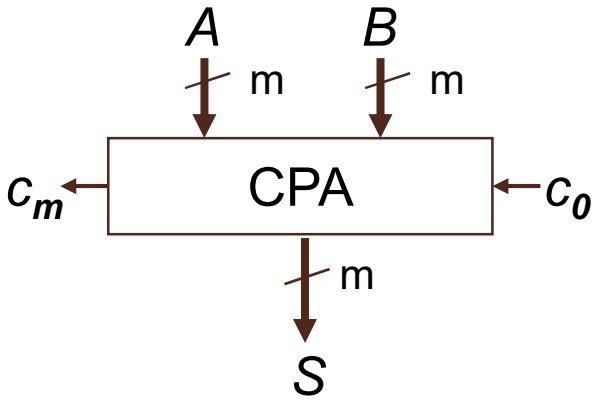
  - ✓ Complejidad: 12 puertas
  - ✓ Con tecnología CMOS 0.5 µm
    - tiempo de respuesta: entre 1 y 2 ns
    - superficie: 1000 µm²

Índice

Operadores de suma

- El sumador serie de *n* bits
  - ✓ CPA (Carry Propagation Adder)
  - ✓ Par hacer un sumador de dos números de *m* bits, se conectan *m* sumadores completos en cascada
    - la salida de acarreo del sumador *i*-ésimo se conecta a la entrada de acarreo del sumador *i*+*l*-ésimo
    - el operador resultante tiene una entrada y una salida de acarreo globales

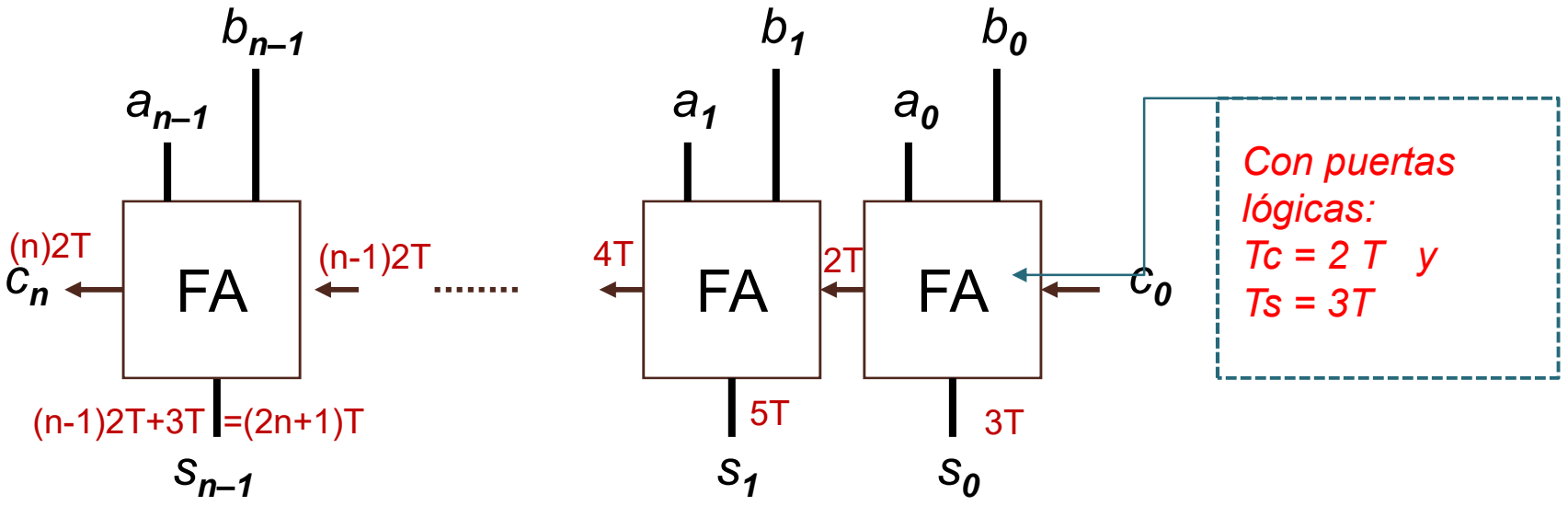




Operadores de suma

- Sumador CPA
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

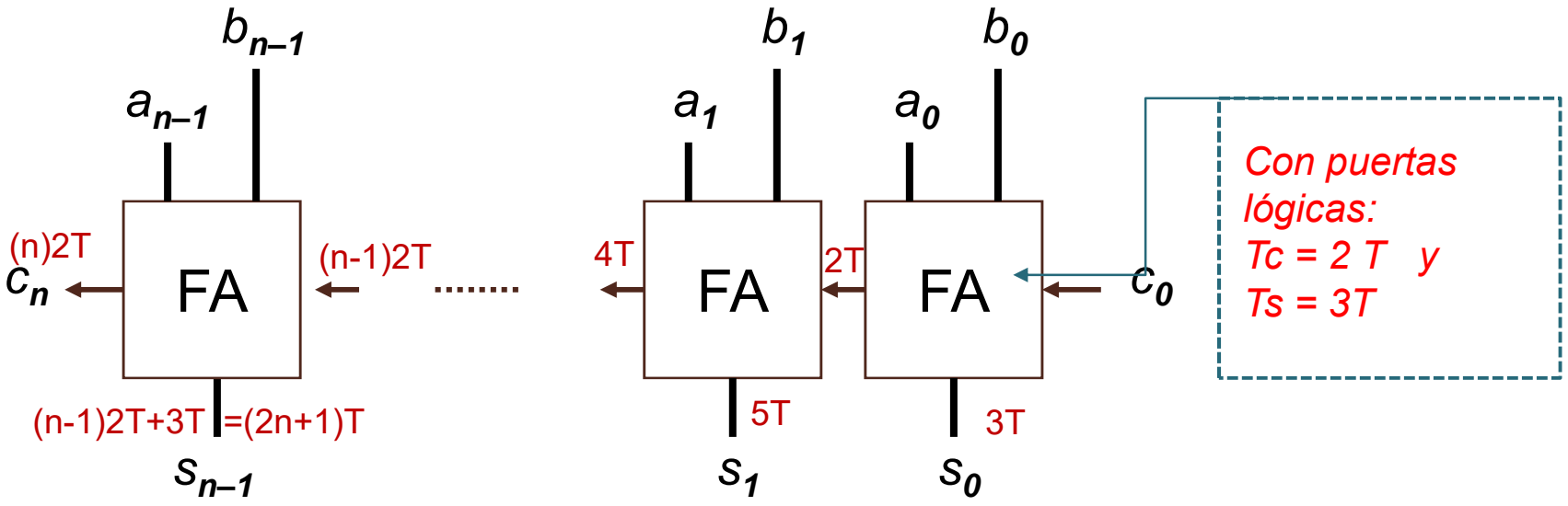
$$T_{op}(n\ bits) =$$



RESUMEN

- Sumador CPA
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (n-1)\cdot t_c + máx\{t_c, t_s\} =$$

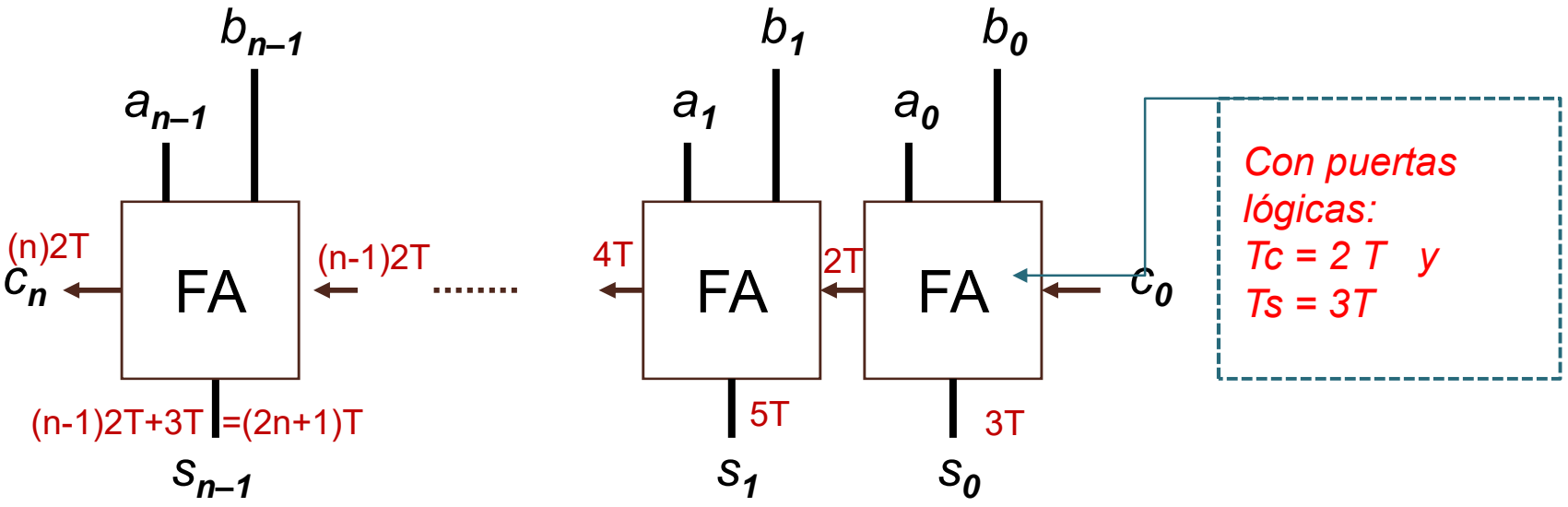


RESUMEN

- Sumador CPA
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (n-1)\cdot t_C + máx\{t_C, t_S\} =$$

$$T_{op}(n\ bits) = 2\,(n-1)\,T + 3T = 2nT - 2T + 3T$$



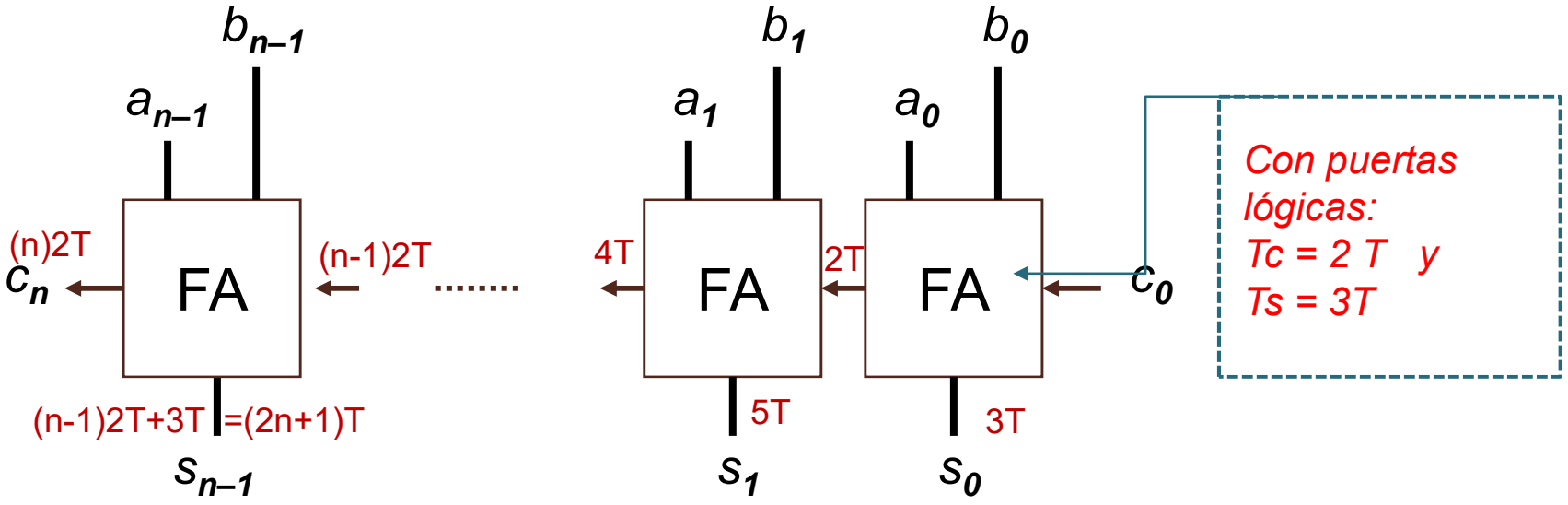
RESUMEN

- Sumador CPA
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (n-1)\cdot t_C + máx\{t_C, t_S\} =$$

$$T_{op}(n\ bits) = 2\,(n-1)\,T + 3T = 2nT - 2T + 3T$$

$$T_{op}(n\ bits) = (2n+1)T$$



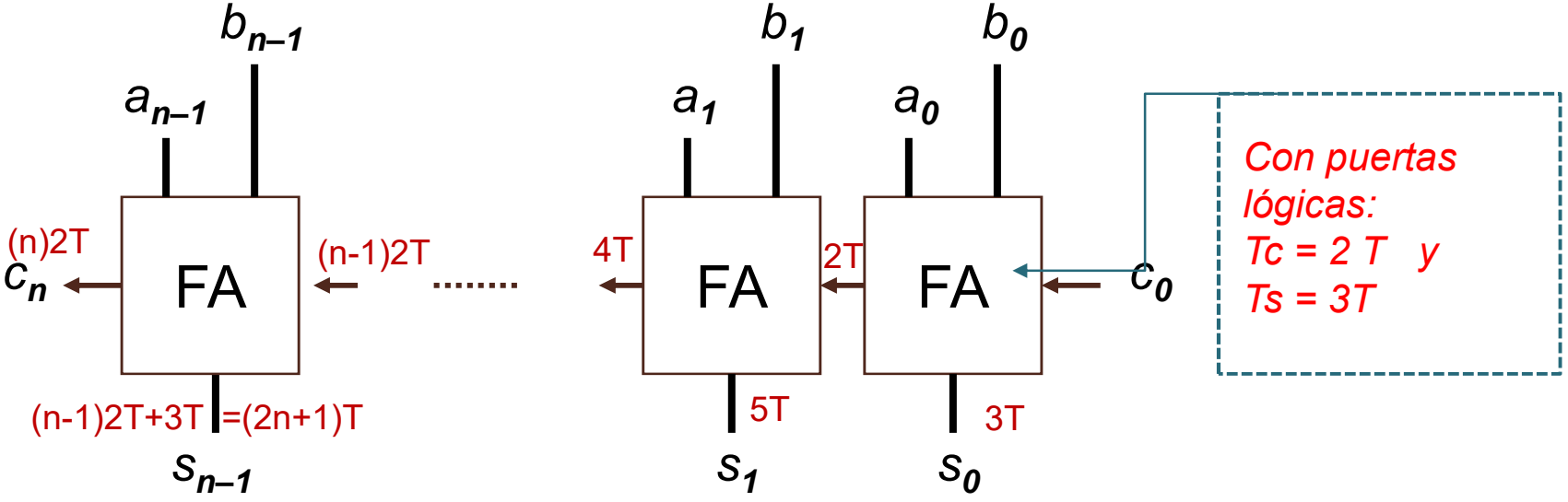
RESUMEN

- Sumador CPA
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (n-1)\cdot t_C + máx\{t_C, t_S\} =$$

$$T_{op}(n\ bits) = 2\,(n-1)\,T + 3T = 2nT - 2T + 3T$$

$$T_{op}(n\ bits) = (2n+1)T$$

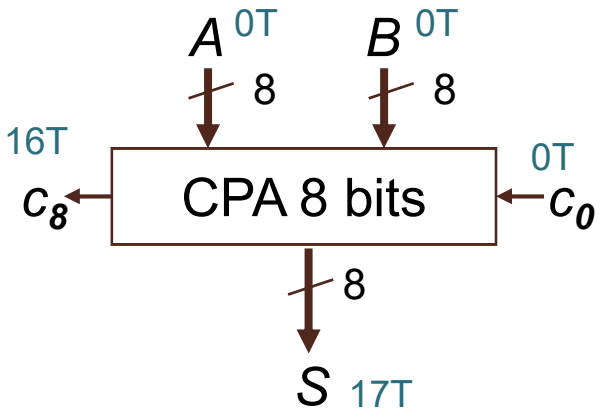


RESUMEN

- Sumador CPA : Particularizando a 8 bits
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (2n+1)T$$

$$T_{op}(8\ bits) = (2*8+1)T = 17\ T$$



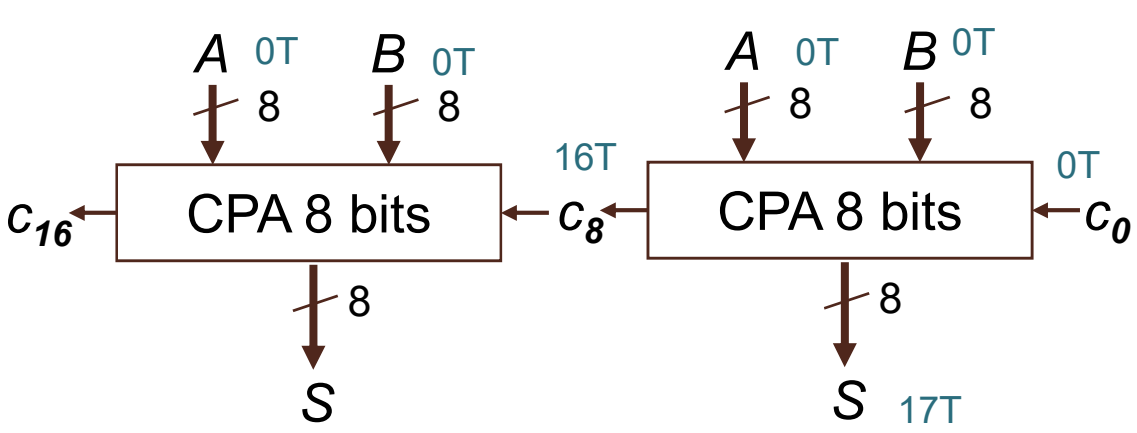
Con puertas lógicas:
Tc = 2 T y
Ts = 3T

RESUMEN

- Sumador CPA : ¿Y si se ponen dos en serie?
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 2: No todas las entradas llegan a la vez

$$T_{op}(n\ bits) = Llegada\ más\ lenta + (2n+1)T$$

$T_{op}(16\ bits) = ¿?$



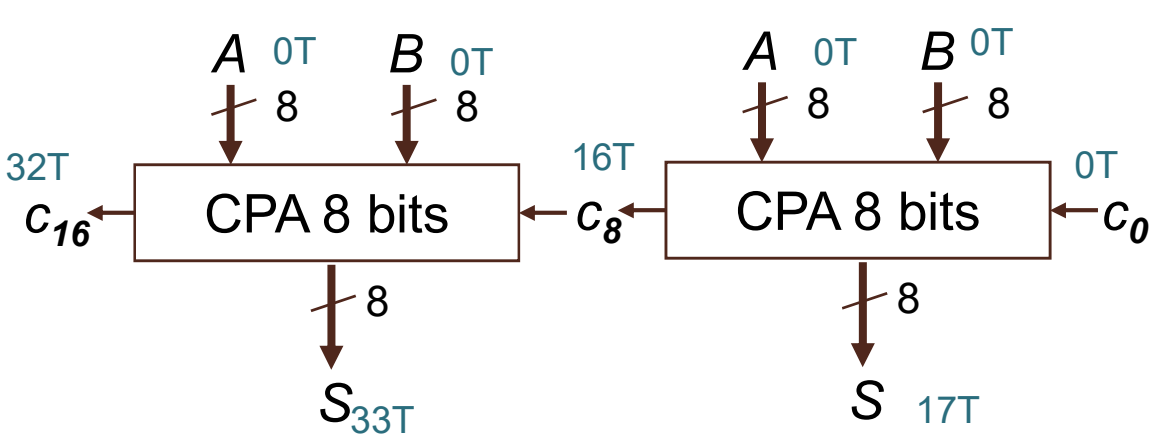
Con puertas lógicas:
Tc = 2 T y
Ts = 3T

RESUMEN

- Sumador CPA : ¿Y si se ponen dos en serie?
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 2: No todas las entradas llegan a la vez

$$T_{op}(n\ bits) = Llegada\ más\ lenta + (2n+1)T$$

$$T_{op}(16\ bits) = T_{c8} + T(8\ bits) = 16\ T + 17T = 33\ T$$



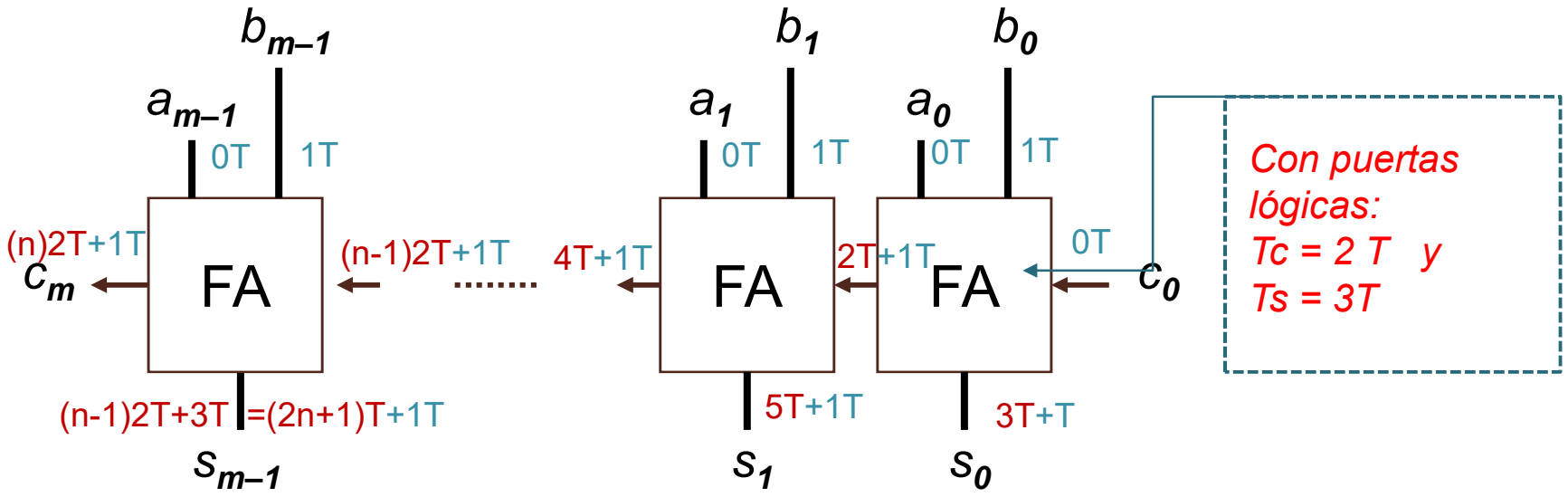
Con puertas lógicas:
Tc = 2 T y
Ts = 3T

RESUMEN

- Sumador CPA ¿Y si llega más lenta B?
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 2: Alguna entrada a_i b_i c_0 llega más tarde
    - Ejemplo: B llega 1T más tarde que A y c_0

$$T_{op}(n\ bits) = max\{a_i, b_i, c_0\} + (2n+1)T$$

$$T_{op}(n\ bits) = 1T + (2n+1)T$$



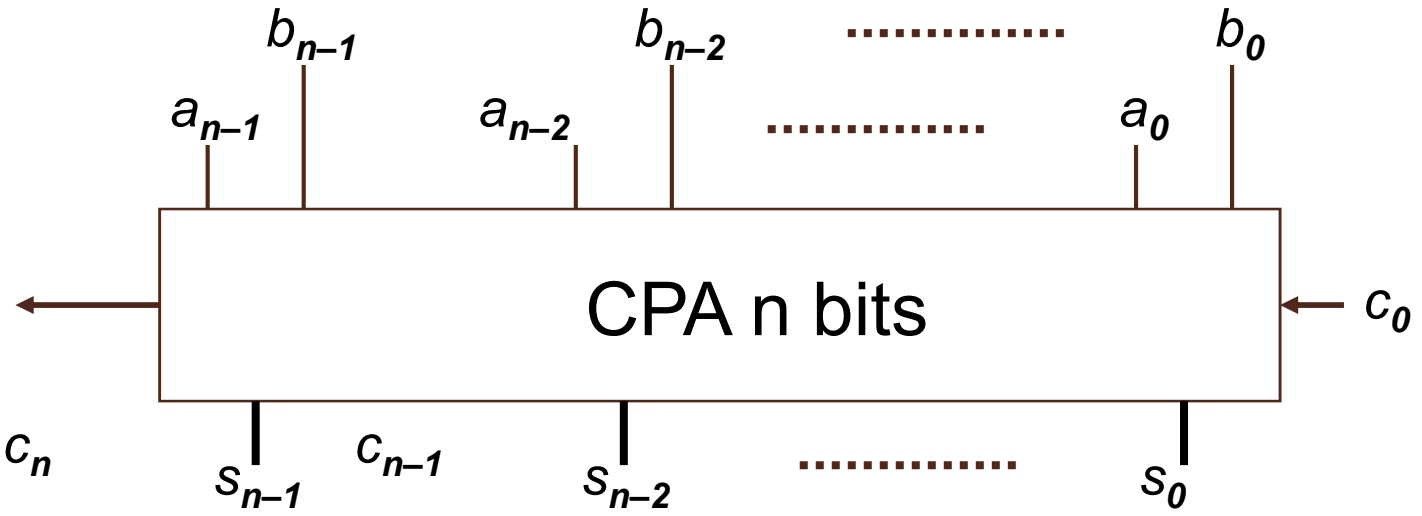
RESUMEN

- Sumador CPA (FA con las puertas lógicas vistas)
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador
    - Caso 1: Todas las entradas a la vez

$$T_{op}(n\ bits) = (2n+1)T$$

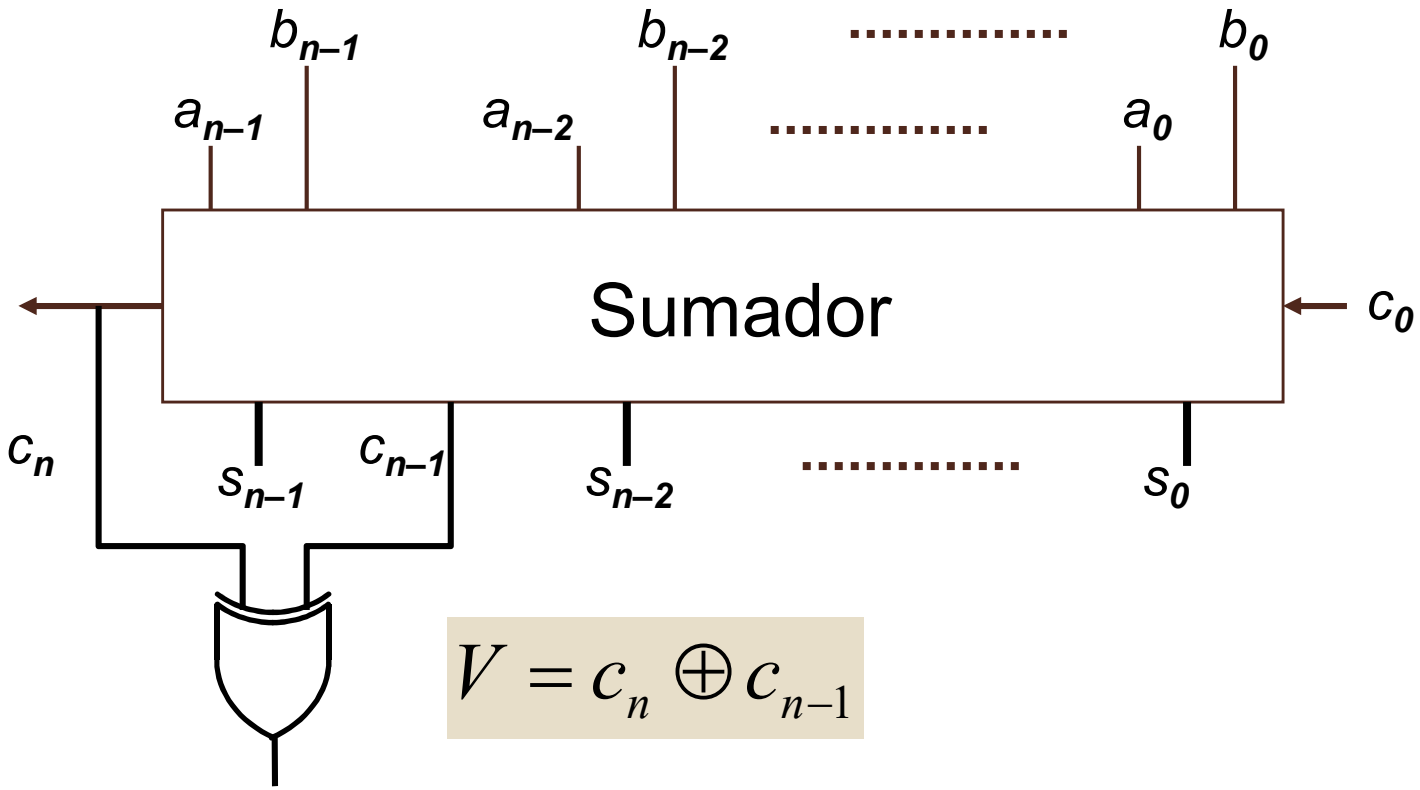
    - Caso 2: T_{op} = Alguna entrada más lenta

$$T_{op}(n\ bits) = Tretraso\ Entrada + (2n+1)T$$



Operadores de suma

- Cálculo del desbordamiento en la aritmética en Ca2
  - ✓ Se detecta cuando los bits de acarreo de orden *n* y *n–1* no son iguales
  - ✓ El signo del resultado s_{n-1} es incorrecto



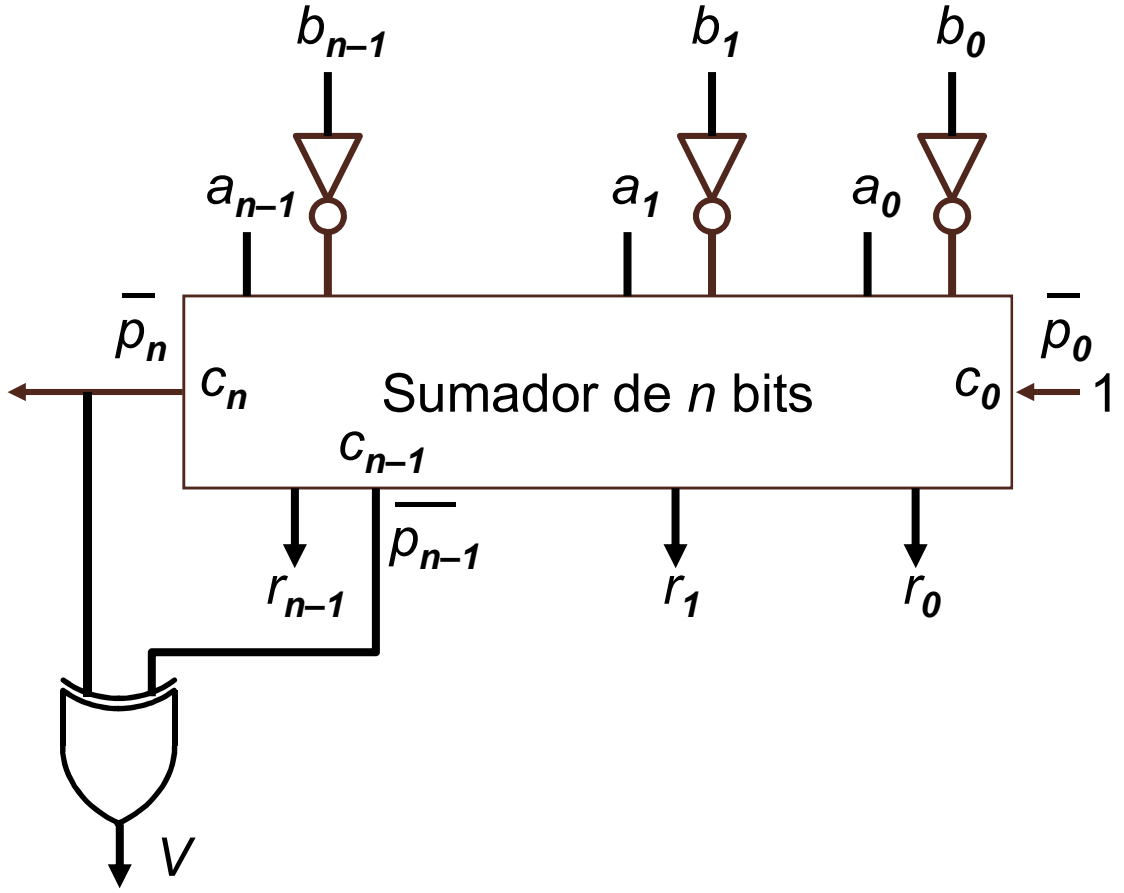
$$V = c_n \oplus c_{n-1}$$

En la aritmética CBN el desbordamiento V = Cn

Índice

Operadores de resta y comparación

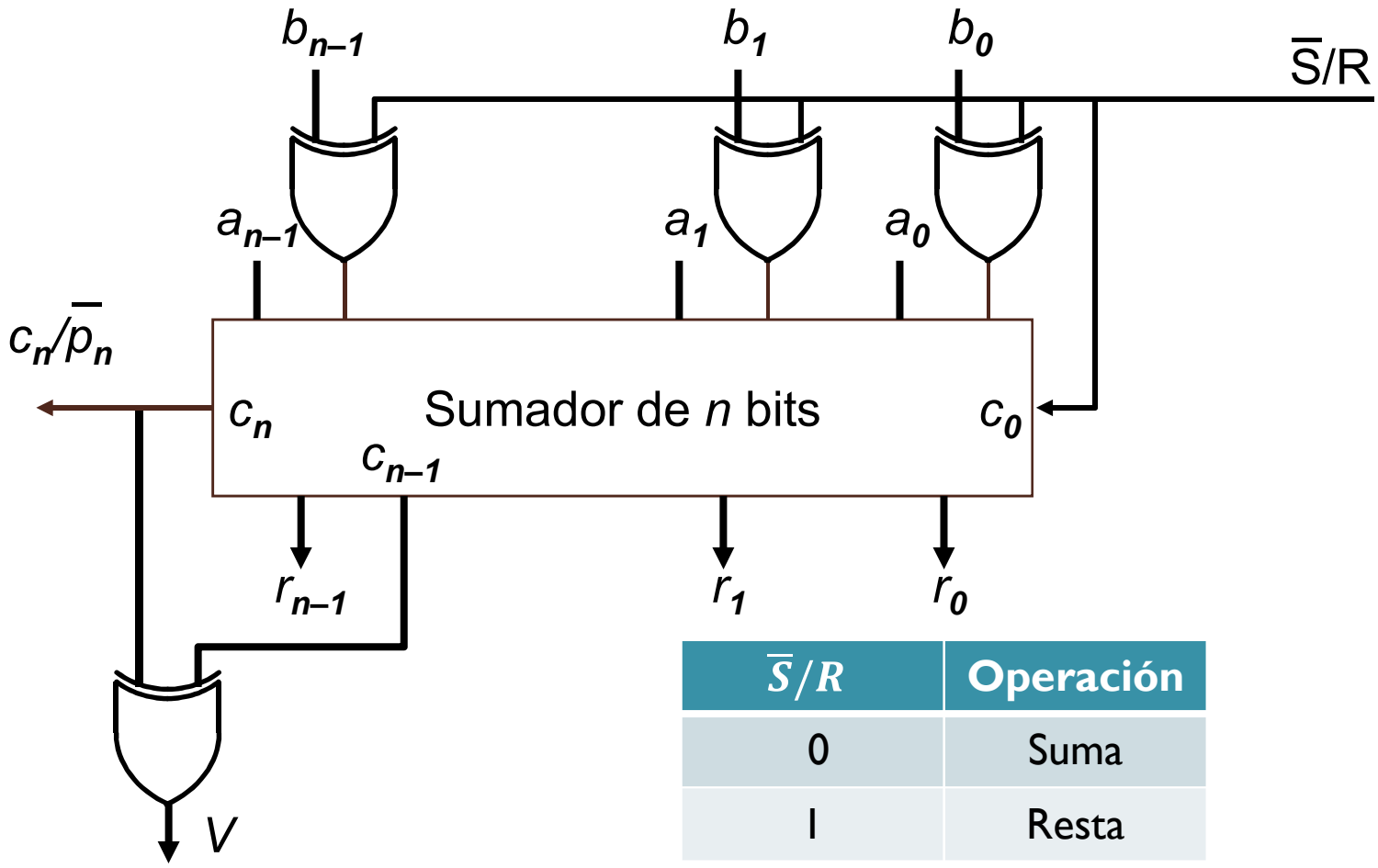
- Diseño del restador a partir del sumador
  - ✓ R = A–B = A + Ca2(B) = A + not(B) + 1
  - ✓ La detección de desbordamiento es igual que con la suma:
    - En CBN: p_n=**1** (**c_n=*0***)
    - En Ca2: $p_n \neq p_{n-1}$ (**c_n** xor **c_{n-1}=*1***)



Índice

Operadores de resta y comparación

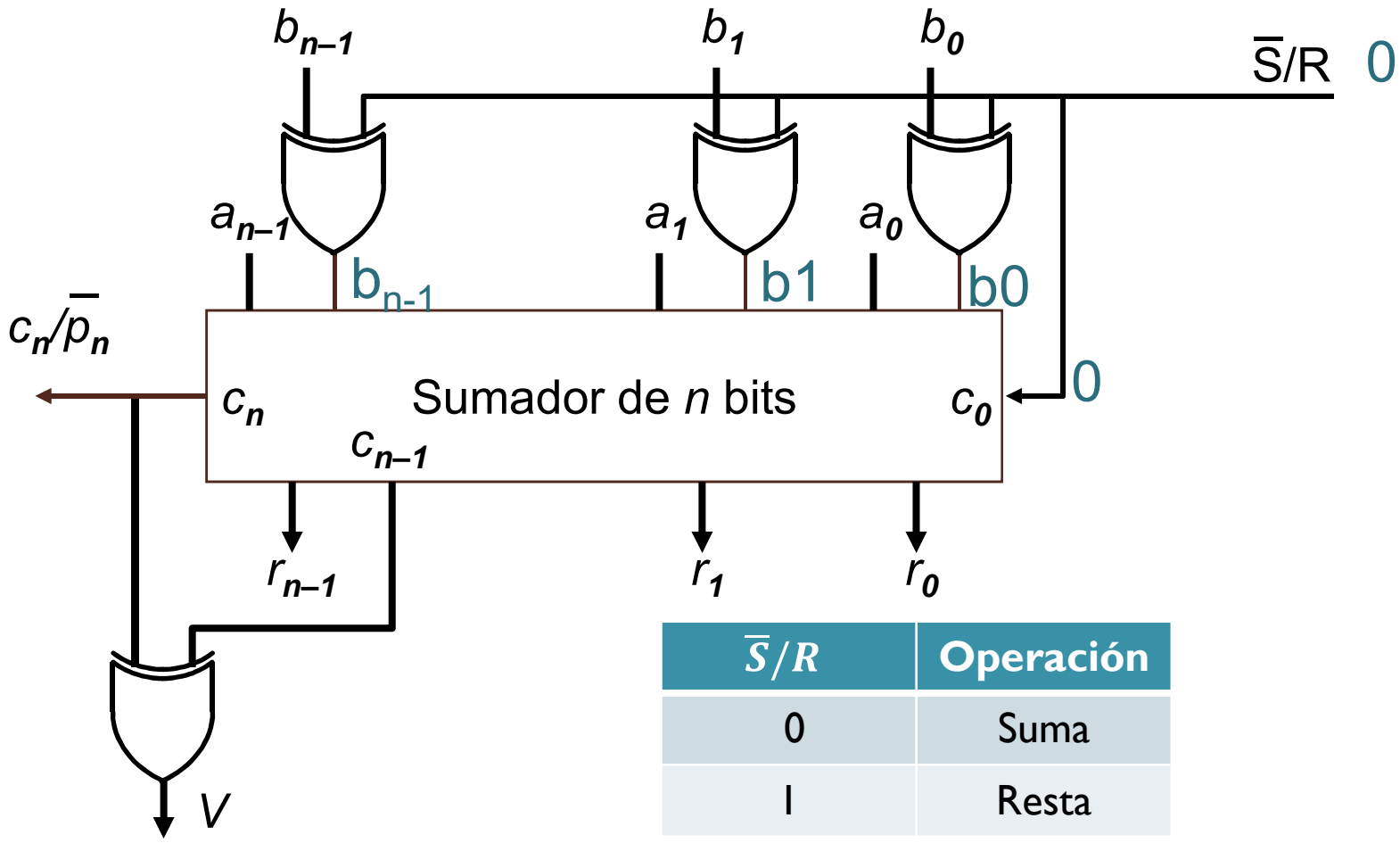
- Diseño clásico del sumador/restador



$\overline{S}/R$	Operación
0	Suma
1	Resta

Operadores de resta y comparación

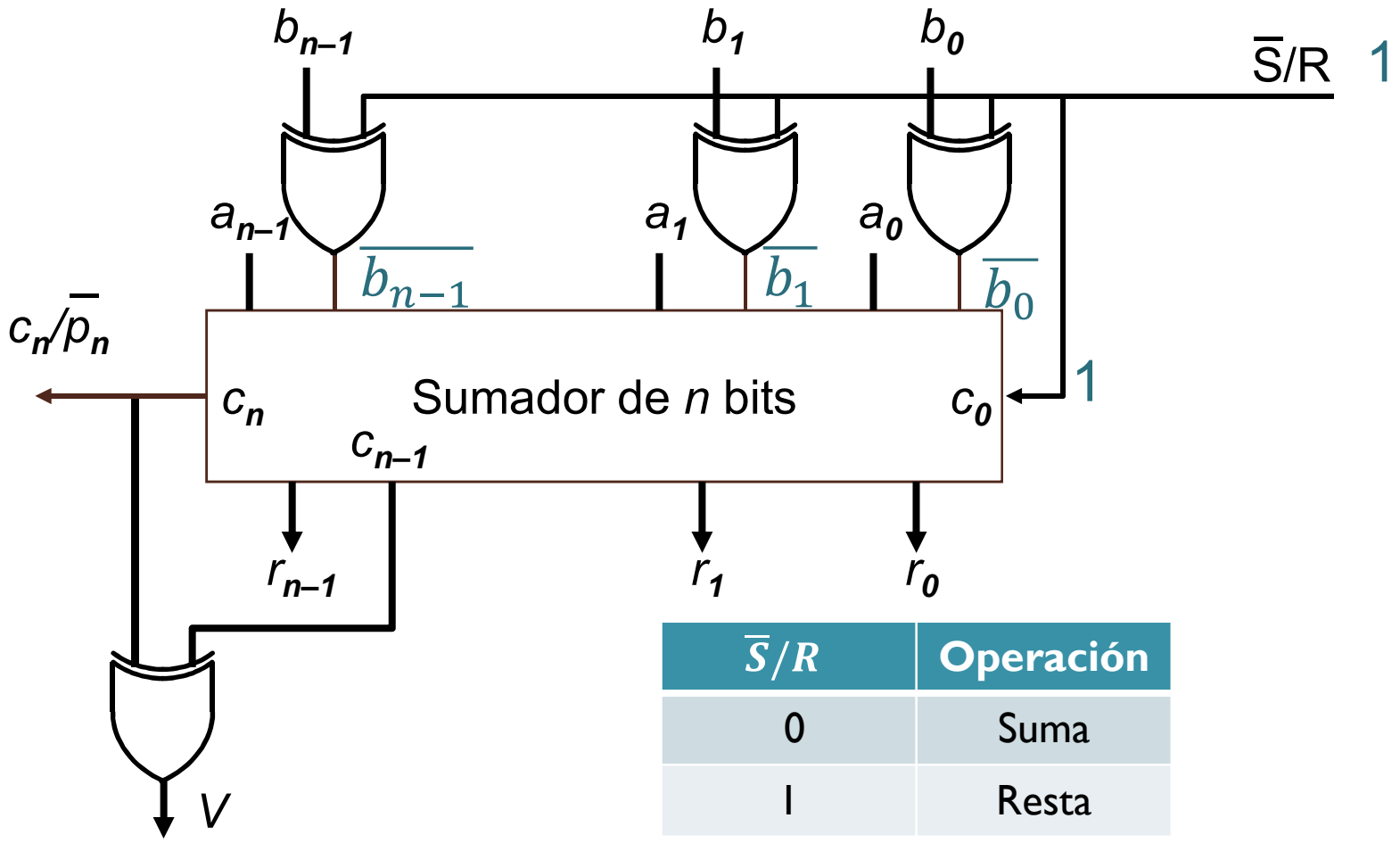
- Diseño clásico del sumador/restador



$\overline{S}/R$	Operación
0	Suma
1	Resta

Operadores de resta y comparación

- Diseño clásico del sumador/restador



$\overline{S}/R$	Operación
0	Suma
1	Resta

Índice

Operadores de suma

- El sumador serie de *n* bits. Complejidad
  - ✓ El tiempo de cálculo de un sumador serie para operandos de *n* bits se puede expresar en términos de los retardos de un sumador completo:

    $t(n\ bits) = (n-1) \cdot t_C + máx\{t_C, t_S\}$

  - ✓ Asintóticamente el tiempo de cálculo es lineal, $t(n\ bits) = O(n)$
  - ✓ El coste espacial también es lineal, $coste(n\ bits) = O(n)$
  - ✓ La suma serie es poco eficiente para las aplicaciones propias de un procesador convencional, con n=32 o n=64 bits

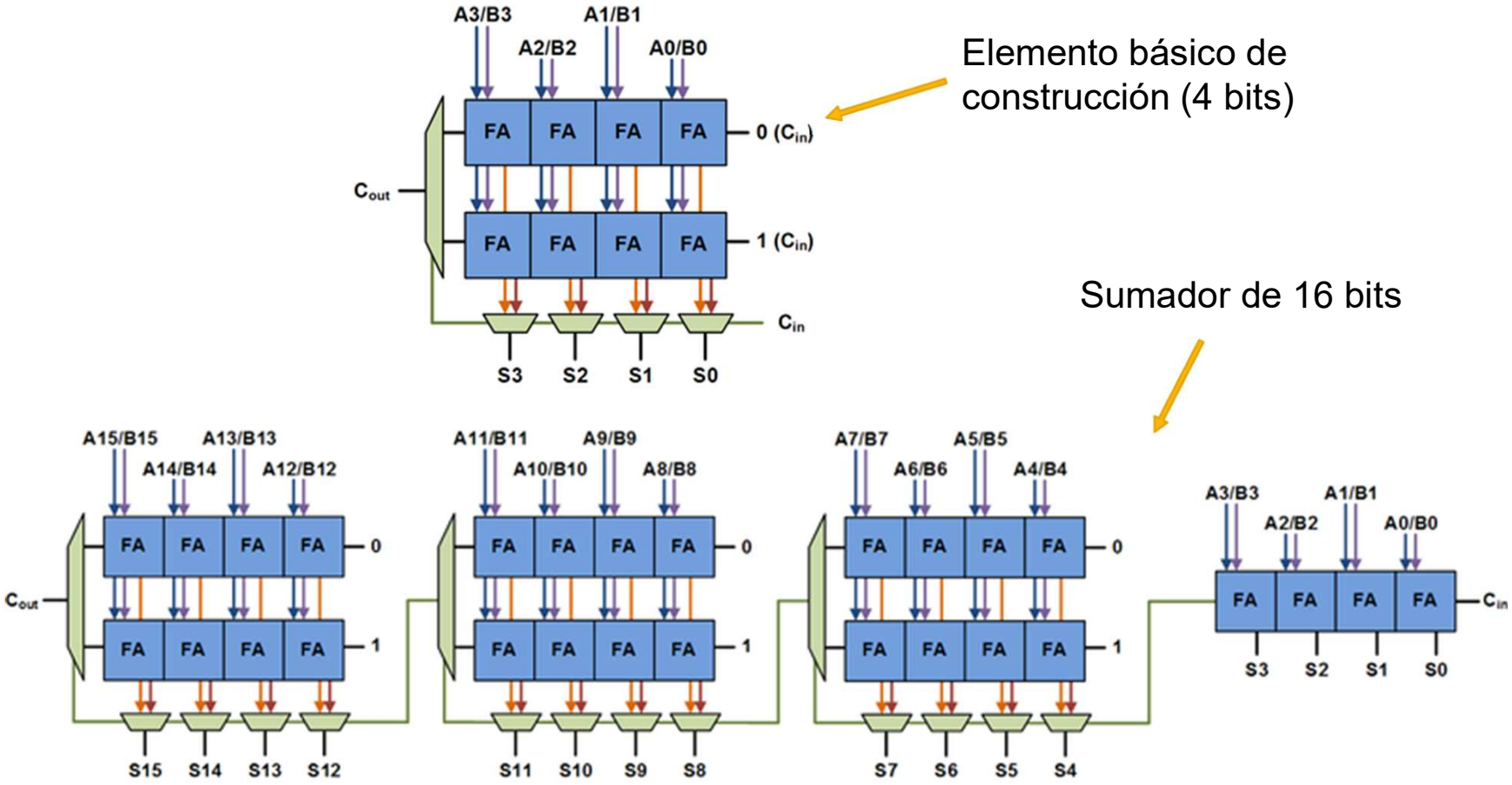
Índice

Operadores de suma

- Mejoras en la suma
  - ✓ Anticipación del acarreo: CLA, *Carry Lookahead Adder*
    - Calcula los bits de acarreo c_i antes que los bits de suma s_i
    - El retardo del cálculo de los acarreos es $O(\log(n))$
  - ✓ Selección del acarreo: CSA, *Carry Select Adder*
    - Divide la suma en dos partes y gestiona la parte alta con los posibles valores del acarreo proveniente de la parte baja

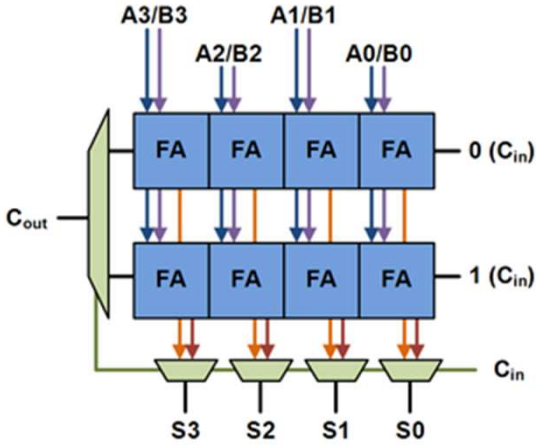
Operadores de suma

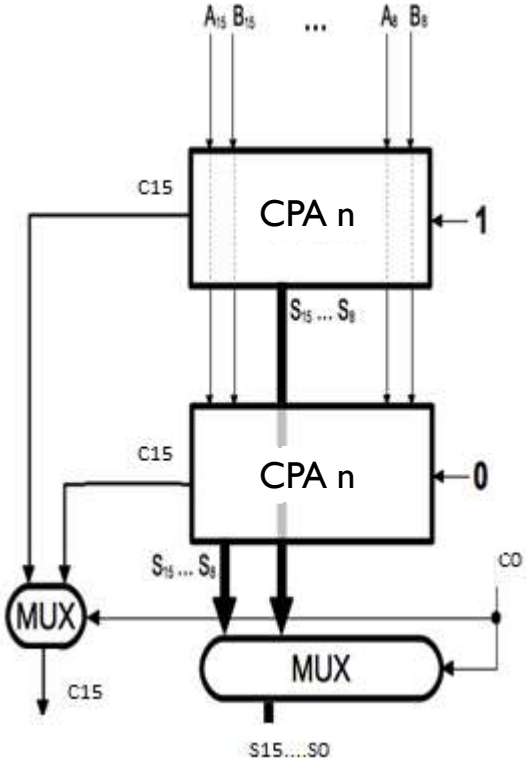
- CSA (Carry Select Adder)
  - ✓ Acelera la operación de suma a base de invertir más circuitos

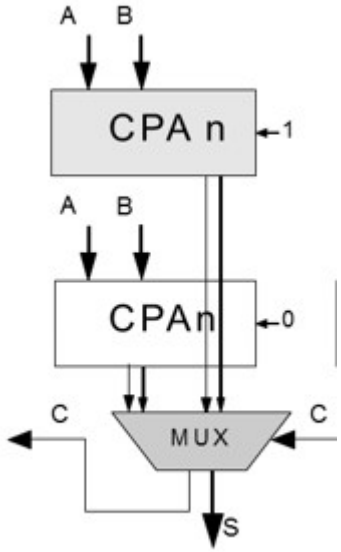


RESUMEN

- Sumador CSA: Varias formas de representación del mismo circuito
  - ✓ T_{op} = Entrada más lenta + Tiempo calculo operador





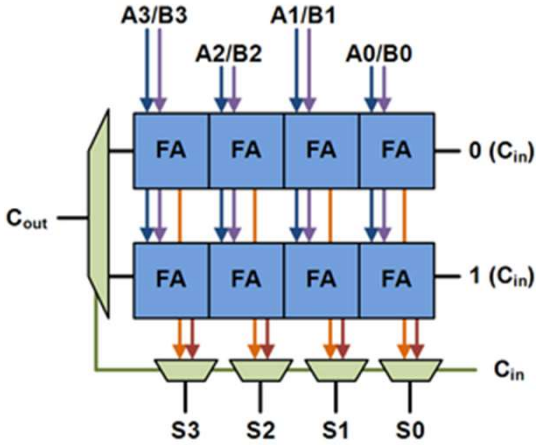


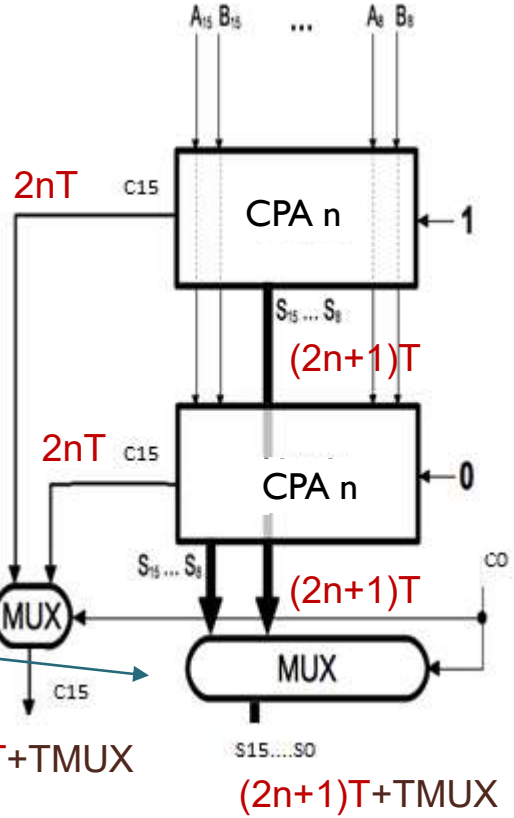
RESUMEN

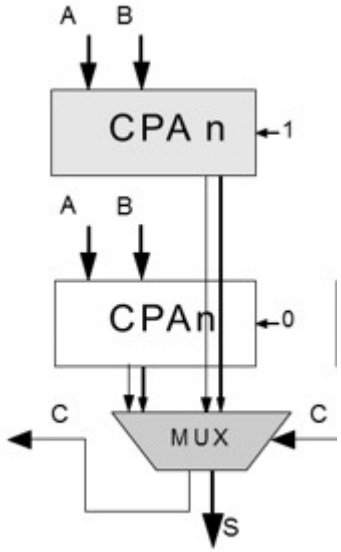
- Sumador CSA

✓ T_{op} = Entrada más lenta + Tiempo calculo operador

Entradas Mismo TIEMPO







TEntrada+ TMUX

El máximo:

Top = (2n+1)T+TMUX

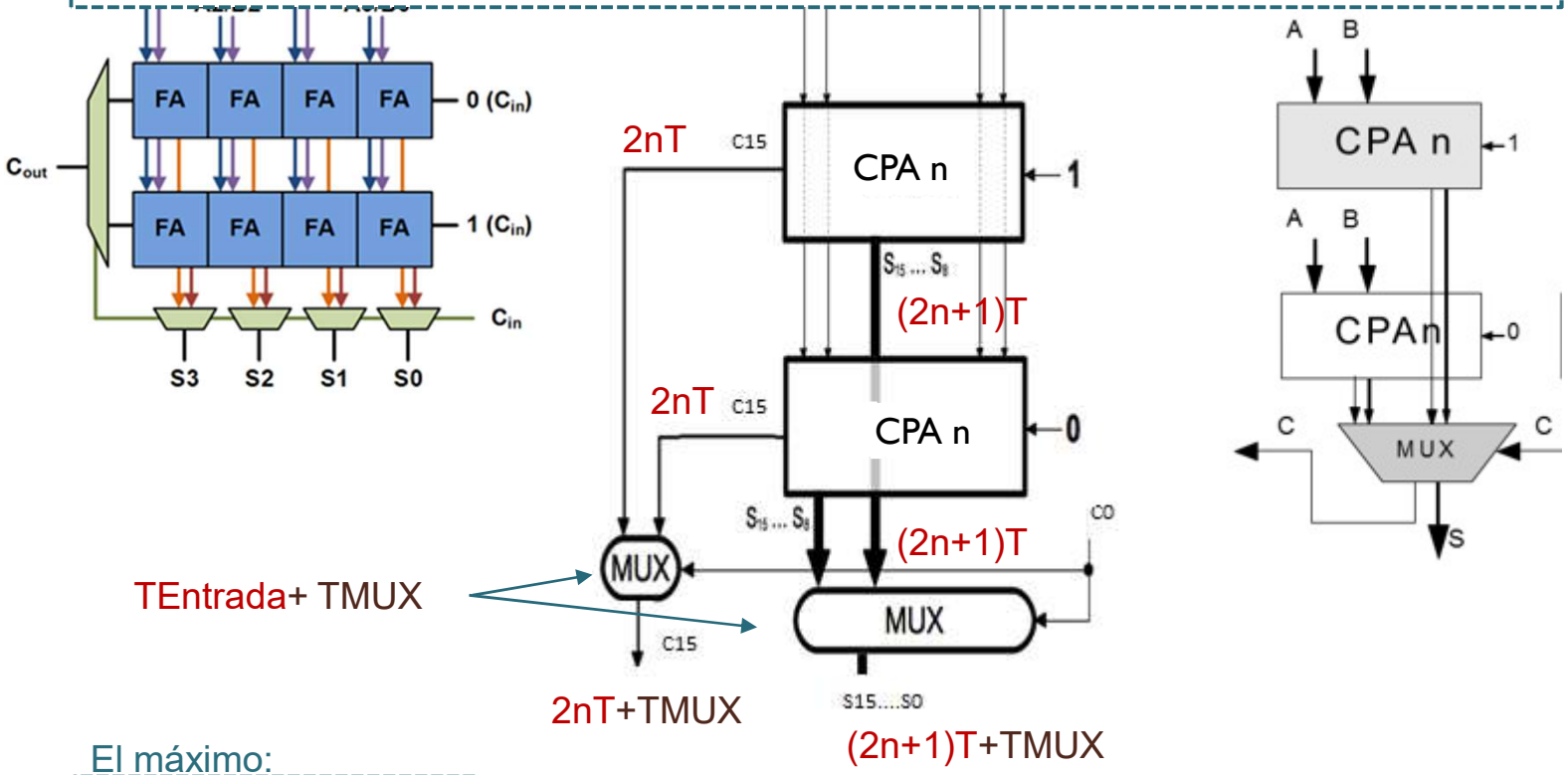
RESUMEN

- Sumador CSA

✓ T_{op} = Entrada más lenta + Tiempo calculo operador

Entradas Mismo TIEMPO

Si alguna entrada llega más tarde, pues en ese caso se suma ese tiempo



TEntrada+ TMUX

2nT+TMUX

(2n+1)T+TMUX

El máximo:

Top = (2n+1)T+TMUX

Índice

Fundamentos

Desplazamientos y aritmética entera (I)

- ✓ Desplazar *n* bits hacia la izquierda es equivalente a multiplicar por 2^n
  - Entran *n* ceros por la derecha
  - Operación válida para enteros con y sin signo
  - Nombre de la operación: desplazamiento **lógico** hacia a la izquierda

Java

```
int a = -12;
int b = a << 3;
System.out.println(a + " * 8 = " + b);
```

```
-12 * 8 = -96
```

Fundamentos

Desplazamientos y aritmética entera (II)

- ✓ Desplazar *n* bits hacia la derecha es equivalente a dividir por 2^n
  - Enteros sin signo: entran *n* ceros por la izquierda
  - Enteros con signo: el bit de signo se replica *n* veces

Sin signo: desplazamiento **lógico**		Con signo: desplazamiento **aritmético**	
0 1 1 0 0 1	25	0 1 1 0 0 1	+25
>> 2	/ 2^2	>> 2	/ 2^2
0 0 0 1 1 0	6	0 0 0 1 1 0	6
1 0 1 0 0 0	40	1 0 1 0 0 0	–24
>> 2	/ 2^2	>> 2	/ 2^2
0 0 1 0 1 0	10	1 1 1 0 1 0	–6

Fundamentos

- Los compiladores evitan las operaciones de multiplicación siempre que es posible

```
int a,b,c,d;
a = a*2;        // 2=2^1
b = b*8;        // 8=2^3
c = c*1024;     // 1024=2^10
d = d*5         // 5=2^2+1
```

```
lw $s0, a
lw $s1, b
lw $s2, c
lw $s3, d
add $s0, $s0, $s0
sll $s1, $s1, 3
sll $s2, $s2, 10
sll $t0, $s3, 2
add $s3, $s3, $t0
sw $s0, a
sw $s1, b
...
```

Fundamentos

- Anatomía de la multiplicación sin signo
  - ✓ En general, para representar el producto de dos números de *n* bits hacen falta 2*n* bits
  - ✓ El procedimiento *humano* se basa en sumas y desplazamientos

				1	1	0	1	(13_{10})
			×	1	0	0	1	(9_{10})
				1	1	0	1	
			0	0	0	0	0	
		0	0	0	0	0	0	
	1	1	0	1	0	0	0	
0	**1**	**1**	**1**	**0**	**1**	**0**	**1**	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Pesos

Notación
M = Multiplicando; m_i = bit i-ésimo
Q = Multiplicador; q_i = bit i-ésimo
P = Producto; p_i = bit i-ésimo

$$P = M \times Q = \sum_{i=0}^{n-1} M q_i 2^i$$

$(117_{10}) = 1101_2 \times (1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0)$

Fundamentos

- Problemática de la multiplicación con signo
  - ✓ Hay que extender el signo del multiplicando
  - ✓ El peso del bit de signo del multiplicador es de -2^{n-1} (y no 2^{n-1})
    - Conviene buscar codificaciones alternativas

1	1	1	1	1	1	0	1	(-3_{10})
			×	1	0	0	1	(-7_{10})
1	1	1	1	1	1	0	1	
			0	0	0	0	0	
		0	0	0	0	0	0	
	0	0	1	1	0	0	0	
0	**0**	**0**	**1**	**0**	**1**	**0**	**1**	$(+21_{10}) = 11111101_2 \times(-1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0)$
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Pesos

$$P = M \times Q = \sum_{i=0}^{n-2} Mq_i 2^i - Mq_{n-1}2^{n-1}$$

Fundamentos

La división

- ✓ Dados un dividendo y un divisor, produce dos resultados: el cociente y el resto (o módulo)
- ✓ Ejemplo sin signo: algoritmo humano de restas y desplazamientos
  - Si no cabe (✗): 0 al cociente
  - Si cabe (✓): restar y 1 al cociente

```
   1 1 0 1          | 0 1 0 1
 - 1 0 1            -------
 -----------          1 0
✓  0 0 1 1
✗- 0 1 0 1
 -----------
   0 0 1 1  Resto final
```

- ✓ Con signo: por convención, el resto tiene el mismo signo que el divisor

Fundamentos

- La multiplicación y la división en alto nivel
  - ✓ El módulo

Java

```
int x,y,z,t;
x = 13;
y = 5;
z = x/y;
t = x%y;
System.out.println(x + " = " + y + "*" + z + " + " + t);
```

```
13 = 5*2 + 3
```

Java,C

```
int x,y,z;
x = 0;
y = 1;
z = y/x;
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by
zero at …
```

Índice

Multiplicación y división en el MIPS

- Instrucciones de desplazamiento
  - ✓ Son de la forma "operación Rr, Ri, Long", donde Long puede ser una constante o un registro
  - ✓ El desplazamiento máximo es de 31 posiciones. Sólo cuentan los 5 bits de menor peso de Long

tipo	formato R	
izquierda	`sll rd,rt,inm`	`sllv rd,rs,rt`
derecha (lógico)	`srl rd,rt,inm`	`srlv rd,rs,rt`
derecha (aritmética)	`sra rd,rt,inm`	`srav rd,rs,rt`

Multiplicación y división en el MIPS

- Instrucciones de multiplicación y división generales
  - ✓ Dos registros especiales de 32 bits: HI y LO
    - Combinados forman un registro de 64 bits
  - ✓ Operaciones

    `mult $2, $3`: HI-LO ← $2*$3; Operandos con signo

    `multu $2, $3`: HI-LO ← $2*$3; Operandos positivos sin signo

    `div $2, $3`: LO ← $2/$3; HI ← $2 mod $3; Con signo

    `divu $2, $3`: LO ← $2/$3; HI ← $2 mod $3; Sin signo
  - ✓ Transferencia de resultados
    - **`mfhi $2`**: $2 ← HI
    - **`mflo $2`**: $2 ← LO

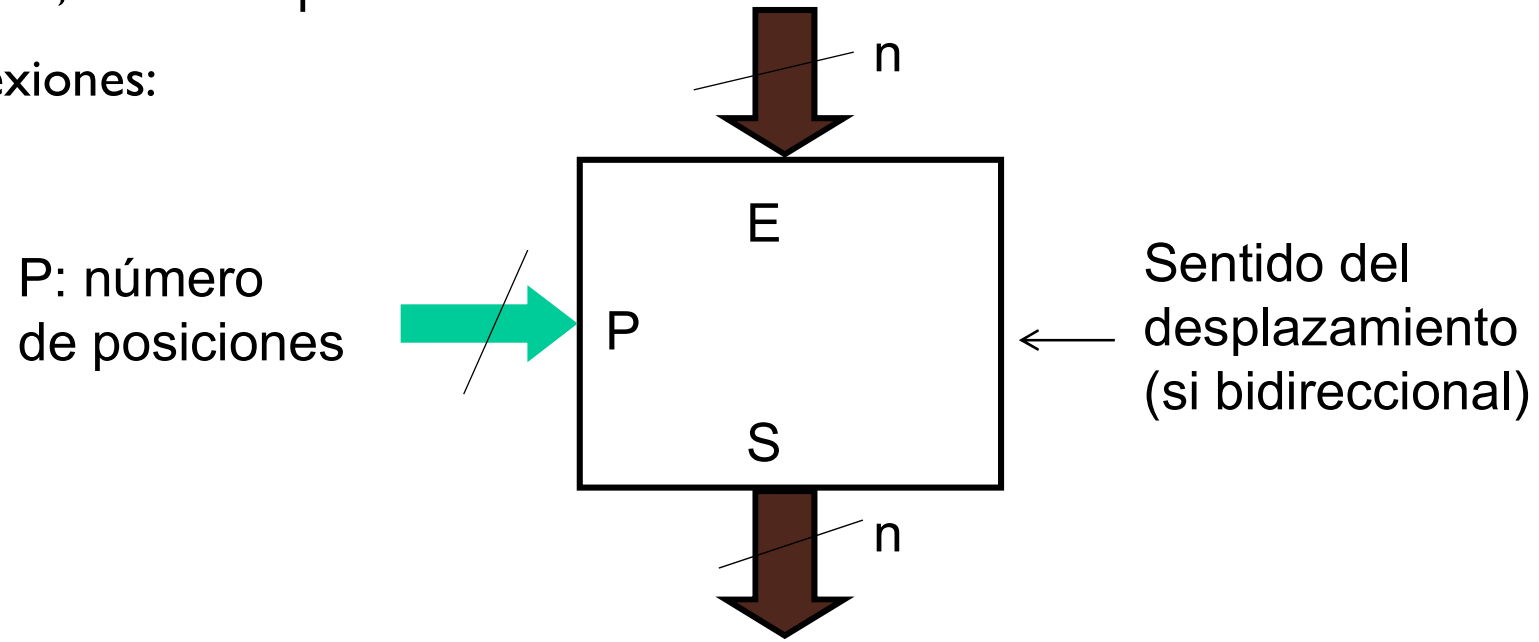
Multiplicación y división en el MIPS

- Instrucciones de multiplicación y división generales
  - ✓ Hay pseudoinstrucciones que permiten almacenar el resultado en un registro destinatario de propósito general y multiplicar por constantes
  - ✓ Ninguna de estas instrucciones comprueba desbordamientos o división por cero: hay que hacerlo por software

Índice

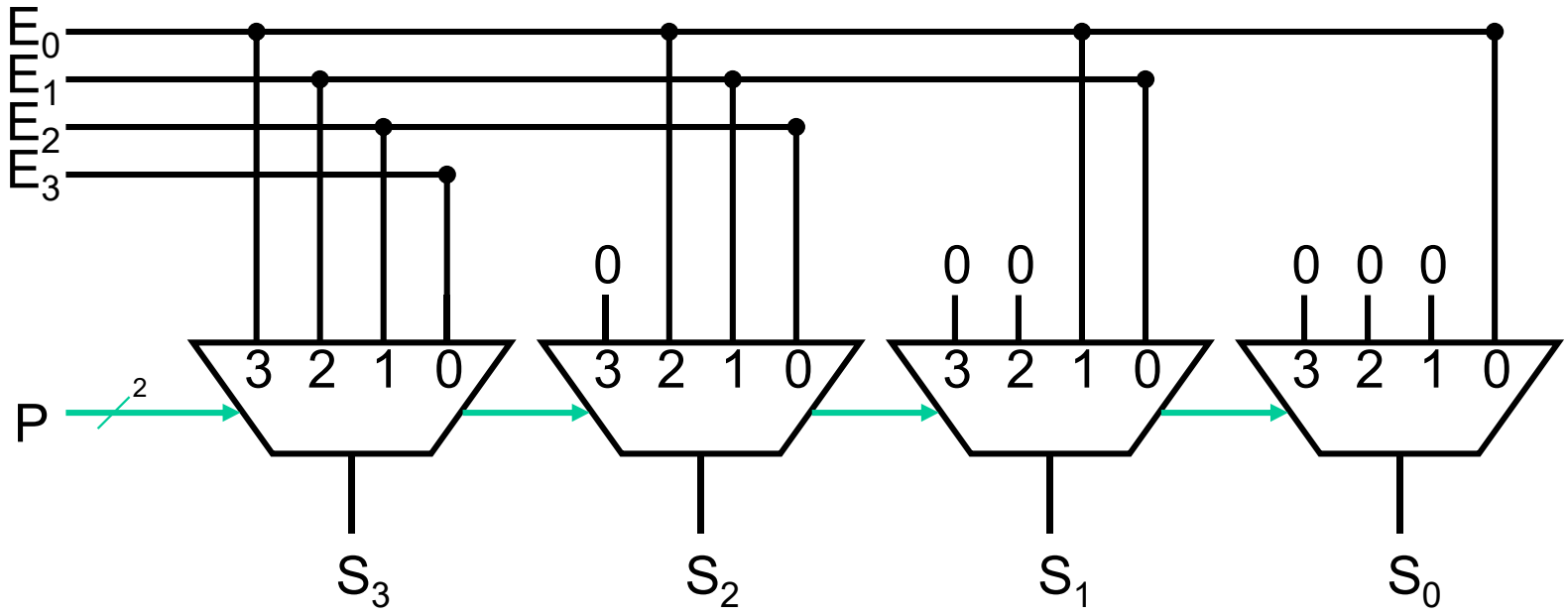
Operadores de desplazamiento

- El Barrel Shifter
  - ✓ Un Barrel Shifter es un circuito que permite realizar desplazamientos variables sobre datos de n bits
  - ✓ Puede implementarse mediante multiplexores
  - ✓ Dependiendo del diseño, hace desplazamientos lógicos o aritméticos hacia la derecha, hacia la izquierda o bidireccionales
  - ✓ Conexiones:



Operadores de desplazamiento

- Barrel shifter: ejemplo de diseño
  - ✓ Implementación de un operador de desplazamiento lógico hacia la izquierda (`sll`) para datos de 4 bits



Índice

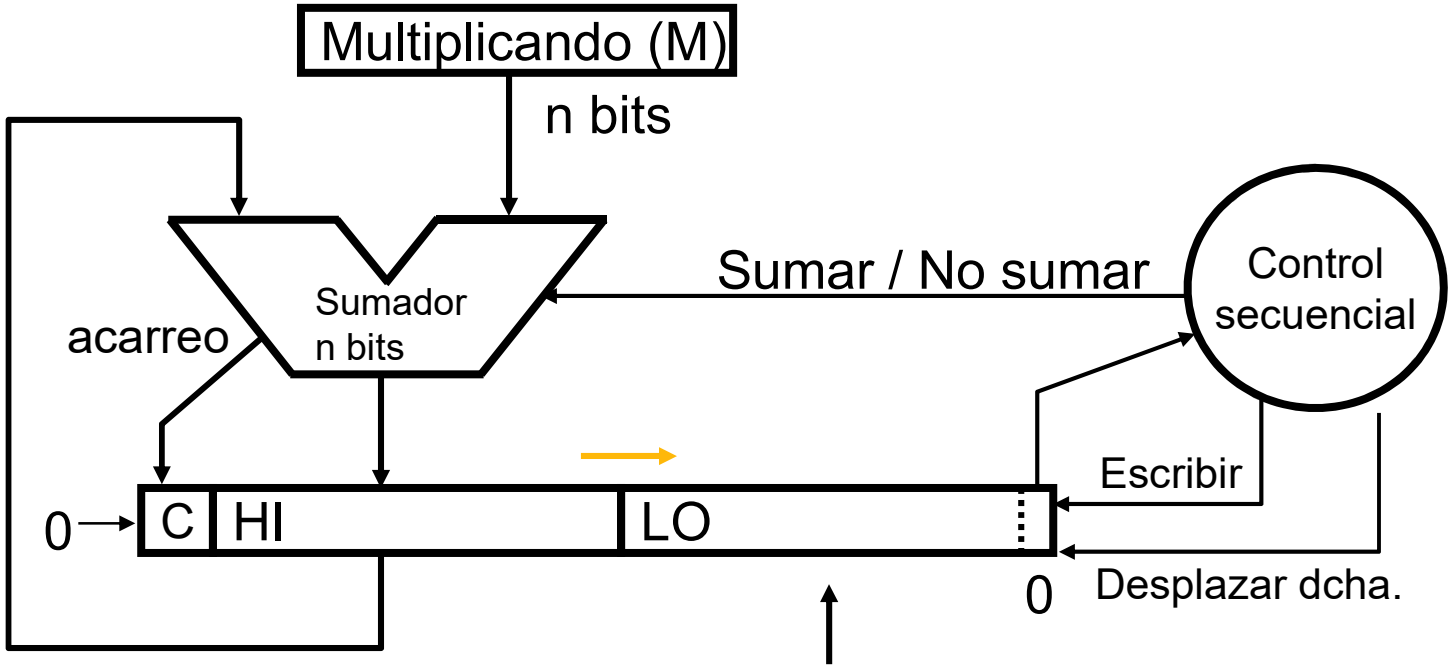
Índice

Operadores de multiplicación sin signo

- Operadores secuenciales aritméticos
  - ✓ Son circuitos secuenciales síncronos que hacen una operación dada
  - ✓ Necesitan un cierto número de ciclos de reloj para hacer la operación
  - ✓ El ciclo de reloj se ajusta para que puedan actuar los circuitos
  - ✓ Si un operador necesita n ciclos de t segundos para una operación,
    - el tiempo de operación será $T = n \times t$
    - la productividad será $P = f/n$, donde $f = 1/t$ es la frecuencia de trabajo del reloj
- Notación:
  - ✓ M = Multiplicando; m_i = bit i-ésimo de M
  - ✓ Q = Multiplicador; q_i = bit i-ésimo de Q
  - ✓ P = Producto; p_i = bit i-ésimo del producto
  - ✓ n = Número de dígitos de los operandos M y Q (de 0 a n–1)

Operadores de multiplicación sin signo

- Operador para el algoritmo de sumas y desplazamientos
  - ✓ M y Q de n bits; P de 2n bits

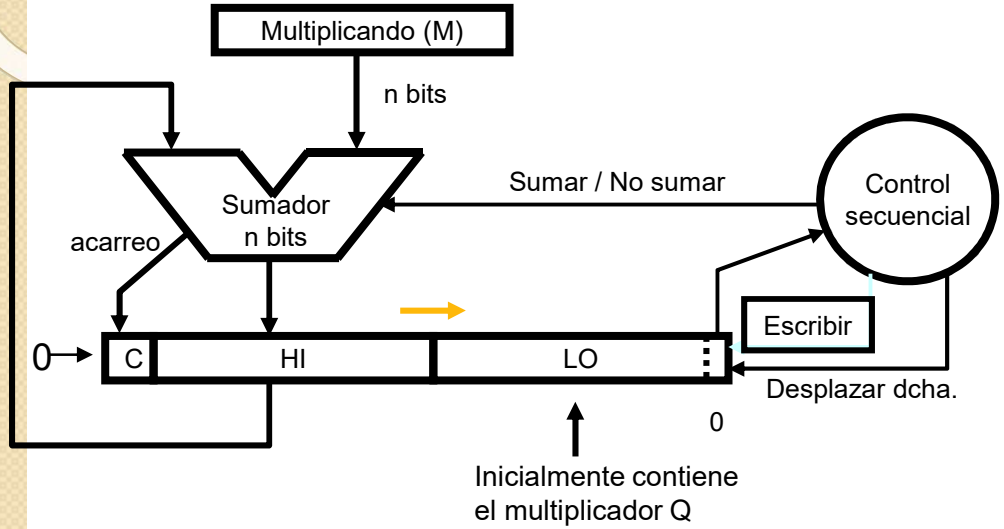


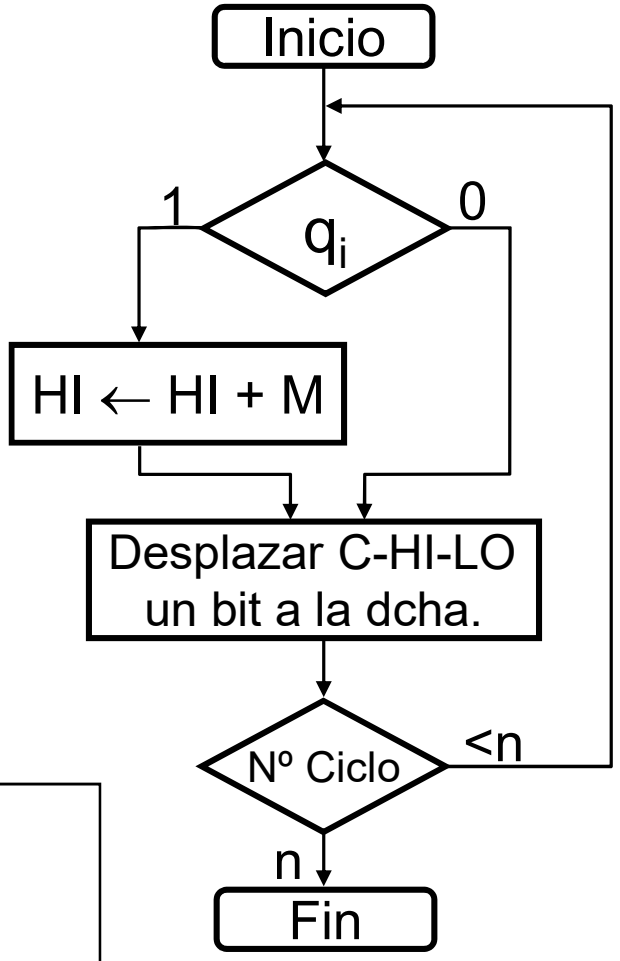
Inicialmente contiene el multiplicador Q

Al finalizar, HI y LO contienen el producto

Operadores de multiplicación sin signo

- Algoritmo con que funciona el operador secuencial





- El algoritmo requiere n ciclos
- En cada ciclo hay que hacer hasta una suma y un desplazamiento

Operadores de multiplicación sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1		
2		
3		
4		

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101

```
  0  0000   C HI
+    1011     M
-------------
     0 1011
```

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010
2	No sumar	0 0101 1010

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010
2	No sumar	0 0101 1010
	Desplazar C-HI-LO 1 bit a la derecha	0 0010 1101

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
		0 0000 0101
		0 1011 0101
	O 1 bit a la derecha	0 0101 1010
		0 0101 1010
	O 1 bit a la derecha	0 0010 1101
3	HI ← HI + M	0 1101 1101

```
  0  0010  C HI
+    1011     M
-------------
  0 1101
```

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010
2	No sumar	0 0101 1010
	Desplazar C-HI-LO 1 bit a la derecha	0 0010 1101
3	HI ← HI + M	0 1101 1101
	Desplazar C-HI-LO 1 bit a la derecha	0 0110 1110

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010
2	No sumar	0 0101 1010
	Desplazar C-HI-LO 1 bit a la derecha	0 0010 1101
3	HI ← HI + M	0 1101 1101
	Desplazar C-HI-LO 1 bit a la derecha	0 0110 1110
4	No sumar	0 0110 1110

Multiplicación de números enteros sin signo

- Ejemplo: n=4; M=1011_2; Q=0101_2 ($11_{10} \times 5_{10} = 55_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 0101
1	HI ← HI + M	0 1011 0101
	Desplazar C-HI-LO 1 bit a la derecha	0 0101 1010
2	No sumar	0 0101 1010
	Desplazar C-HI-LO 1 bit a la derecha	0 0010 1101
3	HI ← HI + M	0 1101 1101
	Desplazar C-HI-LO 1 bit a la derecha	0 0110 1110
4	No sumar	0 0110 1110
	Desplazar C-HI-LO 1 bit a la derecha	0 0011 0111

Multiplicación de números enteros sin signo

- Ejercicio: n=4; M=1101_2; Q=1011_2 ($13_{10} \times 11_{10} = 143_{10}$)

Ciclo	Acción	C-HI-LO
0	Valores iniciales	0 0000 1011
1		
2		
3		
4		

Solución: 1000 1111

Ejercicios boletín T3_EjerciciosMultiplicacionDivisión.pdf

PROBLEMA 15 del boletín "Tema 3. Problemas de artimética entera.pdf"

para practicar el algoritmo de multiplicación sin signo. Tiene varios ejemplos de multiplicación.

En general los ejercicios para practicar son infinitos, en decimal sabemos siempre el resultado de la operación, por tanto pasando el resultado a binario ya sabemos lo que tenemos que obtener.

Índice

Operadores de multiplicación con signo

- Tratamiento del signo por separado
  - ✓ Se trata de multiplicar los valores absolutos y considerar el signo aparte.

    Considerando que Signo(X) es el bit de signo de X:

```
Signo_Prod ← Signo(M) XOR Signo(Q);
si M < 0 entonces M ← M; fin si;
si Q < 0 entonces Q ← -Q; fin si;
P ← M × Q;
si Signo_Prod = 1 entonces P ← -P; fin si;
```

Operadores de multiplicación con signo

- Tratamiento del signo por separado

```
Signo_Prod ← Signo(M) XOR Signo(Q);
si M < 0 entonces M ← M; fin si;
si Q < 0 entonces Q ← -Q; fin si;
P ← M × Q;
si Signo_Prod = 1 entonces P ← -P; fin si;
```

- Inconvenientes
  - ✓ Requiere cierto hardware adicional para el caso particular de números con signo, a fin de complementar M, Q o P
  - ✓ Existen otros métodos para tratar uniformemente el producto de números con o sin signo (algoritmo de Booth, más adelante)

Operadores de multiplicación con signo

- Sumas y desplazamientos con extensión de signo
  - ✓ Puede funcionar con signo **sólo** si Q es positivo
  - ✓ Para ello, hay que extender el signo de los productos intermedios
  - ✓ Ejemplo: n = 4; M = –3; Q = 6; Representados en Ca2

				1	1	0	1	(-3_{10})
			×	0	1	1	0	(6_{10})
0	0	0	0	0	0	0	0	
1	1	1	1	1	0	1		
1	1	1	1	0	1			
0	0	0	0	0				
1	**1**	**1**	**0**	**1**	**1**	**1**	**0**	(-18_{10})

Para funcionar en cualquier caso, pueden procesarse los signos de M y Q de antemano:

```
si Q < 0 entonces
  Q ← -Q;
  M ← -M;
fin si
P ← M × Q;
```

Aunque es más sencillo que el anterior, también requiere procesar por separado los casos de multiplicación de números con o sin signo y complementar M y Q si Q<0

Operadores de multiplicación con signo

- Algoritmo de Booth
  - ✓ Consiste en recodificar el multiplicador como una suma de potencias positivas o negativas de la base: usa dígitos 0, +1 y –1

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática
  - ✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática
  - ✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

SUMO Y RESTO

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

SUMO Y RESTO

$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

REAGRUPO

$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

SUMO Y RESTO

$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

REAGRUPO

$-2^2 \times q_2 \times M + 2^2 \times q_1 \times M \qquad - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

SUMO Y RESTO

$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

FACTOR COMUN

$-2^2 \times q_2 \times M + 2^2 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

SUMO Y RESTO

$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

FACTOR COMUN

$2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática
  - ✓ $Q = (q_2\ q_1\ q_0)$ y $P = MXQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática
  - ✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

REAGRUPO

$$2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

REAGRUPO

$$2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^1 \times q_0 \times M - 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática
  - ✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

FACTOR COMUN

$$2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^1 \times q_0 \times M - 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MxQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

FACTOR COMUN

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M - 2^0 \times q_0 \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MXQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M - 2^0 \times q_0 \times M$$

si existiera $q_{-1} = 0$ siempre entonces:

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MXQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M - 2^0 \times q_0 \times M$$

si existiera $q_{-1} = 0$ siempre entonces:

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M + 2^0 \times (q_{-1} - q_0) \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = MXQ = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

$$-2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^1 \times q_1 \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$$

SUMO Y RESTO

$$-2^2 \times (q_1 - q_2) \times M - 2^1 \times q_1 \times M + 2^0 \times q_0 \times M + 2^0 \times q_0 \times M - 2^0 \times q_0 \times M$$

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M - 2^0 \times q_0 \times M$$

si existiera $q_{-1} = 0$ siempre entonces:

$$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M + 2^0 \times (q_{-1} - q_0) \times M$$

$$2^2 \times q_2' \times M + 2^1 \times q_1' \times M + 2^0 \times q_0' \times M$$

Operadores de multiplicación con signo

- Algoritmo de Booth: Justificación matemática

✓ $Q = (q_2\ q_1\ q_0)$ y $P = M \times Q = -2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

si existiera $q_{-1} = 0$ siempre entonces:

$2^2 \times (q_1 - q_2) \times M + 2^1 \times (q_0 - q_1) \times M + 2^0 \times (q_{-1} - q_0) \times M$

$2^2 \times q_2' \times M + 2^1 \times q_1' \times M + 2^0 \times q_0' \times M$

$P = M \times Q = M \times Q'$

$q_i' = q_{i-1} - q_i \in \{0, +1, -1\}$

q_i	q_{i-1}	Dígito Booth
0	0	0
0	1	+1
1	0	−1
1	1	0

Operadores de multiplicación con signo

- Recodificación del multiplicador por el método de Booth
  - ✓ Ejemplo: Obténgase el código Booth de 1110 0111 0011 (-397_{10})

    - Solución: 0 0 –1 0 +1 0 0 –1 0 +1 0 –1 =

$$-1\times2^0 + 1\times2^2 - 1\times2^4 + 1\times2^7 - 1\times2^9 =$$

$$-1 + 4 - 16 + 128 - 512 = -397$$

Operadores de multiplicación con signo

- Algoritmo de Booth
  - ✓ Consiste en recodificar el multiplicador como una suma de potencias positivas o negativas de la base: usa dígitos 0, +1 y –1
  - ✓ Por ejemplo:
    - El número 30 puede expresarse como (32 – 2)
    - $30_{10} = 0011110_2 = 0\ +1\ 0\ 0\ 0\ -1\ 0_{Booth} = -1 \cdot 2^1 + 1 \cdot 2^5$

Operadores de multiplicación con signo

- Algoritmo de Booth

✓ Consiste en recodificar el multiplicador como una suma de potencias positivas o negativas de la base: usa dígitos 0, +1 y –1

✓ Por ejemplo:

- El número 30 puede expresarse como (32 – 2)
- $30_{10} = 0011110_2 = 0\ +1\ 0\ 0\ 0\ -1\ 0_{Booth} = -1 \cdot 2^1 + 1 \cdot 2^5$

							0	1	0	1	1	0	1	(45_{10})
						×	0	+1	0	0	0	−1	0	(30_{Booth})
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	0	1	0	0	1	1		← Ca2 de M (restar M)
0	0	0	0	0	0	0	0	0	0	0	0			
0	0	0	0	0	0	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0					
0	0	0	1	0	1	1	0	1						← Copia de M (sumar M)
0	0	0	0	0	0	0	0							
0	0	0	1	0	1	0	1	0	0	0	1	1	0	(1350_{10})

Operadores de multiplicación con signo

- Algoritmo de Booth

✓ Funciona con números positivos o negativos:

- Multiplicación sin signo: suponer un bit de signo de M = 0
- Multiplicación con signo: extender el MSB de M como signo

✓ Ejemplo: Multiplicar con y sin signo los números 1101_2 y $0{+}10{-}1_{Booth}$

Sin signo

Signo positivo implícito →

				0	1	1	0	1	(13_{10})
				×	0	+1	0	−1	(3_{Booth})
1	1	1	1	0	0	1	1		
0	0	0	0	0	0	0			
0	0	1	1	0	1				
0	0	0	0	0					
0	0	1	0	0	1	1	1		(39_{10})

Con signo

Signo implícito →

				1	1	1	0	1	(-3_{10})
				×	0	+1	0	−1	(3_{Booth})
0	0	0	0	0	0	1	1		
0	0	0	0	0	0	0			
1	1	1	1	0	1				
0	0	0	0	0					
1	1	1	1	0	1	1	1		(-9_{10})

RESUMEN

- Multiplicación C.B.N: número naturales (sin signo)

$$P = MxQ = \sum_{i=0}^{n-1}(Mx\ 2^i x q_i) \quad donde\ q_i = \{0/1\}$$

Operador secuencial: Si qi=1 Entonces Sumar C:HI + M →C:HI

0→C →HI →LO

- Multiplicación C2: número enteros (con signo)

$$P = MxQ = -Mx\ 2^{n-1} x q_{n-1} + \sum_{i=0}^{n-2}\left(Mx\ 2^i x q_i\right) \quad donde\ q_i = \{0/1\}$$

$$P = MxQ' = \sum_{i=0}^{n-1}\left(Mx\ 2^i x q_i'\ \right) \quad donde\ \ q_i' = q_{i-1} - q_i = \{0/+1/-1\}$$

Operador secuencial: Si q_i'=+1 Entonces Sumar S:HI + S:M →S:HI

Si q_i'=-1 Entonces Restar S:HI – S:M →S:HI

S(Signo)→S →HI →LO

RESUMEN

- Multiplicación C.B.N: número naturales (sin signo)

$$P = MxQ = \sum_{i=0}^{n-1}(Mx\ 2^i x q_i) \quad donde\ q_i = \{0/1\}$$

Operador secuencial: Si qi=1 Entonces Sumar C:HI + M →C:HI

0→C →HI →LO

- Multiplicación C2: número enteros (con signo)

$$P = MxQ = -Mx\ 2^{n-1} x q_{n-1} + \sum_{i=0}^{n-2}\left(Mx\ 2^i x q_i\right) \quad donde\ q_i = \{0/1\}$$

$$P = MxQ' = \sum_{i=0}^{n-1}\left(Mx\ 2^i x q_i'\ \right) \quad donde\ \ q_i' = q_{i-1} - q_i = \{0/+1/-1\}$$

Operador secuencial: Si q_i'=+1 Entonces Sumar S:HI + S:M →S:HI

Si q_i'=-1 Entonces Sumar S:HI + C2(S:M) →S:HI

S(Signo)→S →HI →LO

RESUMEN

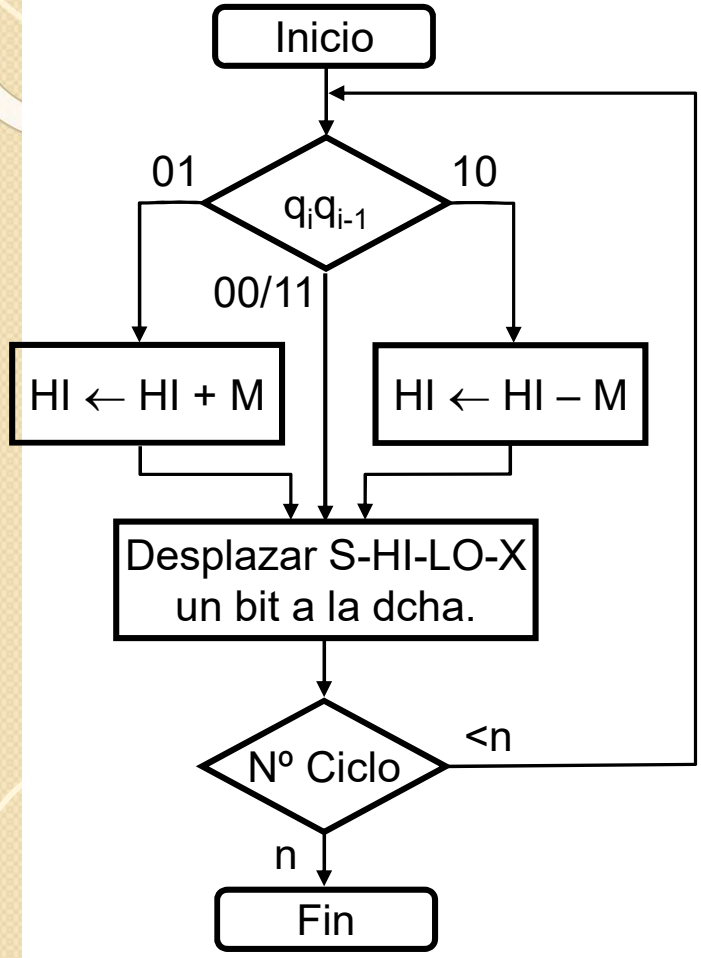
- Recodificación del multiplicador por el método de Booth
  - ✓ Deben considerarse parejas de bits correlativos, de dcha. a izda.
  - ✓ Debe suponerse un bit implícito = 0 a la derecha del LSB
  - ✓ Debe aplicarse la siguiente tabla de conversión:

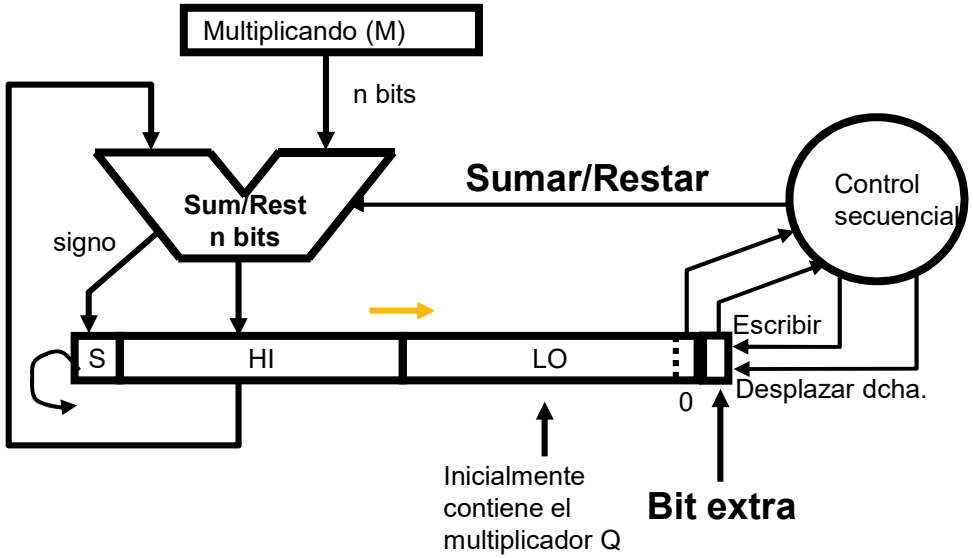
q_i	q_{i-1}	Dígito Booth
0	0	0
0	1	+1
1	0	−1
1	1	0

Para recordar:
Dígito Booth = $q_{i-1} - q_i$

Operadores de multiplicación con signo

- Modificación algoritmo 2 y operador secuencial para Booth

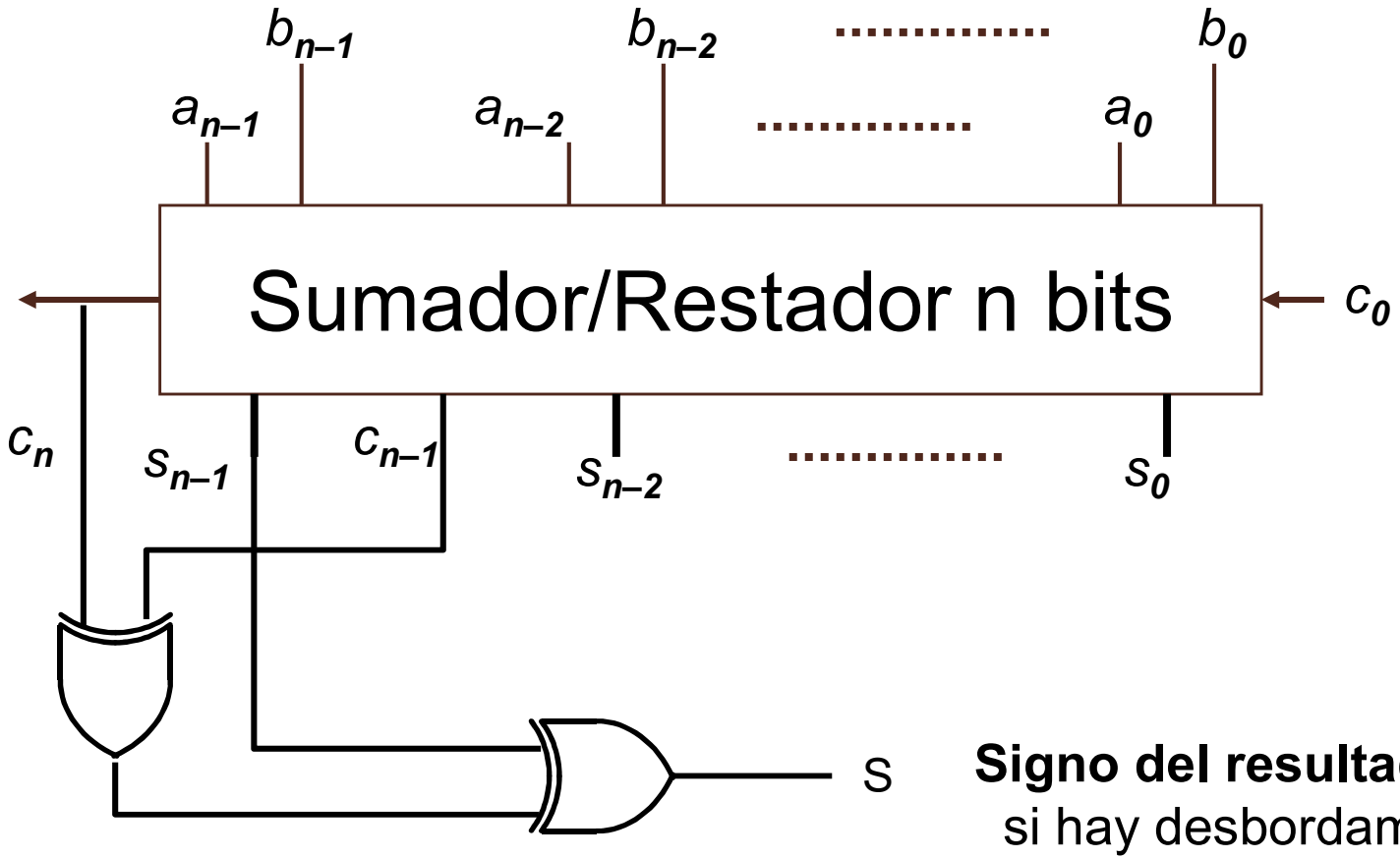




- El algoritmo requiere n ciclos
- En cada ciclo hay que hacer una suma o una resta más un desplazamiento

Operadores de multiplicación con signo

- Detalle del cálculo del bit de signo adicional



Signo del resultado:
si hay desbordamiento, invertir el MSB

Operadores de multiplicación con signo

- Ejercicio: n=4; M=1101_{C2}; Q=0110_{C2} $(-3_{10}\times 6_{10}=-18_{10})$

Cicle	Acció	S-HI-LO-X
0		
1		
2		
3		
4		

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=$1\text{-}1101_{C2}$; Q=0110_{C2} $(-3_{10}\times 6_{10}=-18_{10})$

  C2(M)= -M=0-0011

Bit extra

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1		
2		
3		
4		

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=1-1101; Q=0110 $(-3_{10} \times 6_{10} = -18_{10})$

  C2(M)= -M=0-0011

q' = 0 – 0 = 0 NADA

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 0011 0
2		
3		
4		

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=1-1101; Q=0110 $(-3_{10} \times 6_{10} = -18_{10})$

  C2(M)= -M=0-0011

q' = 0 – 1 = -1 RESTAR

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 0011 0
2	Caso 10: HI ← HI – M	0 0011 0011 0
	a la derecha	
3		
4		

```
  0 0000  S HI
+ 0 0011  S M
-------------
  0 0011
```

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=1-1101; Q=0110 $(-3_{10} \times 6_{10} = -18_{10})$

  C2(M)= -M=0-0011

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 0011 0
2	Caso 10: HI ← HI – M	0 0011 0011 0
	Desplazar S-HI-LO 1 bit a la derecha	0 0001 1001 1
3		
4		

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=$1\text{-}1101_2$; Q=0110_2 $(-3_{10}\times 6_{10}=-18_{10})$

  C2(M)= -M=0-0011

q' = 1 – 1 = 0 NADA

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 0011 0
2	Caso 10: HI ← HI – M	0 0011 0011 0
	Desplazar S-HI-LO 1 bit a la derecha	0 0001 1001 1
3	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 1100 1
4		

Solución: 1110 1110

Operadores de multiplicación con signo

- Ejercicio: n=4; M=1-1101; Q=0110 $(-3_{10} \times 6_{10} = -18_{10})$

  C2(M)= -M=0-0011

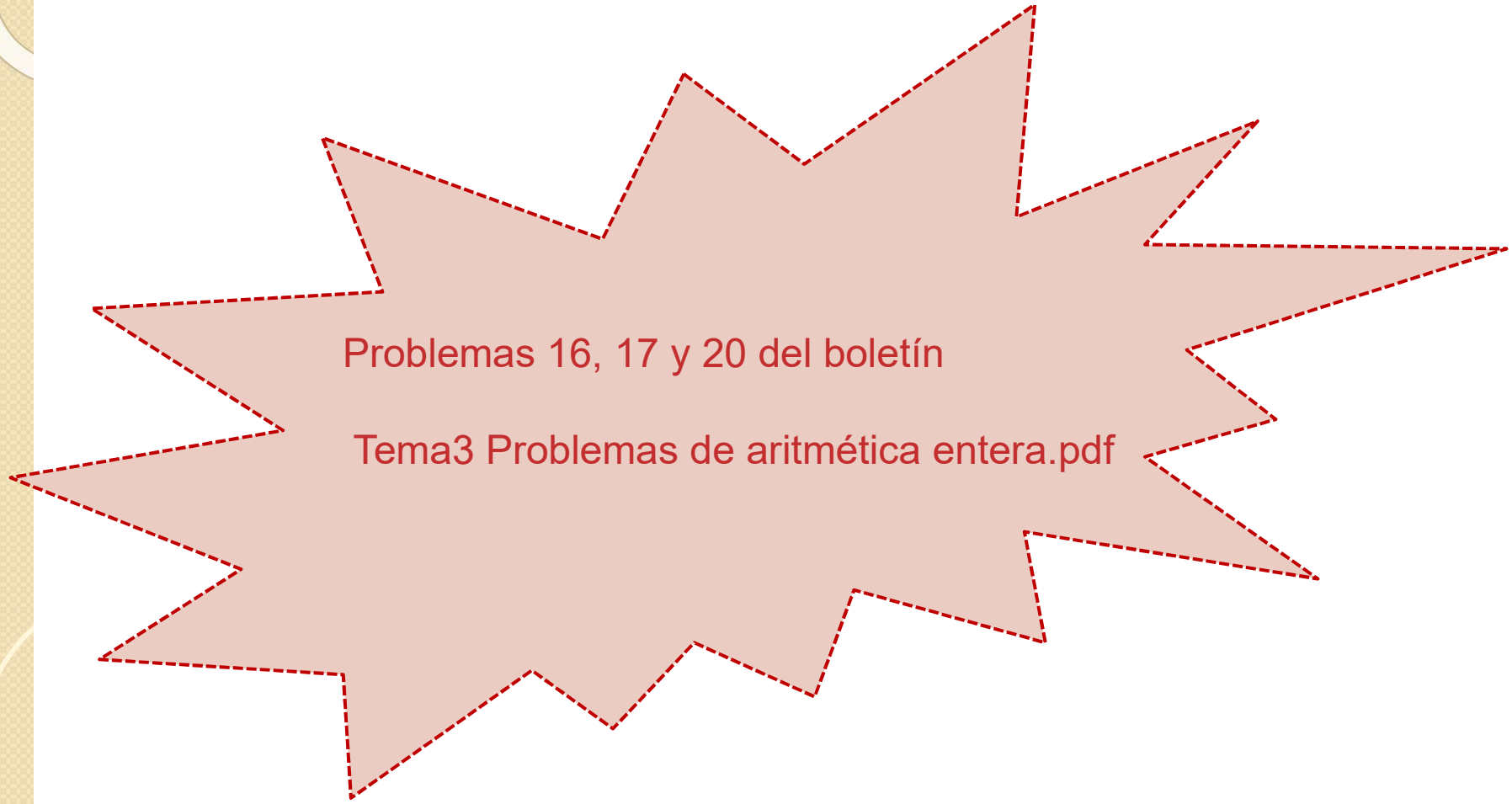
q' = 1 – 0 = +1 SUMAR

Cicle	Acció	S-HI-LO-X
0	Valores iniciales	0 0000 0110 0
1	Caso 00: No hacer nada	
	Desplazar S-HI-LO 1 bit a la derecha	0 0000 0011 0
2	Caso 10: HI ← HI – M	0 0011 0011 0
	a la derecha	0 0001 1001 1
3		
	a la derecha	0 0000 1100 1
4	Caso 10: HI ← HI + M	1 1101 1100 1
	Desplazar S-HI-LO 1 bit a la derecha	1 1110 1110 0

```
  0 0000  S HI
+ 1 1101  S M
-------------
  1 1101
```

Solución: 1110 1110

Más ejercicios para practicar en casa



Índice

Operadores de multiplicación con signo

- Algoritmo de recodificación por parejas de bits: Justificación matemática
  - ✓ $Q_{booth} = (q_3\ q_2\ q_1\ q_0)$ y tiene que ser un número par de bits
  - ✓ $P = M \times Q_{booth} = 2^3 \times q_3 \times M + 2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Operadores de multiplicación con signo

- Algoritmo de recodificación por parejas de bits: Justificación matemática

✓ $Q_{booth} = (q_3\ q_2\ q_1\ q_0)$ y tiene que ser un número par de bits

✓ $P = M \times Q_{booth} = 2^3 \times q_3 \times M + 2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Sacamos factor común por pares

$$= 2^2 \times (2 \times q_3 + q_2) \times M + 2^0 \times (2 \times q_1 + q_0) \times M$$

Operadores de multiplicación con signo

- Algoritmo de recodificación por parejas de bits: Justificación matemática

✓ $Q_{booth} = (q_3\ q_2\ q_1\ q_0)$ y tiene que ser un número par de bits

✓ $P = M \times Q_{booth} = 2^3 \times q_3 \times M + 2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Sacamos factor común por pares

$$= 2^2 \times (2 \times q_3 + q_2) \times M + 2^0 \times (2 \times q_1 + q_0) \times M$$

$$= 2^2 \times q_2'' \times M + 2^0 \times q_0'' \times M$$

Operadores de multiplicación con signo

- Algoritmo de recodificación por parejas de bits: Justificación matemática

✓ $Q_{booth} = (q_3\ q_2\ q_1\ q_0)$ y tiene que ser un número par de bits

✓ $P = M \times Q_{booth} = 2^3 \times q_3 \times M + 2^2 \times q_2 \times M + 2^1 \times q_1 \times M + 2^0 \times q_0 \times M$

Sacamos factor común por pares

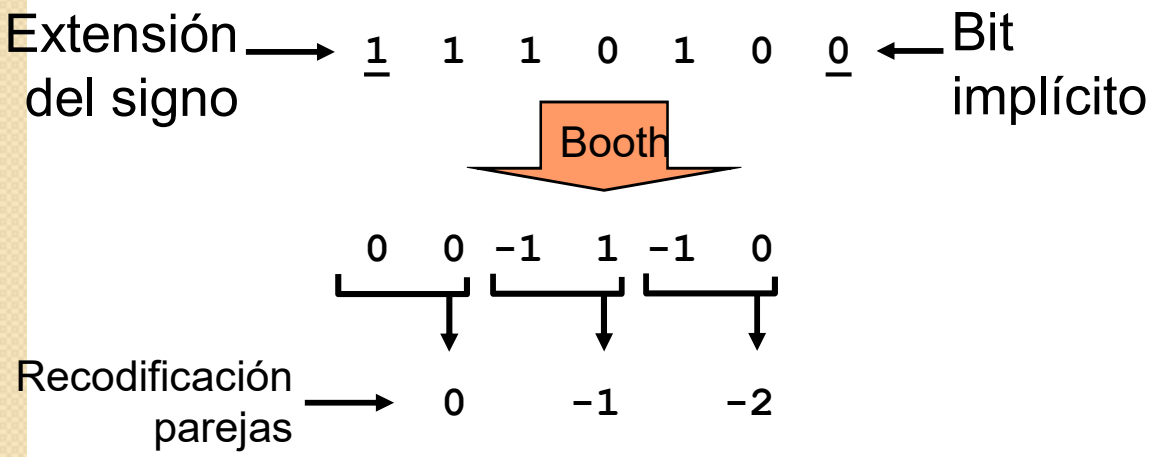
$$= 2^2 \times (2 \times q_3 + q_2) \times M + 2^0 \times (2 \times q_1 + q_0) \times M$$

$$= 2^2 \times q_2'' \times M + 2^0 \times q_0'' \times M$$

✓ $P = M \times Q_{pares}$ $\qquad q_i'' = 2 \times q_{i+1} + q_i$,, $i = 0, 2, 4, \ldots$

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas
  - ✓ n = 5; M = 01101_2 (13_{10}); Q = 11010_2 (-6_{10})



					0	1	1	0	1	
				×	0		-1		-2	
Restar 2×M	1	1	1	1	1	0	0	1	1	0
Restar M	1	1	1	1	0	0	1	1		
Nada	0	0	0	0	0	0				
	1	1	1	0	1	1	0	0	1	0 (-78_{10})

Operadores de multiplicación con signo

- Recodificación por parejas de bits
  - ✓ Extensión de Booth que reduce a la mitad el número de dígitos de Q, reduciendo así el número de productos intermedios a sumar

			Booth		Parejas	
q_{i+1}	q_i	q_{i-1}	q'_{i+1}	q'_i	q''_i	Acción
0	0	0	0	0	0	Nada
0	0	1	0	1	1	Sumar M
0	1	0	1	-1	1	Sumar M
0	1	1	1	0	2	Sumar 2×M
1	0	0	-1	0	-2	Restar 2×M
1	0	1	-1	1	-1	Restar M
1	1	0	0	-1	-1	Restar M
1	1	1	0	0	0	Nada

En cada iteración se desplaza S-HI-LO dos posiciones a la derecha

RESUMEN:Multiplicación C2(con signo)

- Algoritmo de BOOTH:

$P = MxQ' = \sum_{i=0}^{n-1}(Mx\ 2^i x q_i'\ )$ $donde$ $q_i' = q_{i-1} - q_i = \{0/+1/-1\}$

Operador secuencial: Si q_i'=+1 Entonces Sumar S:HI + S:M →S:HI

n+1 ciclos Si q_i'=-1 Entonces Sumar S:HI + C2(S:M) →S:HI

S(Signo)→S →HI →LO

- Recodificación por pares de bits (Solo si n es par)

$P = MxQ' = MxQ'' = \sum_{i=0,2}^{n-2}(Mx\ 2^i x q_i''\ )$ $donde$ $q_i'' = 2q'_{i+1} + q'_i =$

$q_i'' = \{0/+1/-1/-2/+2\}$

Operador secuencial: Si q_i''=+1 Entonces Sumar S:HI + S:M →S:HI

n/2 +1 ciclos Si q_i''=-1 Entonces Sumar S:HI + C2(S:M) →S:HI

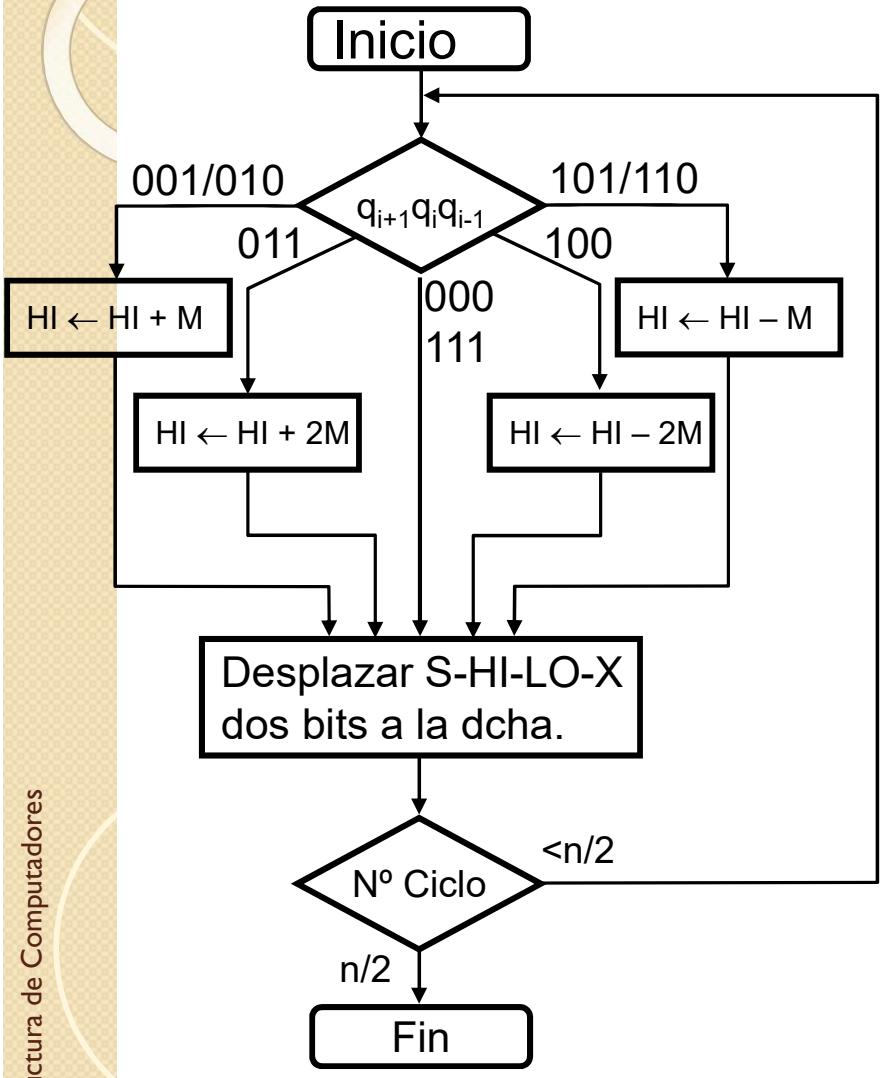
Si q_i''=-2 Entonces Sumar S:HI + C2(S:2M) →S:HI

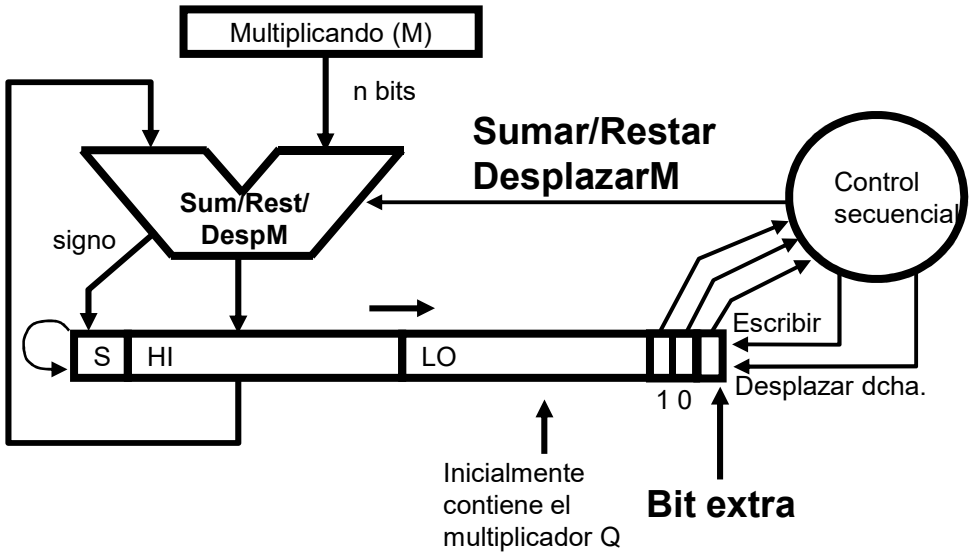
Si q_i''=+2 Entonces Sumar S:HI + S:2M →S:HI

S(Signo)→S →HI →LO 2 veces

Operadores de multiplicación con signo

- Modificación algoritmo 2 y operador para parejas de bits





Los registros HI y LO deben tener un número par de bits

- El algoritmo requiere n/2 ciclos
- En cada ciclo hay que hacer hasta una suma o resta y dos desplazamientos

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas
  - ✓ n=6 (n ha de ser par); M=001101_2 (13_{10}); Q=111010_2 (-6_{10})

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1		
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

+2M = 0-011010$_{C2}$ (26_{10})

✓ n=6 (n ha de ser par); M= 0-001101$_{C2}$ (13_{10}); Q=111010$_{C2}$ (-6_{10})

-M = 1-110011$_{C2}$ (-13_{10})

-2M = 1-100110$_{C2}$ (-26_{10})

Bit extra

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1		
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

✓ n=6 (n ha de ser par); M= $0\text{-}001101_{C2}$ (13_{10}); C

+2M = $0\text{-}011010_{C2}$ (26_{10})

-M = $1\text{-}110011_{C2}$ (-13_{10})

-2M = $1\text{-}100110_{C2}$ (-26_{10})

Qi' = 0 – 0 = 0

Qi+1' = 0 – 1 = -1

2Qi+1' + Qi' = -2 RESTAR 2M

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

✓ n=6 (n ha de ser par); M= $0\text{-}001101_{C2}$ (13_{10}); C

+2M = $0\text{-}011010_{C2}$ (26_{10})

-M = $1\text{-}110011_{C2}$ (-13_{10})

-2M = $1\text{-}100110_{C2}$ (-26_{10})

Qi' = 0 – 0 = 0

Qi+1' = 0 – 1 = -1

2Qi+1' + Qi' = -2 RESTAR 2M

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	0 000000 S HI + 1 100110 -2M 1 100110	
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

+2M = 0-011010$_{C2}$ (26_{10})

✓ n=6 (n ha de ser par); M= 0-001101$_{C2}$ (13_{10}); Q=111010$_{C2}$ (-6_{10})

-M = 1-110011$_{C2}$ (-13_{10})

-2M = 1-100110$_{C2}$ (-26_{10})

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

✓ n=6 (n ha de ser par); M= $0\text{-}001101_{C2}$ (13_{10}); C

+2M = $0\text{-}011010_{C2}$ (26_{10})

-M = $1\text{-}110011_{C2}$ (-13_{10})

-2M = $1\text{-}100110_{C2}$ (-26_{10})

Qi' = 1 – 0 = 1

Qi+1' = 0 – 1 = -1

2Qi+1' + Qi' = -2+1=-1 RESTAR M

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2		
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

✓ n=6 (n ha de ser par); M= $0\text{-}001101_{C2}$ (13_{10}); C

$+2M = 0\text{-}011010_{C2}$ (26_{10})

$-M = 1\text{-}110011_{C2}$ (-13_{10})

$-2M = 1\text{-}100110_{C2}$ (-26_{10})

Qi' = 1 – 0 = 1

Qi+1' = 0 – 1 = -1

2Qi+1' + Qi' = -2+1=-1 RESTAR M

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2	Caso 101: HI ← HI – M	1 101100 101110 1
3		

```
  1 111001  S HI
+ 1 110011  -M
-------------
  1 101100
```

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

+2M = 0-011010$_{C2}$ (26_{10})

✓ n=6 (n ha de ser par); M= 0-001101$_{C2}$ (13_{10}); Q=111010$_{C2}$ (-6_{10})

-M = 1-110011$_{C2}$ (-13_{10})

-2M = 1-100110$_{C2}$ (-26_{10})

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2	Caso 101: HI ← HI – M	1 101100 101110 1
	Desplazar S-HI-LO 2 bits a la derecha	1 111011 001011 1
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

✓ n=6 (n ha de ser par); M= $0\text{-}001101_{C2}$ (13_{10}); C

+2M = $0\text{-}011010_{C2}$ (26_{10})

-M = $1\text{-}110011_{C2}$ (-13_{10})

-2M = $1\text{-}100110_{C2}$ (-26_{10})

Qi' = 1 – 1 = 0

Qi+1' = 1 – 1 = 0

2Qi+1' + Qi' = 0 NADA

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2	Caso 101: HI ← HI – M	1 101100 101110 1
	Desplazar S-HI-LO 2 bits a la derecha	1 111011 001011 1
3		

Operadores de multiplicación con signo

- Ejemplo de multiplicación con recodificación por parejas

+2M = 0-011010$_{C2}$ (26_{10})

✓ n=6 (n ha de ser par); M= 0-001101$_{C2}$ (13_{10}); Q=111010$_{C2}$ (-6_{10})

-M = 1-110011$_{C2}$ (-13_{10})

-2M = 1-100110$_{C2}$ (-26_{10})

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 111010 0
1	Caso 100: HI ← HI – 2M	1 100110 111010 0
	Desplazar S-HI-LO 2 bits a la derecha	1 111001 101110 1
2	Caso 101: HI ← HI – M	1 101100 101110 1
	Desplazar S-HI-LO 2 bits a la derecha	1 111011 001011 1
3	Caso 111: No hacer nada	1 111011 001011 1
	Desplazar S-HI-LO 2 bits a la derecha	1 111110 110010 1

Operadores de multiplicación con signo

- Ejercicio: n = 6; M = 101001_2 (-23_{10}); Q=001001_2 (9_{10})

($-23_{10} \times 9_{10} = -207_{10}$)

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 001001 0
1		
2		
3		

Solución: 111100 110001

Operadores de multiplicación con signo

+2M = 1-010010 (-46_{10})

- Ejercicio: n = 6; M = 1-101001 (-23_{10}); Q=001001 (9_{10})

-M = 0-010111 ($+23_{10}$)

-2M = 0-101110 ($+46_{10}$)

Ciclo	Acción	S-HI-LO-X
0	Valores iniciales	0 000000 001001 0
1		
2		
3		

Solución: 111100 110001

La multiplicación secuencial: resumen

- Implementación del operador:
  - ✓ La multiplicación puede implementarse mediante sumas y desplazamientos. En hardware sólo requiere:
    - registro de desplazamiento
    - sumador o sumador/restador, si es multiplicación con signo
    - circuito de control, si se utiliza un operador secuencial
- El método de Booth
  - ✓ Permite tratar de manera uniforme la multiplicación con o sin signo
- La recodificación del multiplicador

La multiplicación secuencial: resumen

- Implementación del operador:
  - ✓ La multiplicación puede implementarse mediante sumas y desplazamientos. En hardware sólo requiere:
- El método de Booth
  - ✓ Permite tratar de manera uniforme la multiplicación con o sin signo
- La recodificación del multiplicador
  - ✓ Permite reducir el número de dígitos del multiplicador y, por tanto, el número de iteraciones del operador
  - ✓ La recodificación por parejas de bits permite reducir a la mitad el número de ciclos requerido para una multiplicación secuencial
  - ✓ Otras recodificaciones permiten reducir aún más el número de iteraciones