PCS 3115 (PCS2215)

Sistemas Digitais I

Circuitos Combinatórios

Somadores e Subtratores

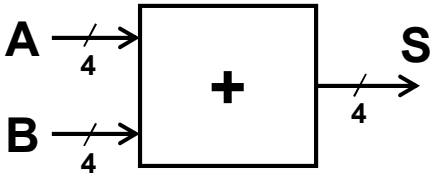
Prof. Dr. Marcos A. Simplicio Jr.

Atualização: Prof. Dr. Marco Túlio Carvalho de Andrade

versão: 5.0 (Abril/2020)

Somador

	3	2	1	0
A: 2	0	0	1	0
B: 6	0	1	1	0
S: A + B	1	0	0	0



- Como construir um somador binário?
  - **Abordagem 1**: método tradicional, via Mapa de Karnaugh (funciona sempre)

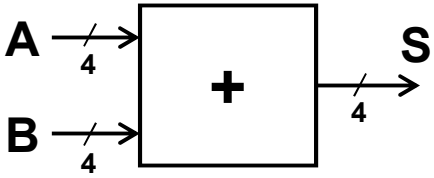
$2^8 = 256$ linhas!!!

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
...								...			
1	1	1	1	1	1	1	1	1	1	1	0

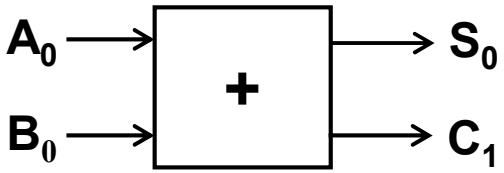
(overflow) ←

Somador

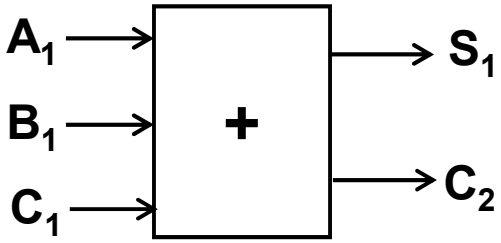
	3	2	1	0
A: 2	0	0	1	0
B: 6	0	1	1	0
S: A + B	1	0	0	0



- Como construir um somador binário?
  - **Abordagem 2**: projetar por faixas (fatias) e combinar



Faixa 0: meio somador



Faixa i > 0: somador completo

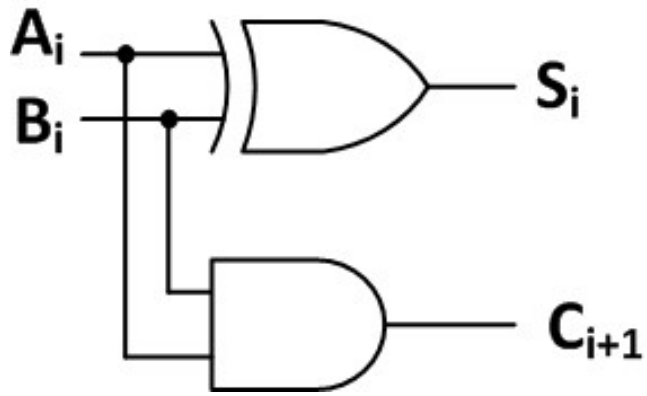
Meio Somador

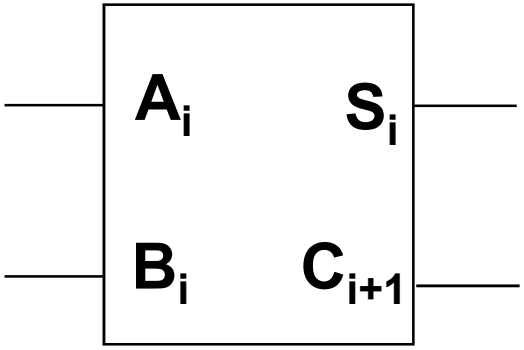
- A soma de dois operandos de 1 bit produz 2 bits
  1. Resultado da soma, e
  2. "Vai-um" (*carry*)
- Operação realizada por "meio somador"
  - Não considera "vem-um"
  - Logo, não funciona para números de 2+ bits
    - Teste: 0010 + 0110 ➔ sem o "vem-um": 0100 (errado...)
      ➔ com o "vem-um": 1000 (correto!!)

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

Meio Somador

- Qual o circuito que implementa o meio somador?





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

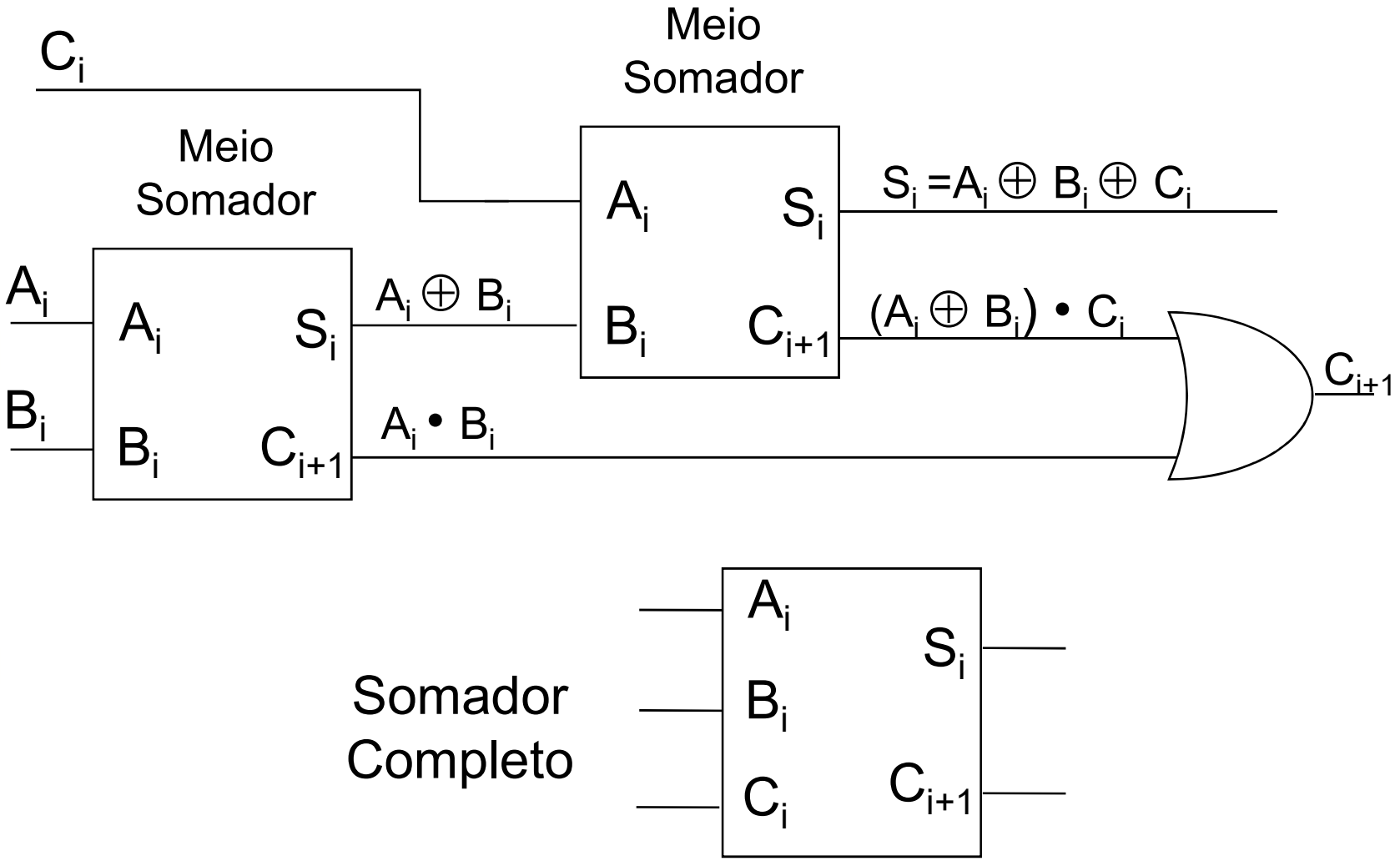
xor and

Somador completo

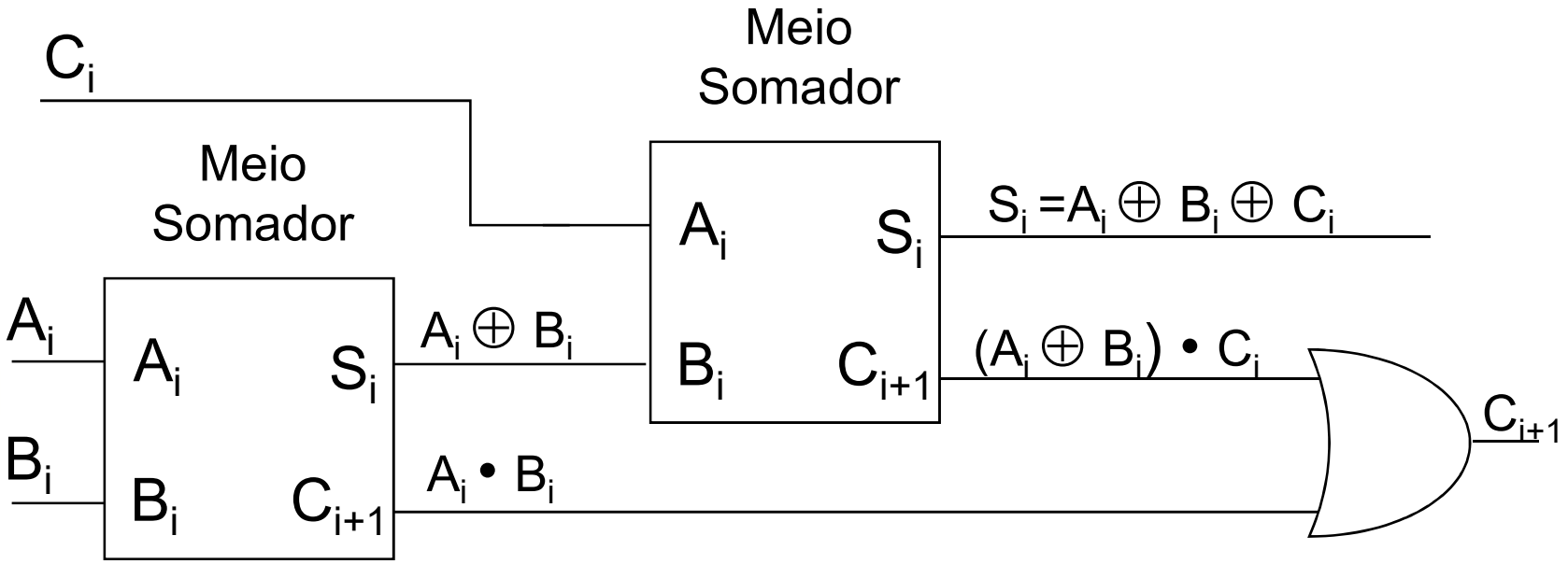
- Usado para soma de operandos com 2+ bits
  - Entrada adicional para tratar o "vem-um" do bloco anterior (entrada "C_i")
- Como implementar um somador completo?
  - Pode-se combinar 2 meio somadores: meia-soma entre A_i e B_i seguida de meia-soma com C_i.
  - "Vai-um" se qualquer das somas levar a "vai-um"

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

Somador completo



Somador completo



$$S_i = A_i \oplus B_i \oplus C_i = A_i'\bullet B_i'\bullet C_i + A_i'\bullet B_i\bullet C_i' + A_i\bullet B_i'\bullet C_i' + A_i\bullet B_i\bullet C_i$$

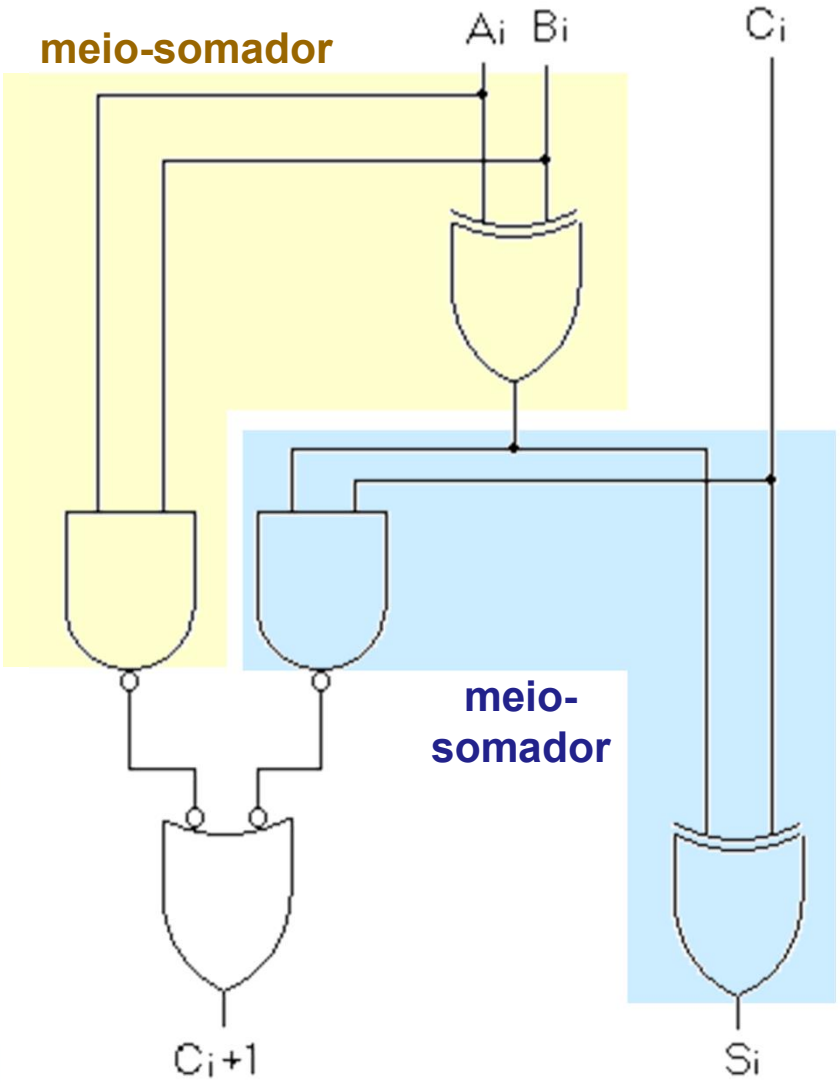
$$C_{i+1} = A_i'\bullet B_i\bullet C_i + A_i\bullet B_i'\bullet C_i + A_i\bullet B_i\bullet C_i' + A_i\bullet B_i\bullet C_i \quad \leftarrow \text{mintermos}$$

$$= (A_i \oplus B_i)\bullet C_i + A_i\bullet B_i \quad \leftarrow \text{meio-somadores}$$

$$= (A_i + B_i)\bullet C_i + A_i\bullet B_i \quad \leftarrow \text{ANDs e ORs}$$

$$= A_i\bullet C_i + B_i\bullet C_i + A_i\bullet B_i \quad \leftarrow \text{AND2}$$

Somador completo



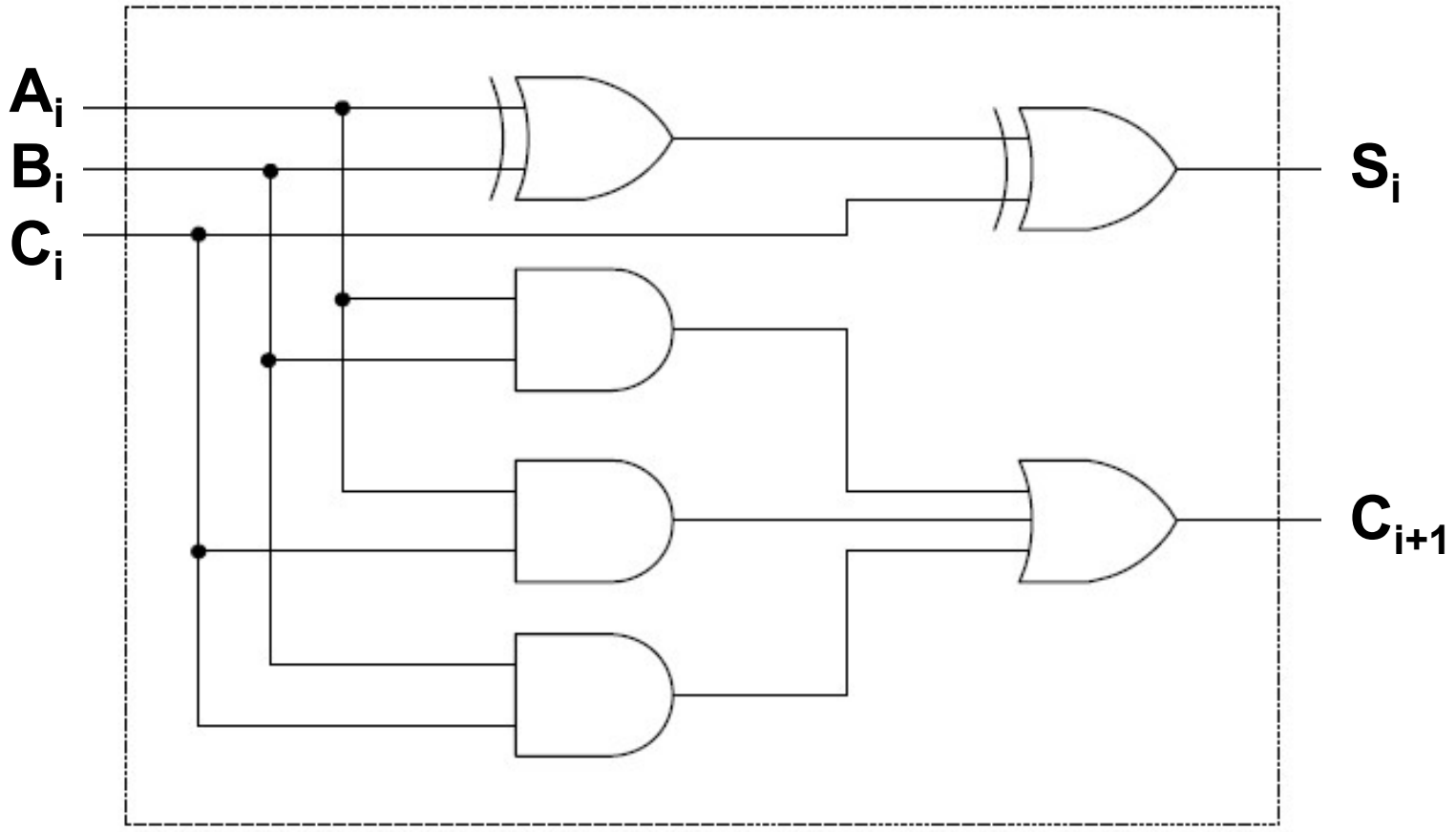
Combinação de meio-somadores:

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

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

➔ **Latência de C_{i+1}: 3**

Somador completo



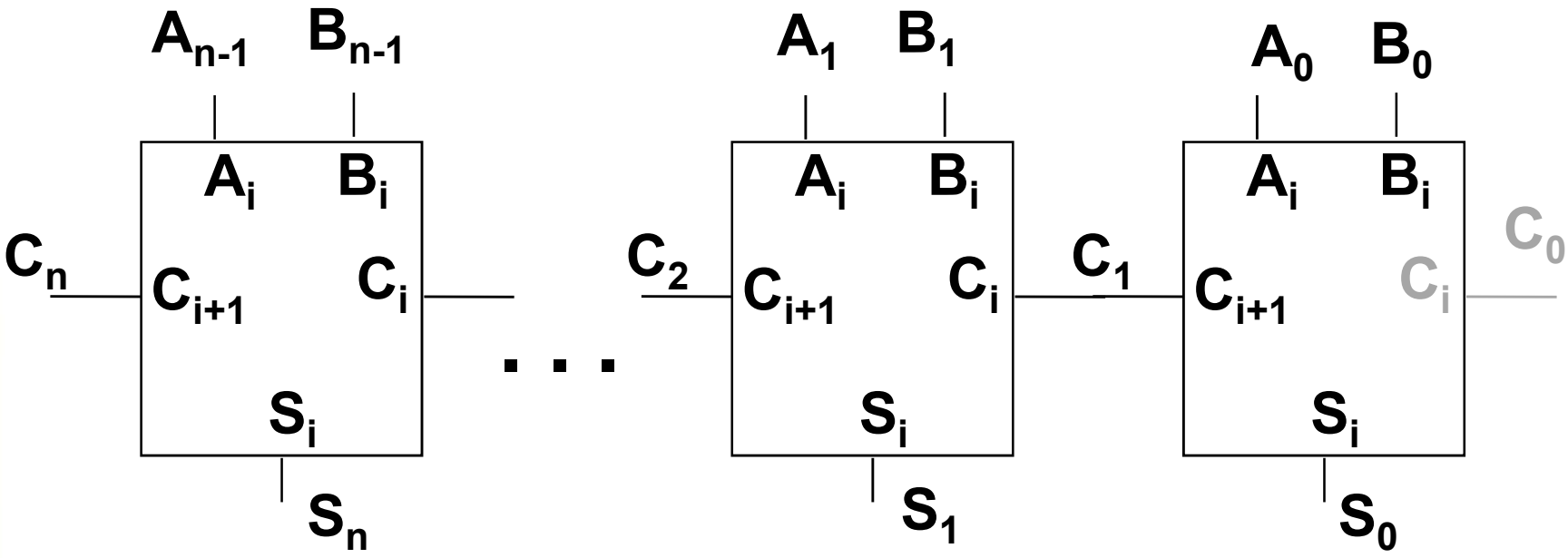
$S_i = A_i \oplus B_i \oplus C_i$; $C_{i+1} = A_i \cdot C_i + B_i \cdot C_i + A_i \cdot B_i$

➔ Latência de C_{i+1}: 2

Somador binário de n bits

- **Pergunta**: dado um somador completo de 1 bit, como obter um somador de n bits?
- **Resposta simples**: associação em série (cascateamento) de *n* somadores completos
- **Resultado**: "somadores com propagação de vai-um", ou *ripple adders*:
  - Cada somador completo trata um bit da soma
  - O "Vai-um" de um estágio se conecta ao "Vem-um" do estágio seguinte.
  - O "Vem-um" do estágio menos significativo costuma ficar em 0 (ou usa-se um meio-somador)

Somador binário de n bits



Somador binário de *n* bits com propagação de "vai-um"

➔ **Problema?**

Latência de propagação de "vai-um" do estágio menos até o mais significativo é proporcional a n…

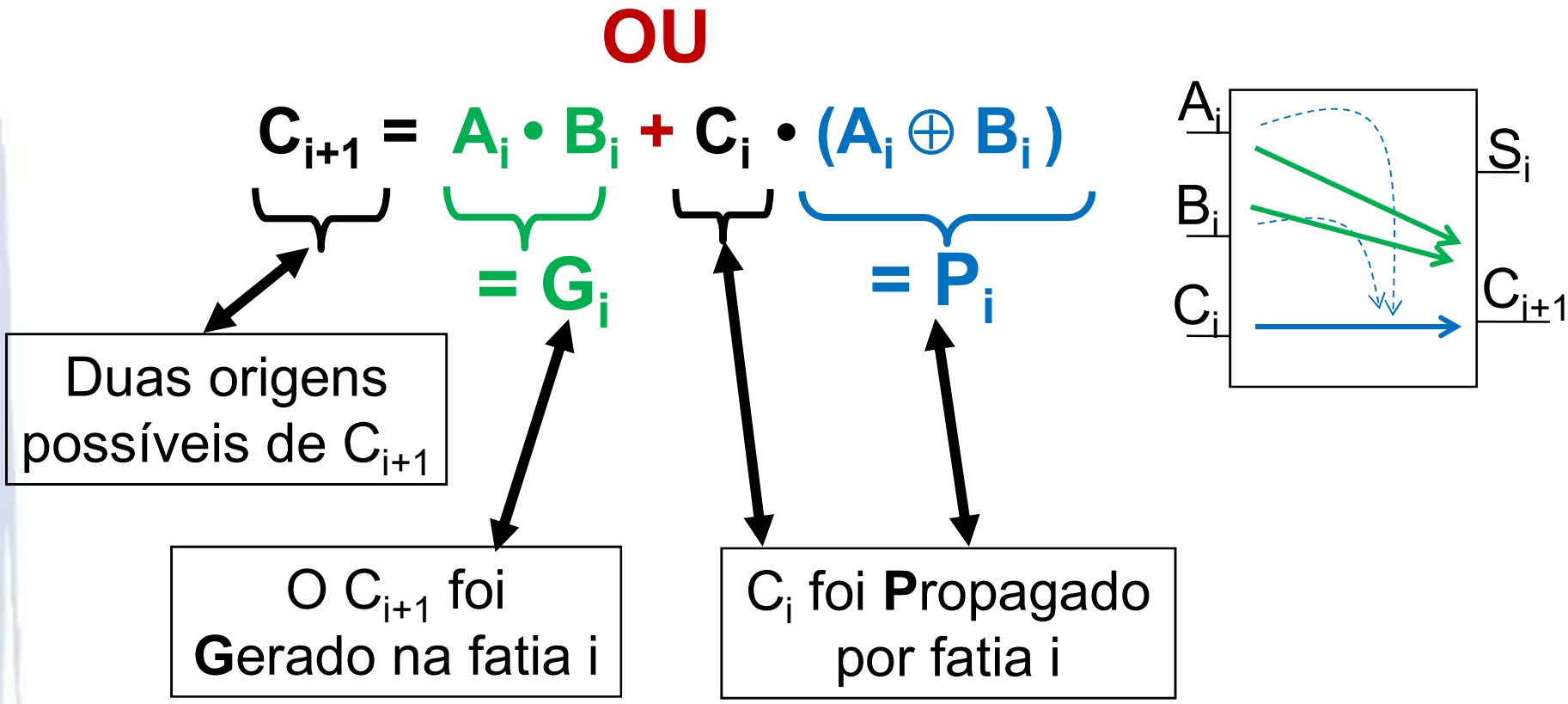
Somador binário de n bits

- **Pergunta**: como solucionar esse problema de latência?
- **Resposta**: montando circuito que calcula saída a partir das entradas diretamente
  - Ex.: Soma de produtos levaria a um atraso de apenas dois níveis (AND e OR)…
- **Resultado**: “somadores com antecipação de vai-um”, ou *carry look-ahead*

- Vamos tentar construir esse circuito...

Somador binário de n bits

- Antecipação de carry (*carry look-ahead*)



Somador binário de n bits

- Antecipação de carry (*carry look-ahead*): generalizando

$$C_1 = G_0 + P_0 \cdot C_0$$

$$(A_i \cdot B_i) \qquad (A_0 \oplus B_0)$$

- Prosseguindo com a generalização:

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$$

$$= \underbrace{G_1 + P_1 \cdot G_0}_{\text{geração}} + \underbrace{P_1 \cdot P_0}_{\text{propagação}} \cdot C_0$$

Somador binário de n bits

- Prosseguindo com a generalização:

$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0)$$

$$= \underbrace{G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot P_0 \cdot G_0 +}_{\text{Geração nas fatias 0, 1 ou 2}}$$

$$\underbrace{P_2 \cdot P_1 \cdot P_0}_{\text{Propagação nas fatias 0, 1 e 2}} \cdot C_0$$

Somador binário de n bits

- Prosseguindo com a generalização:

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

$$= \underbrace{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 +}_{\text{Geração nas fatias 1, 2 ou 3}}$$

$$\underbrace{P_3 \cdot P_2 \cdot P_1 \cdot G_0}_{\text{Geração na fatia 0}} + \underbrace{P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}_{\text{Propagação nas fatias 0, 1, 2 e 3}}$$

- C_4: útil se bloco for conectado em outro bloco usando carry ripple.

Somador binário de n bits

- Se conexão for feita com outro bloco usando carry look-ahead, podemos escrever:

$$C_{BLOCO4} = G_{BLOCO4} + P_{BLOCO4} \cdot C_0$$

onde:

$$G_{BLOCO4} = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0$$

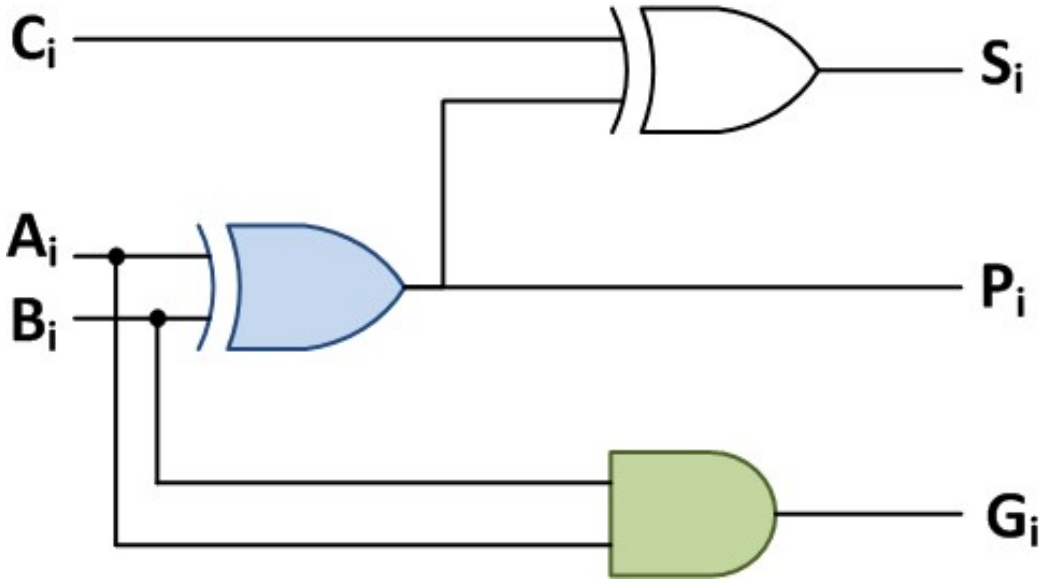
com $G_i = A_i \cdot B_i$

$$P_{BLOCO4} = P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

com $P_i = (A_i \oplus B_i)$

Somador binário de n bits

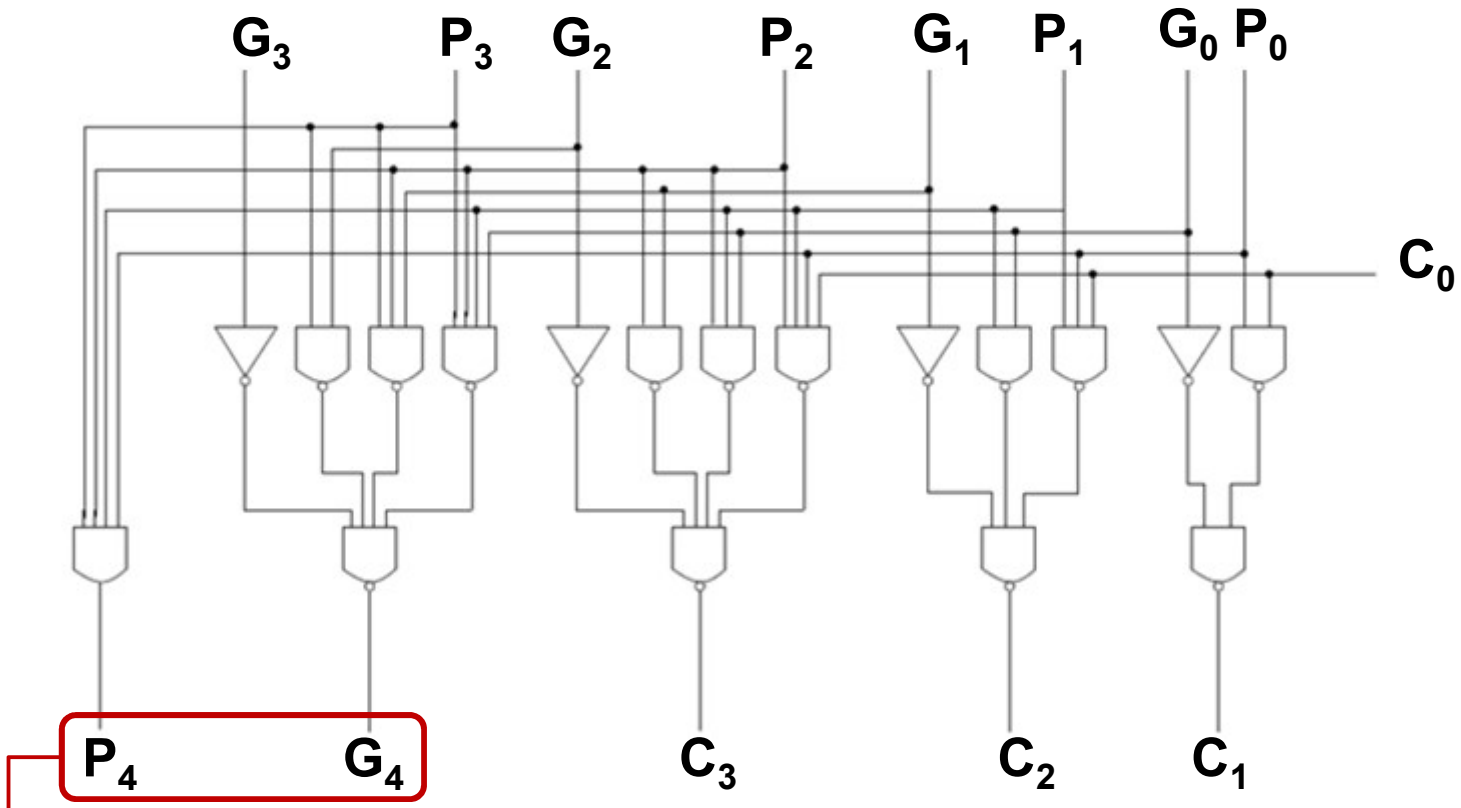
- Carry look-ahead: **Unidade somadora** de 1 bit



$G_i = A_i \bullet B_i$; $P_i = (A_i \oplus B_i)$

Somador binário de n bits

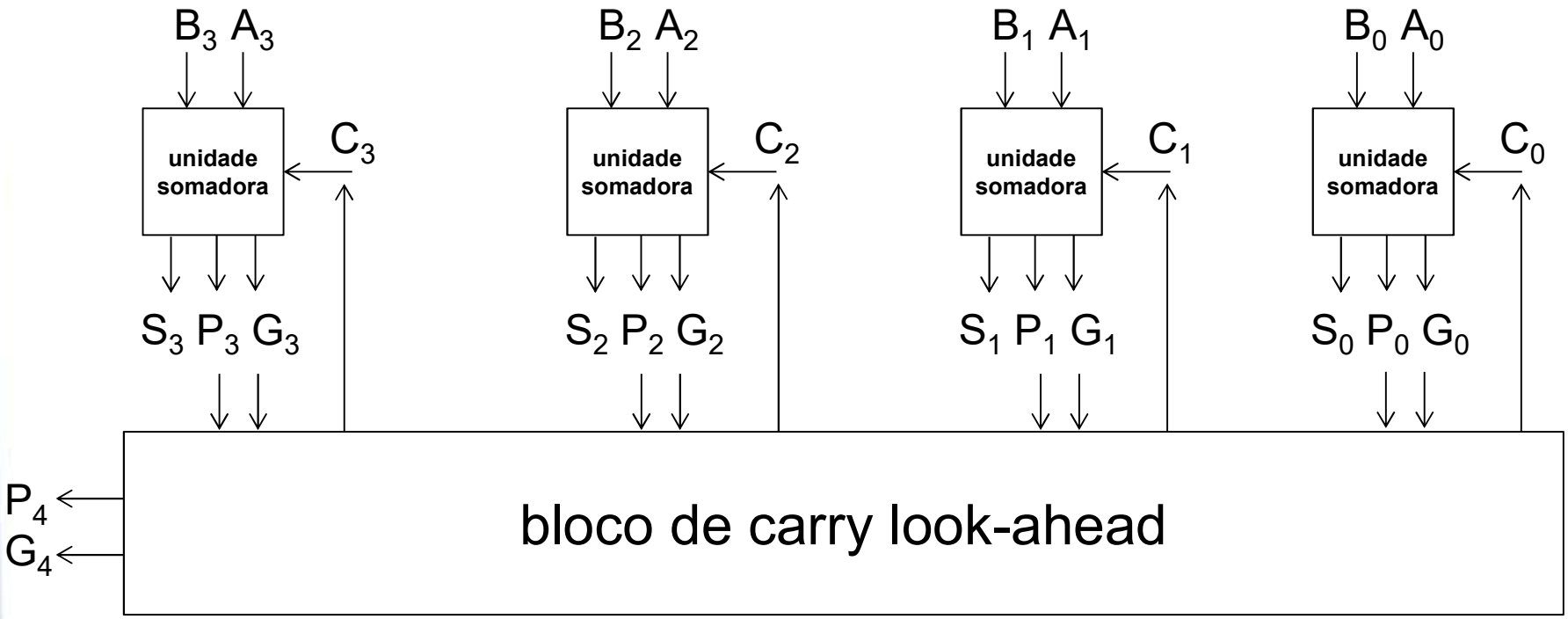
- Carry look-ahead: bloco para **cálculo antecipado** do carry



Podem ser usados para calcular C4 (útil para conectar blocos do tipo carry ripple) ou diretamente em um outro bloco de carry look-ahead

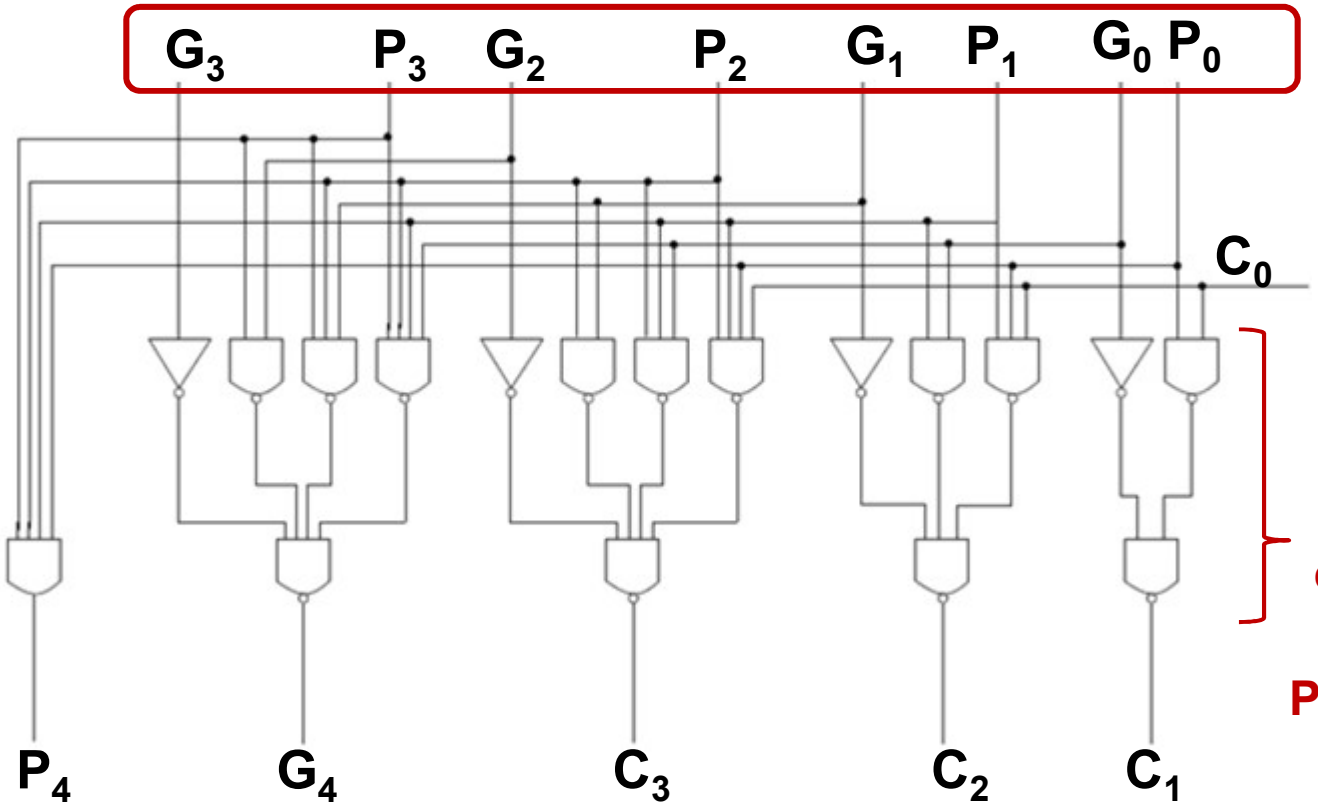
Somador binário de n bits

- Circuito completo com 4 bits

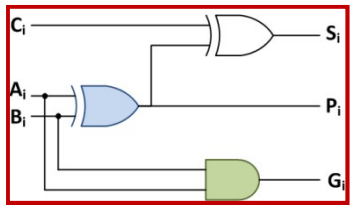


Somador binário de n bits

- Circuito completo com 4 bits: custo



P e G: latência de 1 porta



Latência extra de 2

Por que não fazer em 2 níveis?!

Fazer em 2 níveis seria inviável: 9 entradas ➔ 2^9 mintermos possíveis...
Solução é um compromisso entre latência e tamanho do circuito

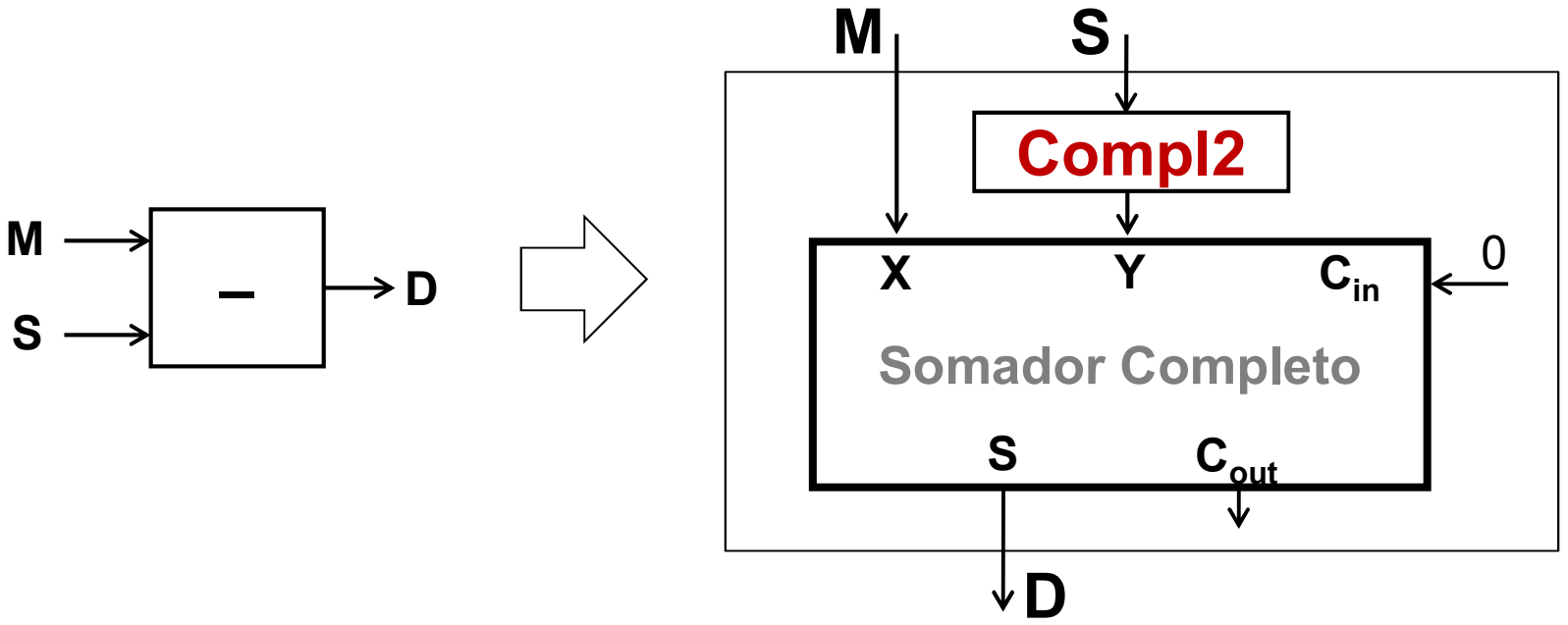
Subtrator binário

- Pode ser construído da mesma forma que construímos o somador binário: a partir da tabela verdade…
- ... ou usar o circuito do somador!

entradas			saídas: +		saídas: –	
c_{IN} [b_{IN}]	x	y	Σ (soma)	c_{OUT}	d (diferença)	b_{OUT}
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	1	1

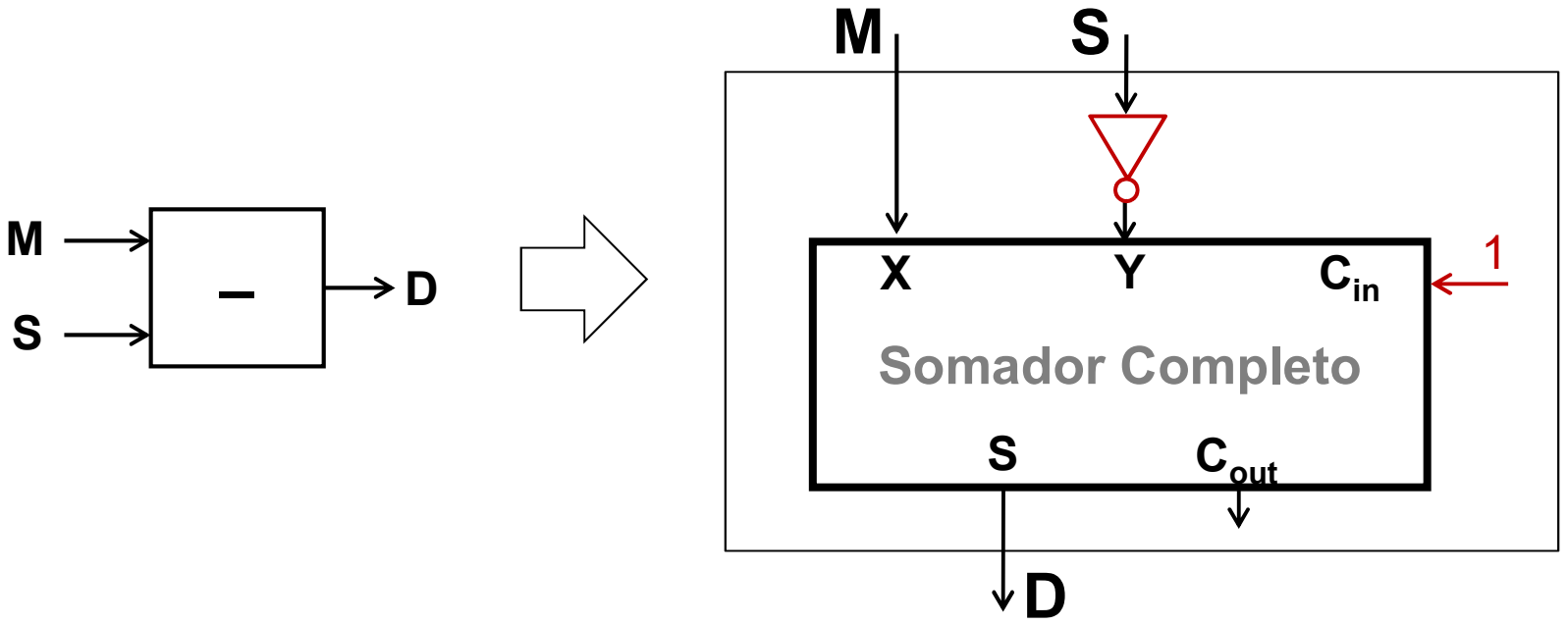
Subtrator binário

- Subtração básica: minuendo – subtraendo
  - D = M – S = M **+** (–S) = M + Compl2(S)
  - Logo: **subtração** pode ser construída **a partir** de uma **operação de adição** em um **somador completo**



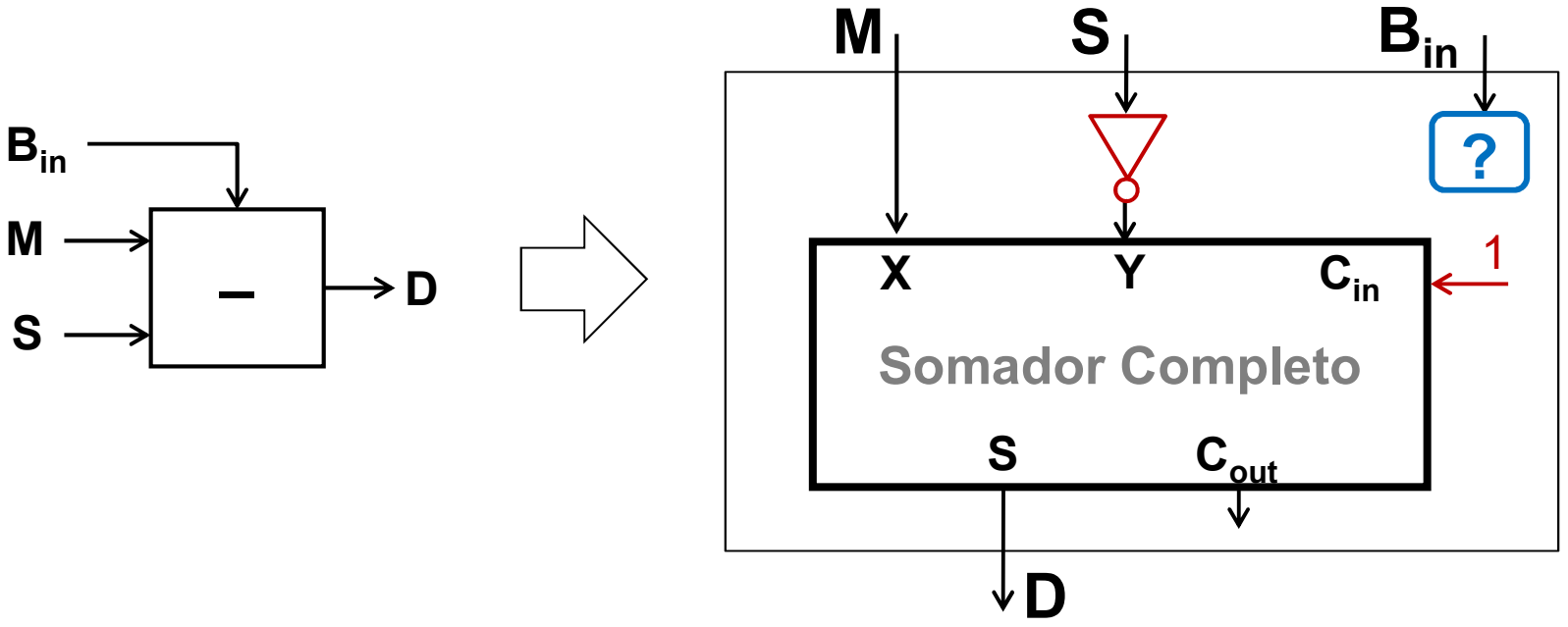
Subtrator binário

- Subtração básica: minuendo – subtraendo
  - D = M – S = M **+** (–S) = M + Compl2(S)
  - Logo: **subtração** pode ser construída **a partir** de uma **operação de adição** em um **somador completo**



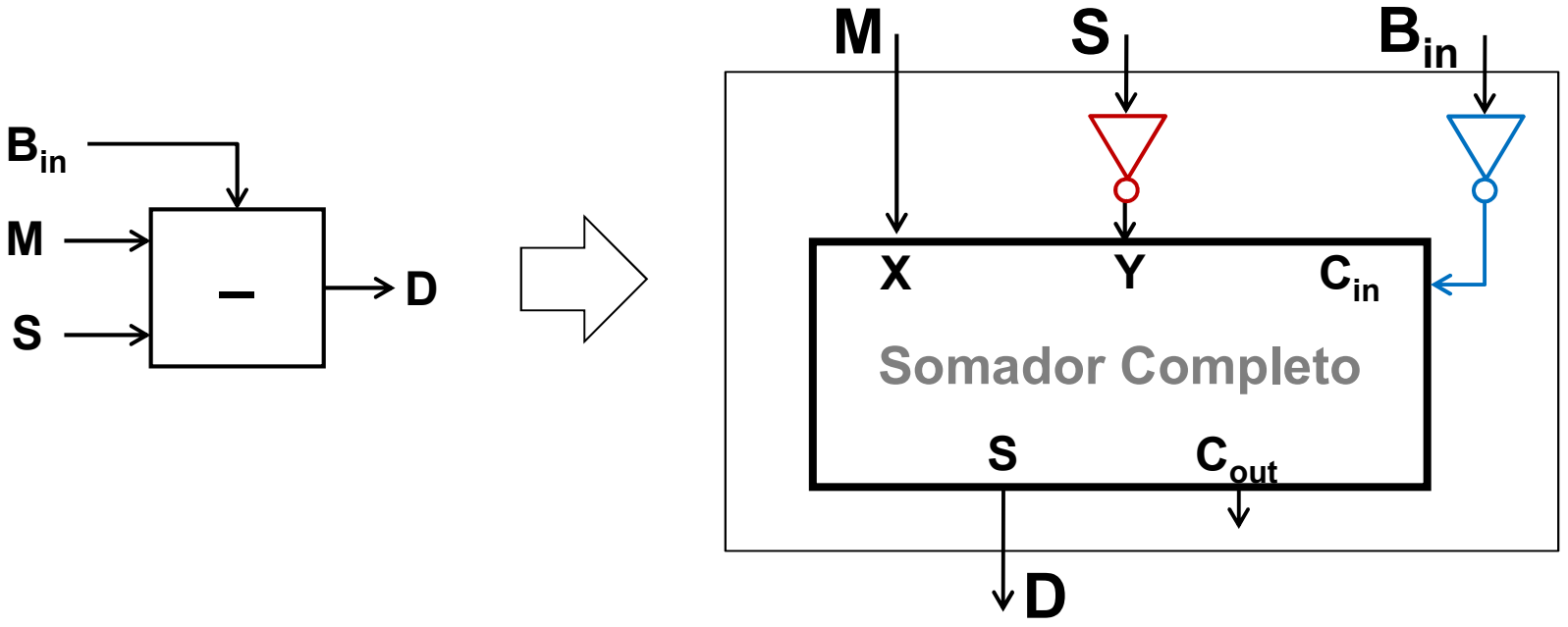
Subtrator binário

- Subtração com empresta-um (borrow)
  - $D = M - S - B_{in} = M + \text{Compl2}(S) - B_{in}$
  - Se B_{in}=1, anula o "+1" do complemento-de-2



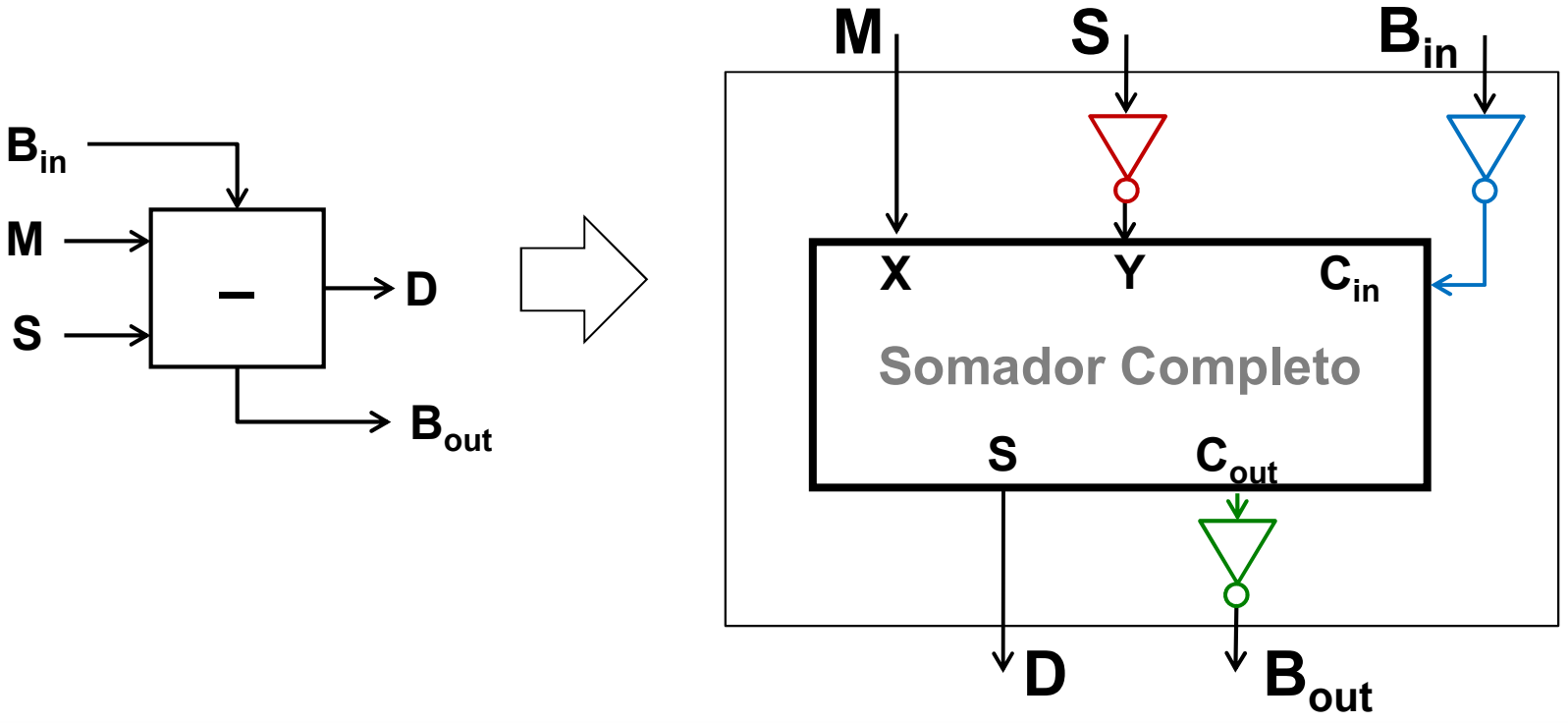
Subtrator binário

- Subtração com empresta-um (borrow)
  - $D = M - S - B_{in} = M + \text{Compl2}(S) - B_{in}$
  - Se B_{in}=1, anula o “+1” do complemento-de-2



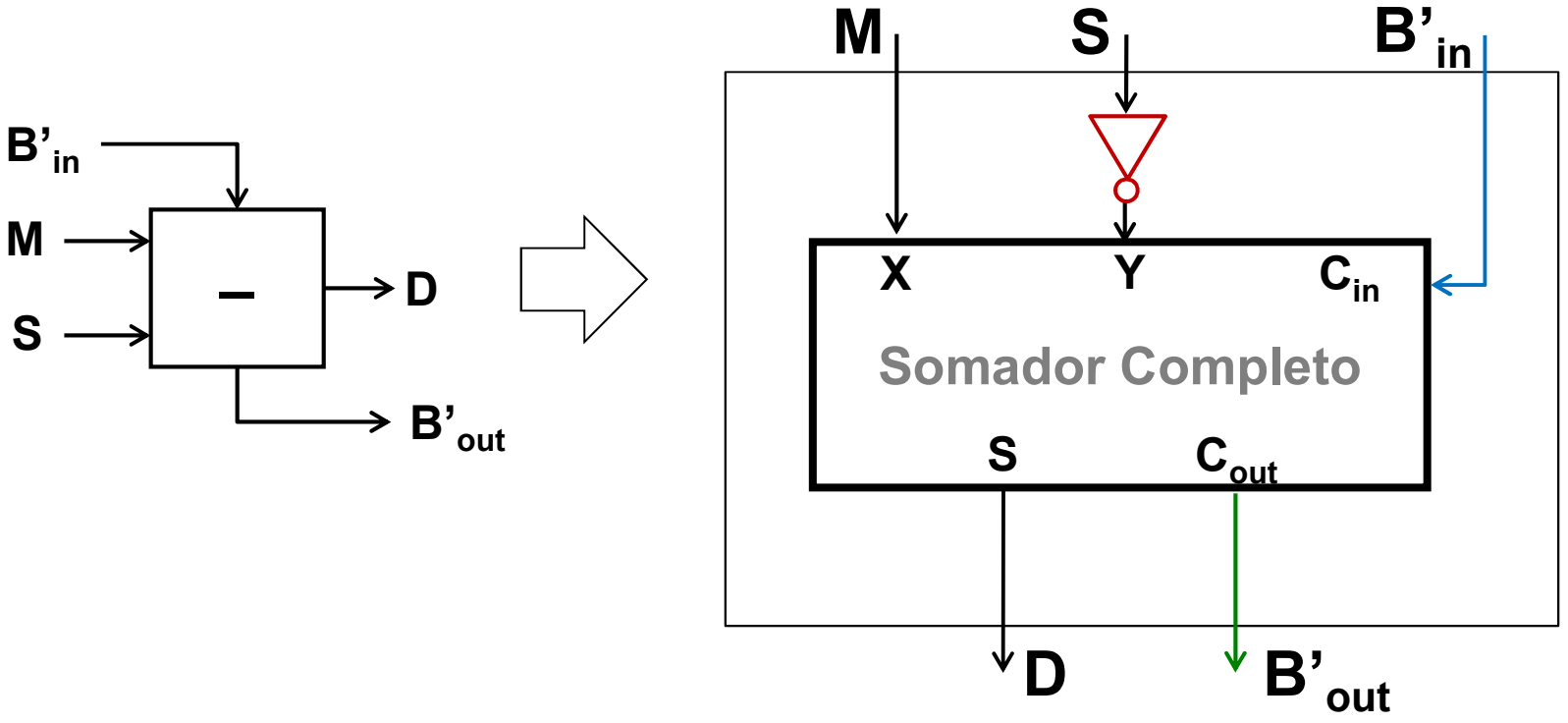
Subtrator binário

- Subtração completa: B_{in} e B_{out}
  - D = M – S – B_{in} = M + Compl2(S) – B_{in}
  - B_{out} = (C_{out})' (vide tabela verdade)



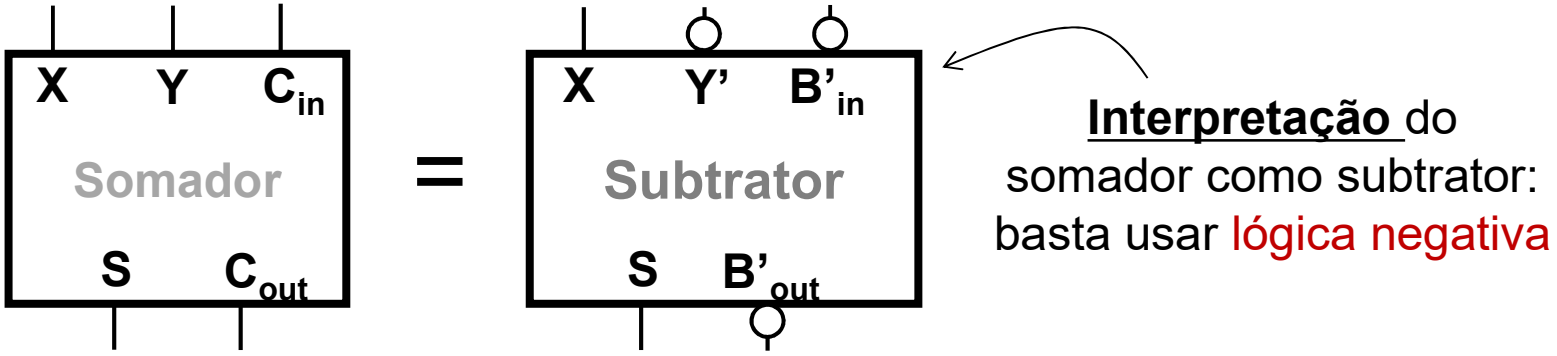
Subtrator binário

- Subtração completa: B_{in} e B_{out} em lógica negativa
  - $D = M - S - B_{in} = M + \text{Compl2}(S) - B_{in}$
  - $B_{out} = (C_{out})'$ (vide tabela verdade)

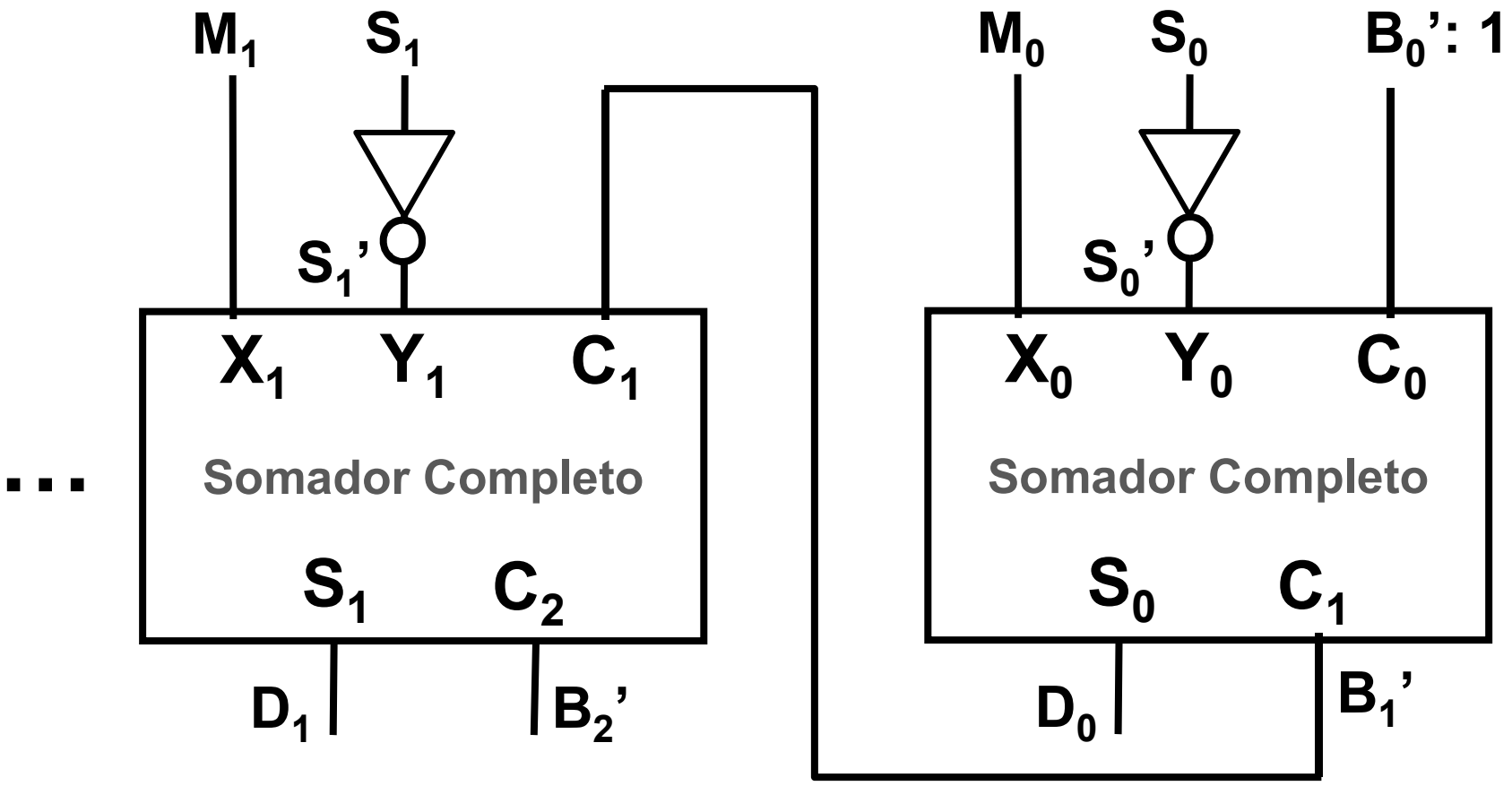


Subtrator binário

- Conclusão: subtrator completo a partir de somador completo
  - Manter minuendo ($\mathbf{M_i}$)
  - **Complementar *bit* a *bit*** o subtraendo ($\mathbf{S_i}$)
  - Como não há borrow (i.e., $B_0 = 0$), fazer $\mathbf{C_0 = 1}$ no somador completo (equivale a $B_0' = 1$)



Subtrator binário



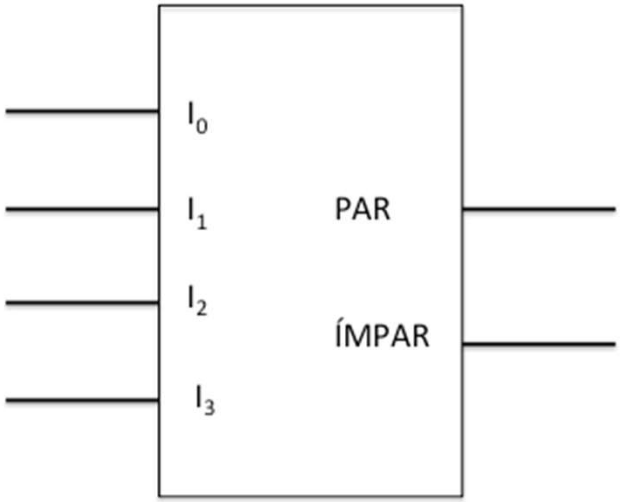
Nota: no final, precisa detectar transbordo (verificar se sinais na entrada diferem dos sinais de saída)

Detectando Overflow

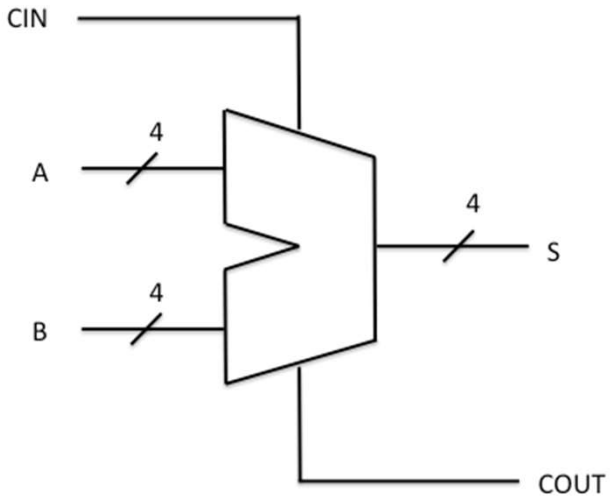
- Quando operamos apenas com **números positivos**
  - Somadores: overflow = carry-out na última camada
  - Subtratores: nunca dão overflow
- Quando operamos com **números em complemento de dois**, a mesma abordagem de detecção vale para somadores e subtratores:
  - Overflow = se bits de sinal das entradas são iguais, mas diferem do bit de sinal da saída
  - Overflow = se o carry-in na camada do bit de sinal difere do carry-out da camada do bit de sinal
  - Obs.: abordagens equivalentes

Exercício (PREC 2016)

- Seja A um número binário de 4 bits. Projete um circuito que calcule $A - 1_{10}$ caso A tenha paridade par e $A - 2_{10}$ caso A tenha paridade ímpar. A subtração deve ser calculada em Complemento de 2. Utilize (somente!) um gerador de paridade de 4 bits (Figura 1) e um somador completo de 4 bits (Figura 2).



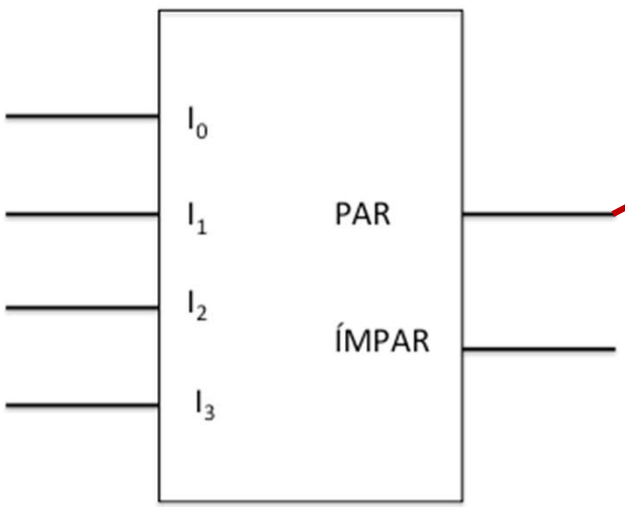
Gerador de Paridade



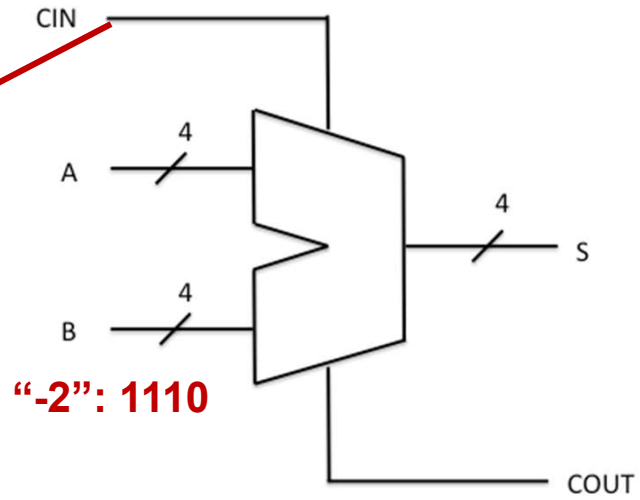
Somador Completo de 4 bits

Exercício (PREC 2016)

- Seja A um número binário de 4 bits. Projete um circuito que calcule $A - 1_{10}$ caso A tenha paridade par e $A - 2_{10}$ caso A tenha paridade ímpar. A subtração deve ser calculada em Complemento de 2. Utilize (somente!) um gerador de paridade de 4 bits (Figura 1) e um somador completo de 4 bits (Figura 2).



Gerador de Paridade



Somador Completo de 4 bits

APÊNDICE

Subtrator: correspondência com somador

Subtrator binário

M_i	S_i	B_i	R_i	B_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabela Verdade do Subtrator Completo:

$$M_i - S_i - B_i$$

minuendo – subtraendo – borrow (“empresta um”)

$$R_i = M_i' \cdot S_i' \cdot B_i + M_i' \cdot S_i \cdot B_i' + M_i \cdot S_i' \cdot B_i' + M_i \cdot S_i \cdot B_i$$

$$B_{i+1} = M_i' \cdot S_i + B_i \cdot M_i' + B_i \cdot S_i$$

Subtrator binário

$$R_i = M_i'.S_i'.B_i + M_i'.S_i.B_i' + M_i.S_i'.B_i' + M_i.S_i.B_i$$

Comparando com somador

$$R_i = M_i \oplus S_i' \oplus B_i'$$

$$\Updownarrow \quad \Updownarrow \quad \Updownarrow \quad \Updownarrow$$

$$Si = Ai \oplus Bi \oplus Ci$$

negar entrada S_i

saída é B'_{i+1}: usá-la no próximo estágio

$$B_{i+1} = M_i'\bullet S_i + B_i\bullet M_i' + B_i\bullet S_i =$$

$$[B_{i+1}]' = [M_i'\bullet S_i + (M_i' + S_i)\bullet B_i]' =$$

$$B_{i+1}' = M_i \bullet S_i' + (M_i + S_i')\bullet B_i' =$$

Comparando com somador

$$B_{i+1}' = M_i \bullet S_i' + M_i \bullet B_i' + S_i' \bullet B_i'$$

$$\Updownarrow \quad \Updownarrow \Updownarrow \quad \Updownarrow \Updownarrow \quad \Updownarrow \Updownarrow$$

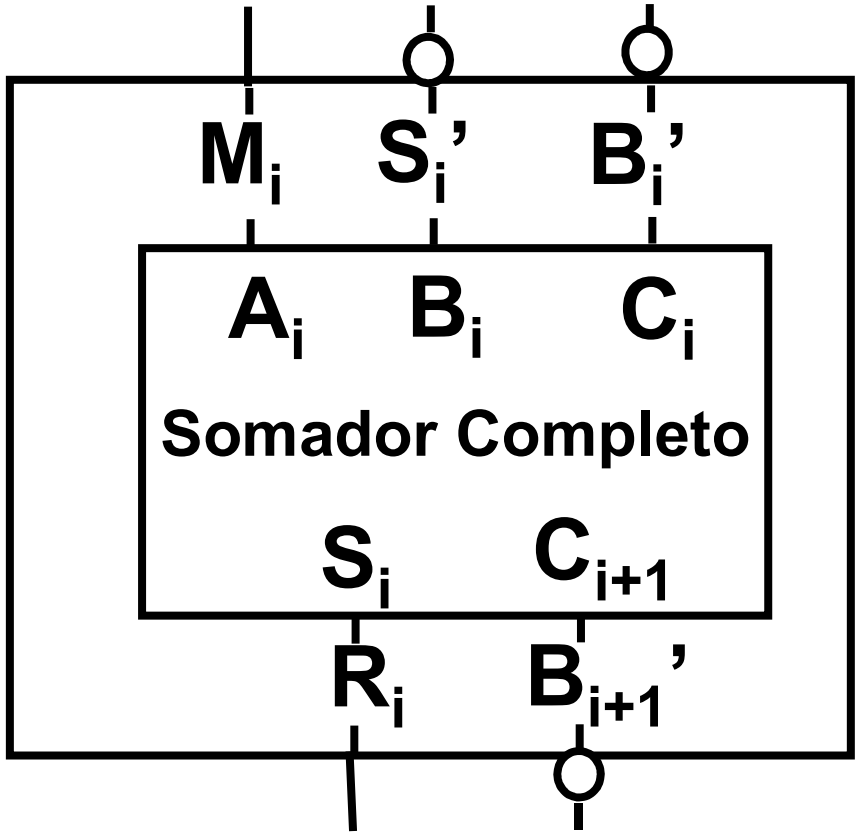
$$C_{i+1} = A_i \bullet B_i + A_i \bullet C_i + B_i \bullet C_i$$

Subtrator binário

$R_i = M_i \oplus S_i' \oplus B_i'$

$Si = Ai \oplus Bi \oplus Ci$

Subtrator Completo



$$B_{i+1}' = M_i \cdot S_i' + M_i \cdot B_i' + S_i' \cdot B_i'$$

$$C_{i+1} = A_i \cdot B_i + A_i \cdot C_i + B_i \cdot C_i$$

APÊNDICE

Somador BCD

Somador BCD

- Relembrando: código BCD
  - 4 bits, representação direta binária de 0 a 9
- Soma BCD: usa a adição binária com 4 bits, mas não é idêntica a ela...
  - Requer correção dos valores inválidos:
    - Aqueles acima de 9
  - Requer correção do “vai-um decimal”:
    - Diferente do vai-um hexadecimal (de 4 bits)

Somador BCD

Comparação entre soma binária (4bits) e BCD

Resultado da soma de 4 bits	Soma Hexa	Vai-um Hexa	Soma BCD	Vai-um BCD	Correção
0 a 9	0 a 9	0	0 a 9	0	-
10 a 15	10-15 (A-F)	0	0 a 5	1	Soma 6 $C_{BCD}=1$
16 a **19**	0-3	1	6 a 9	1	Soma 6 $C_{BCD}=C_{hexa}$

Obs.: 9+9+1 (“vem-um”)

Somador BCD

- Correção da soma BCD em relação ao resultado hexadecimal:
  - Se resultado da soma entre A e F, **OU**
  - Se vai-um$_{hexa}$ = 1

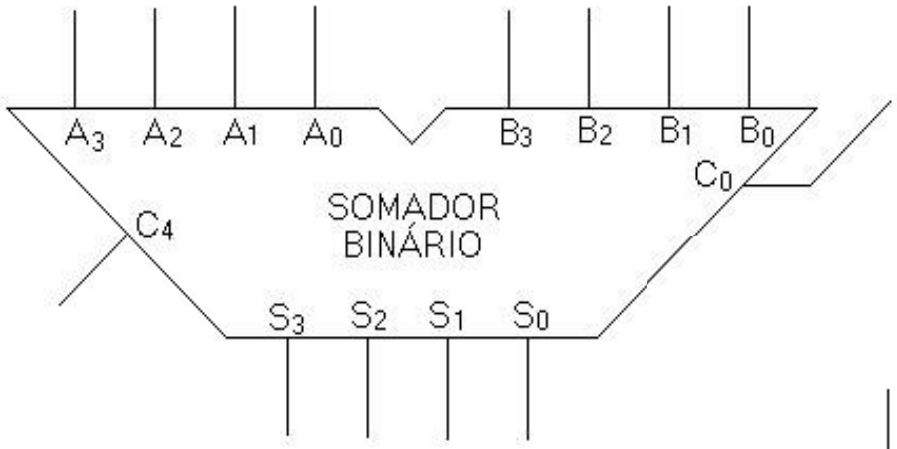
Lógica adicional sobre saída do somador hexa

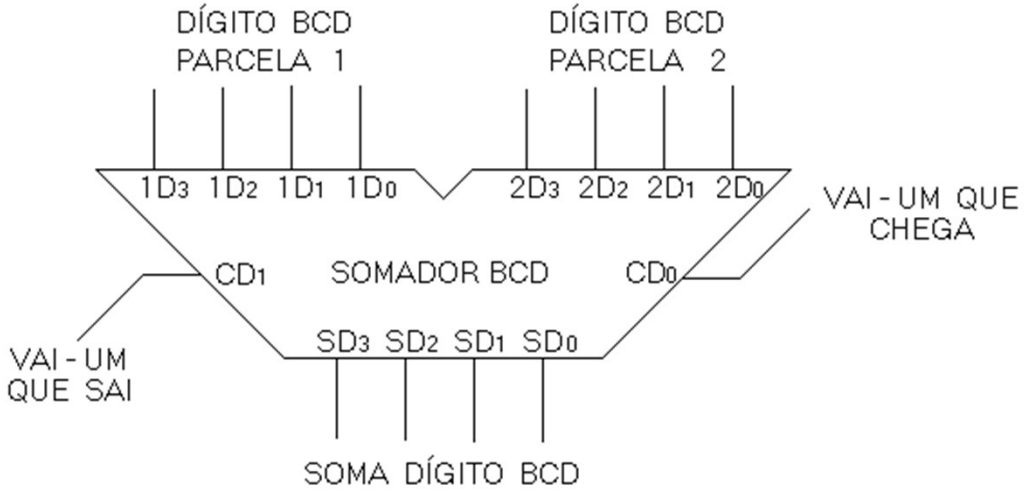
- Nesse caso, ação a tomar:
  - **Somar 6** (0110) à soma hexa
  - Vai-um$_{BCD}$=1

Necessário um segundo somador

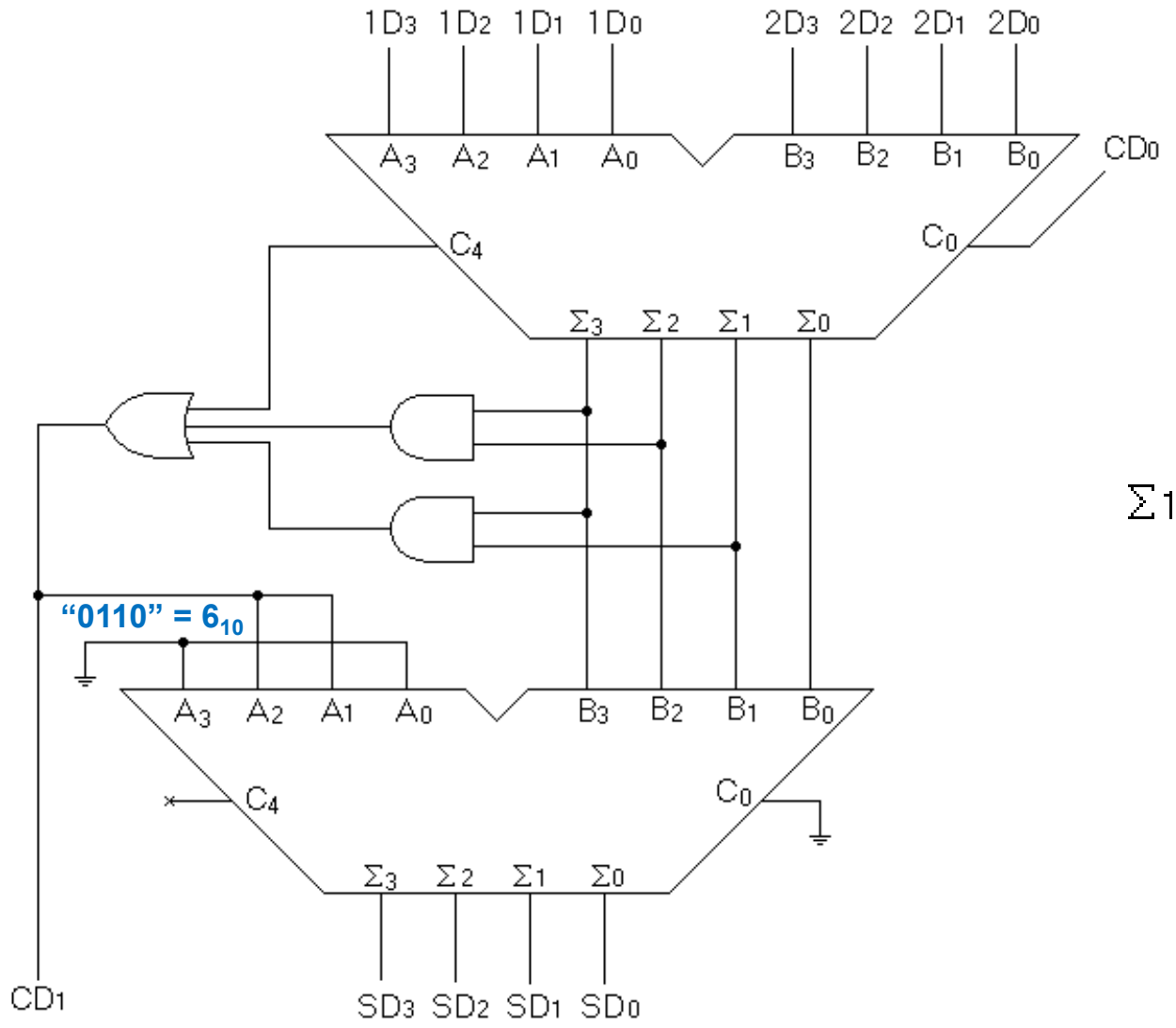
Somador BCD

- Desafio: obter um Somador BCD a partir de um Somador Binário de (4 bits)





Somador BCD



10-16

Σ1Σ0 \ Σ3Σ2	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	1