PCS 3115

Sistemas Digitais I

Circuitos Combinatórios
Blocos Básicos:

Somadores e Subtratores

Prof. Dr. Marcos A. Simplicio Jr.

versão: 3.0 (Jan/2016)

Adaptado por Glauber (2018)

Blocos básicos

- Codificadores e Decodificadores
- (De) Multiplexadores
- Portas tri-state
- Comparadores
- Somadores/Subtratores **(HOJE)**
- **16 de maio: P2**
- Multiplicadores
- ULA
- Gerador/Detector de Paridade

Somadores binários

- Soma binária: uma das operações aritméticas mais comuns em sistemas digitais
- Somador Binário combina dois operandos aritméticos usando as regras da soma binária
- Regras da soma são as mesmas para números sem sinal e para números em Complemento de 2
- Somador pode realizar uma subtração como a “soma do Minuendo com o complemento do Subtraendo”
  - Ou seja: a – b = a + (– b)

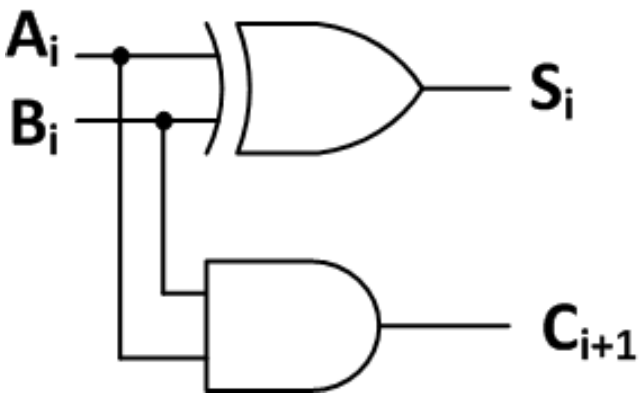
Meio Somador

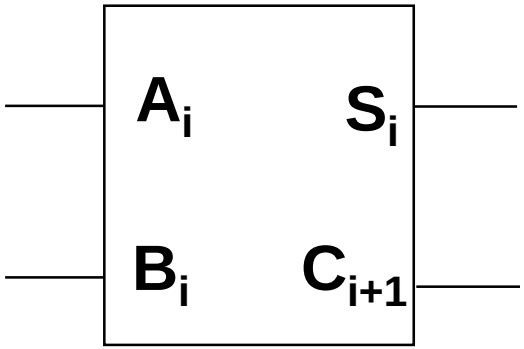
- A soma de dois operandos de 1 bit produz 2 bits
  1. Resultado da soma, e
  2. "Vai-um" (*carry*)
- Operação realizada por "meio somador"
  - Não considera "vem-um"
  - Logo, não funciona para números de 2+ bits
    - Teste: 0010 + 0110 ➔ sem o "vem-um": 0100 (errado...)
      ➔ com o "vem-um": 1000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

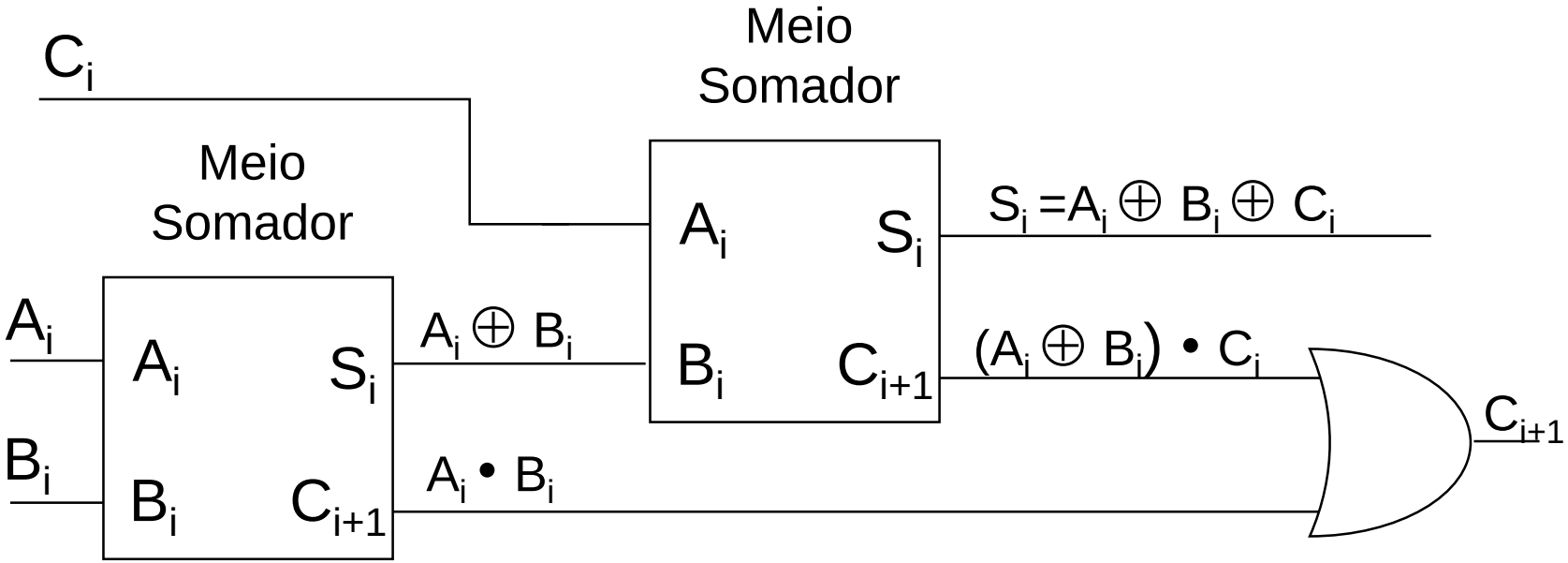
xor and

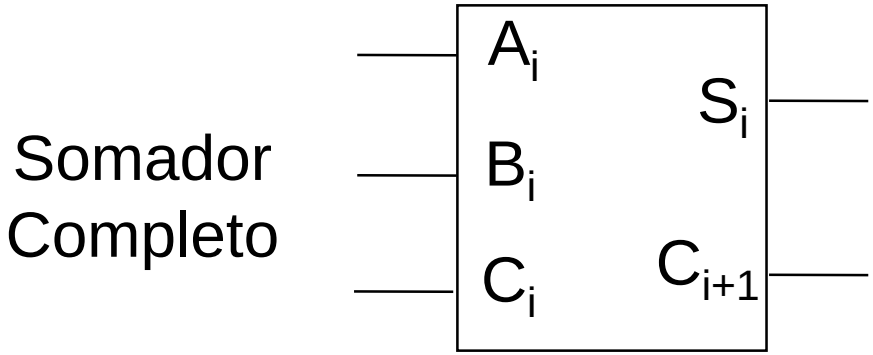
Somador completo

- 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”

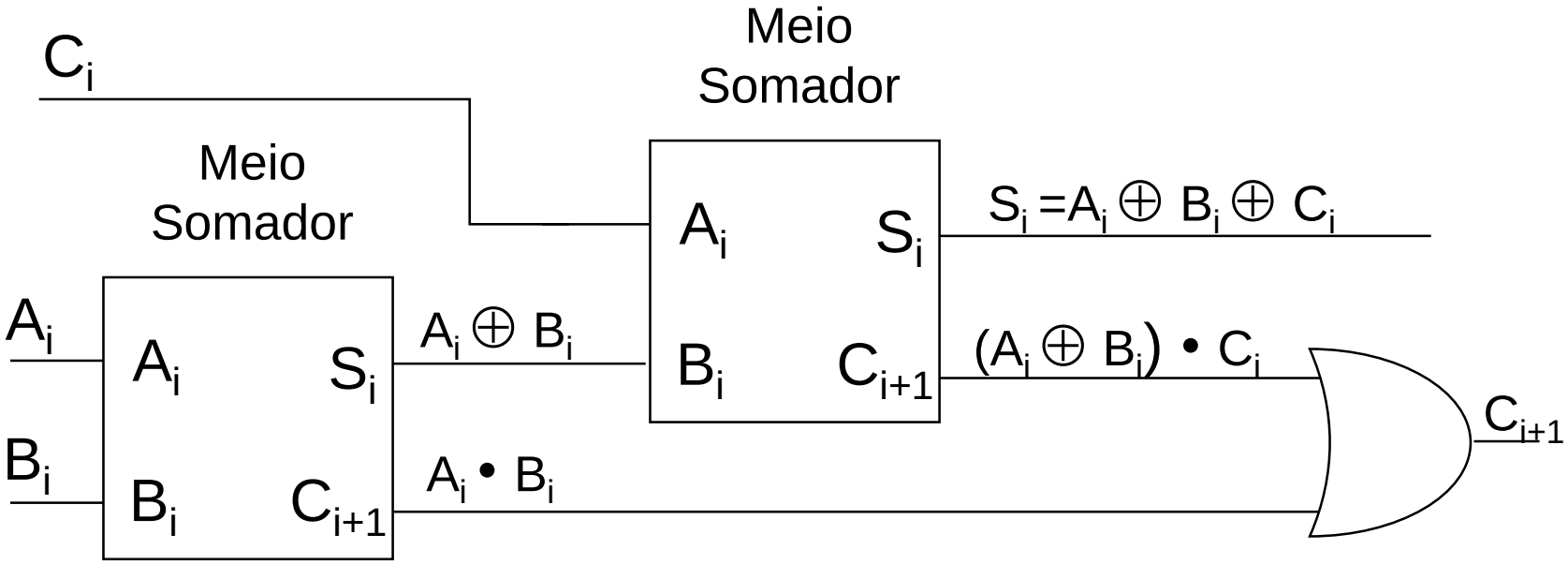
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Somador completo





Somador completo



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

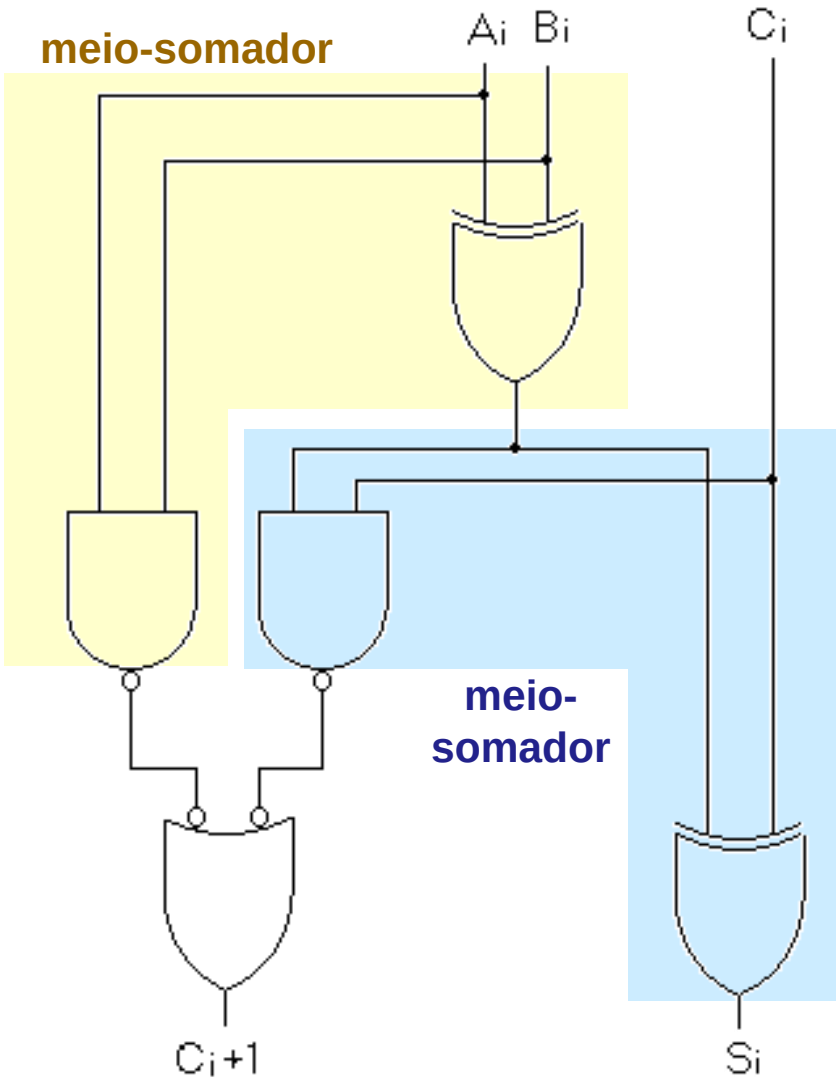
$C_{i+1} = A_i'\bullet B_i\bullet C_i + A_i\bullet B_i'\bullet C_i + A_i\bullet B_i\bullet C_i' + A_i\bullet B_i\bullet C_i$ ← mintermos

$= (A_i \oplus B_i)\bullet C_i + A_i\bullet B_i$ ← meio-somadores

$= (A_i + B_i)\bullet C_i + A_i\bullet B_i$ ← ANDs e ORs

$= A_i\bullet C_i + B_i\bullet C_i + A_i\bullet B_i$ ← soma de produto

Somador completo



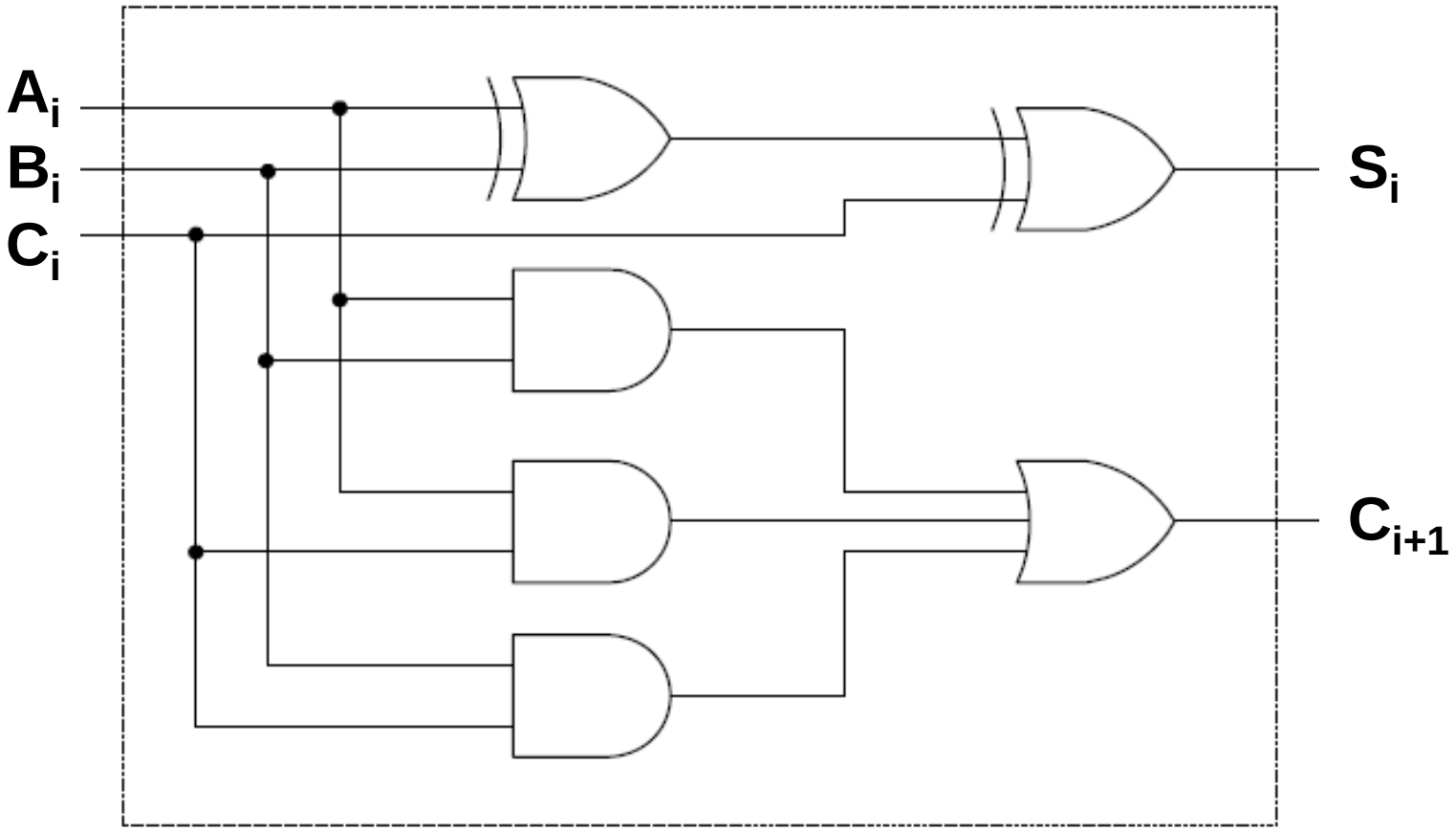
Combinação de meio-somadores:

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

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

➔ **Latência de C_{i+1}: 3**

Somador completo



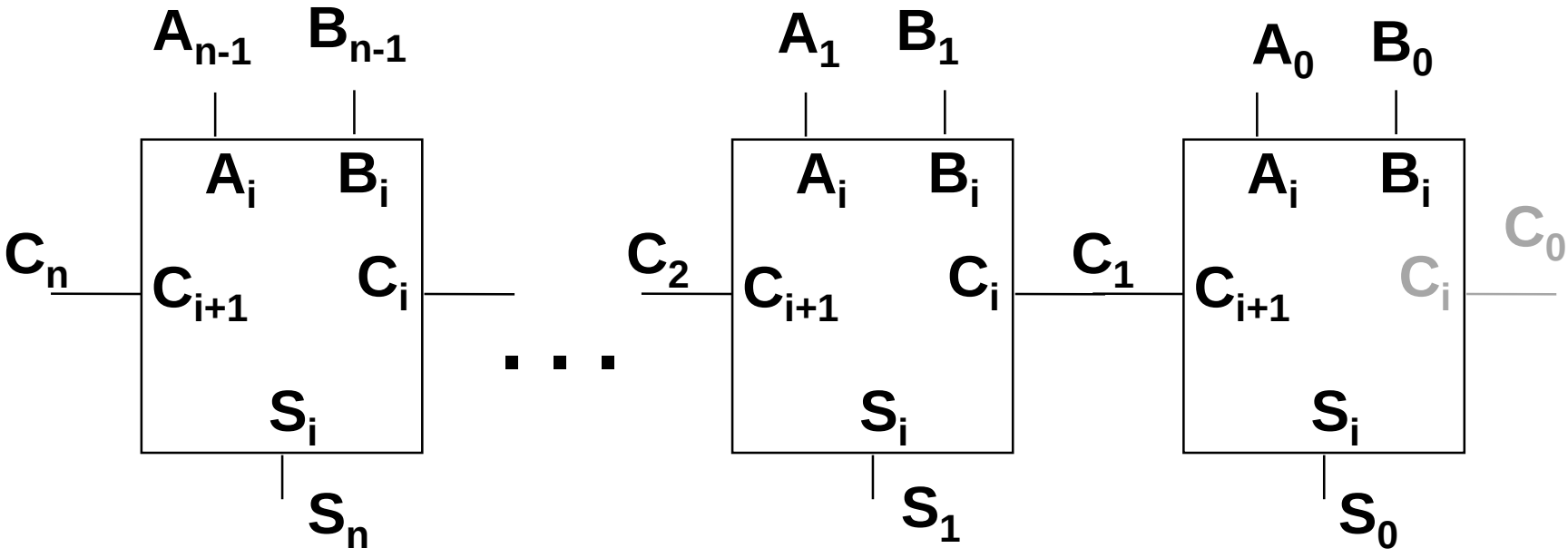
$S_i = A_i \oplus B_i \oplus C_i$; $C_{i+1} = A_i \bullet C_i + B_i \bullet C_i + A_i \bullet B_i$

➔ 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)…
- **Problema**: Somador de 4 bits tem 9 entradas, correspondendo a 2^9 mintermos...

- Solução em 3 níveis: antecipação do vai-um ou *carry look-ahead*

Somador binário de n bits

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

OU

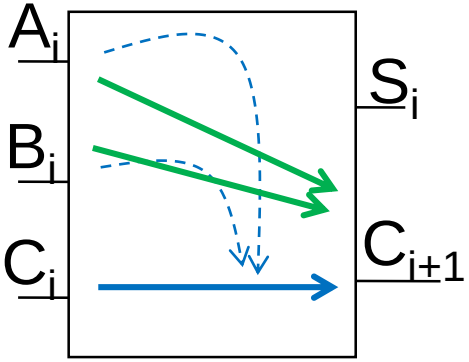
$$C_{i+1} = A_i \cdot B_i + C_i \cdot (A_i \oplus B_i)$$

$= G_i$ $= P_i$

Duas origens possíveis de C_{i+1}

O C_{i+1} foi **G**erado na fatia i

C_i foi **P**ropagado por fatia i



Somador binário de n bits

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

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

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

- 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:

$$C_{BLOCO} = C_4 = G_{BLOCO} + P_{BLOCO} \cdot C_0$$

onde:

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

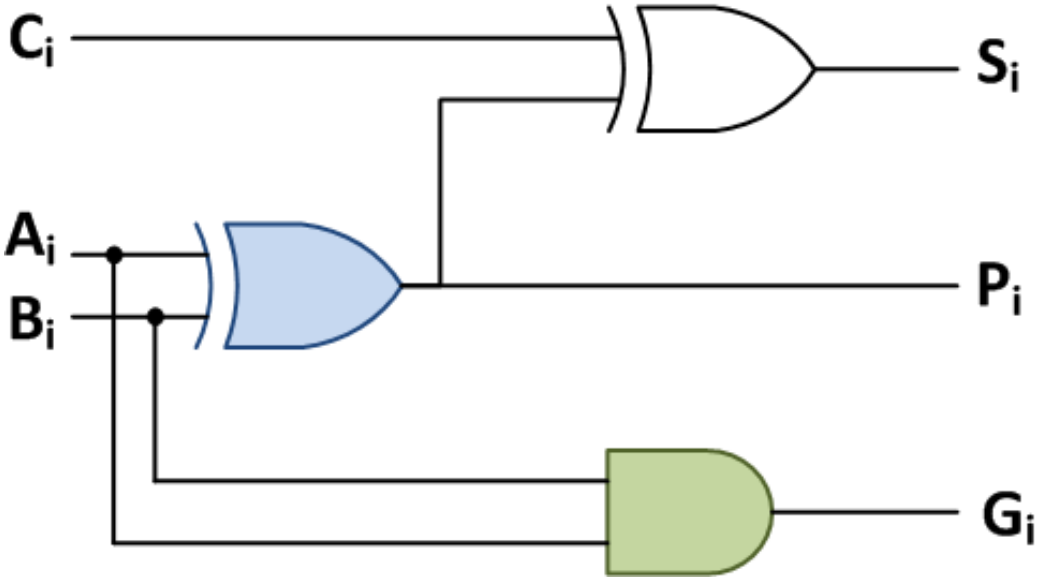
com $G_i = A_i \cdot B_i$

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

com $P_i = (A_i \oplus B_i)$

Somador binário de n bits

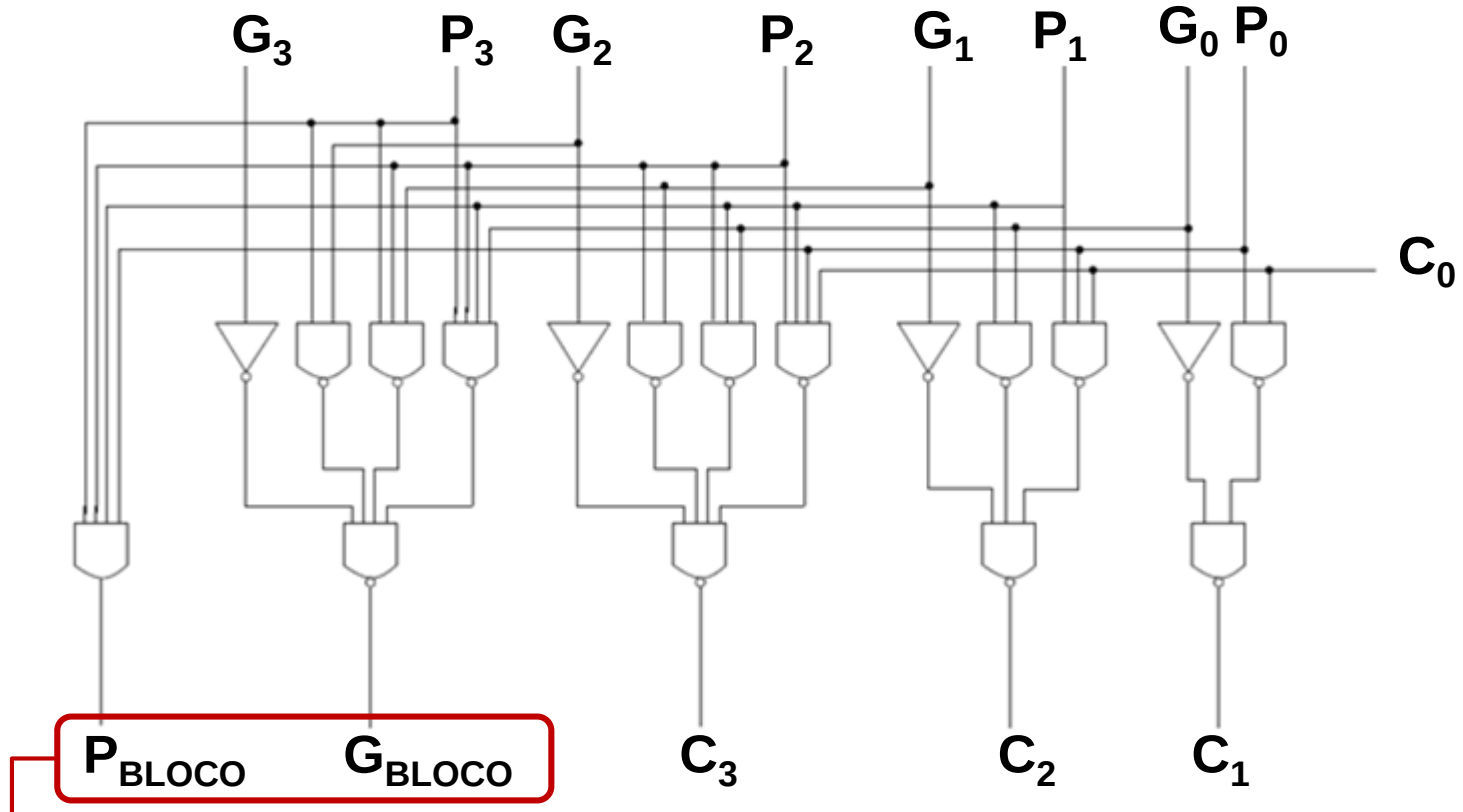
- Carry look-ahead: **Unidade somadora** de 1 bit



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

Somador binário de n bits

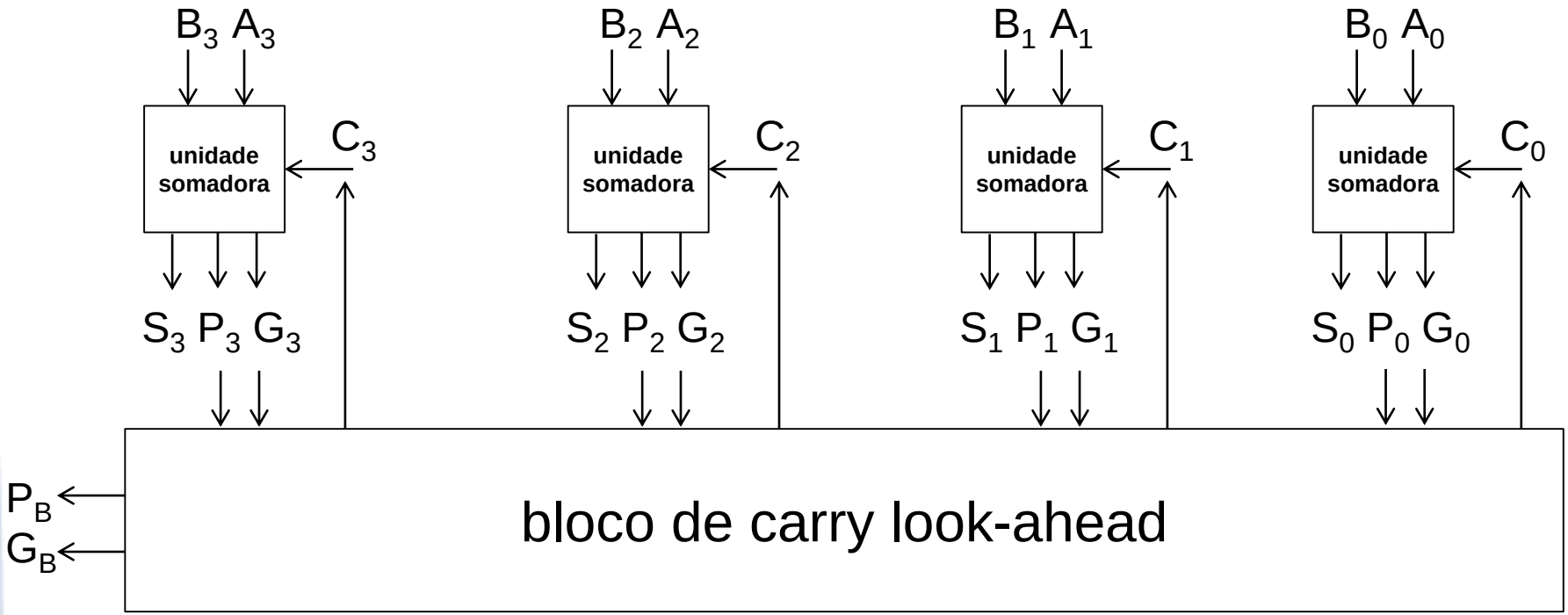
- Carry look-ahead: bloco para **cálculo antecipado** do carry



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

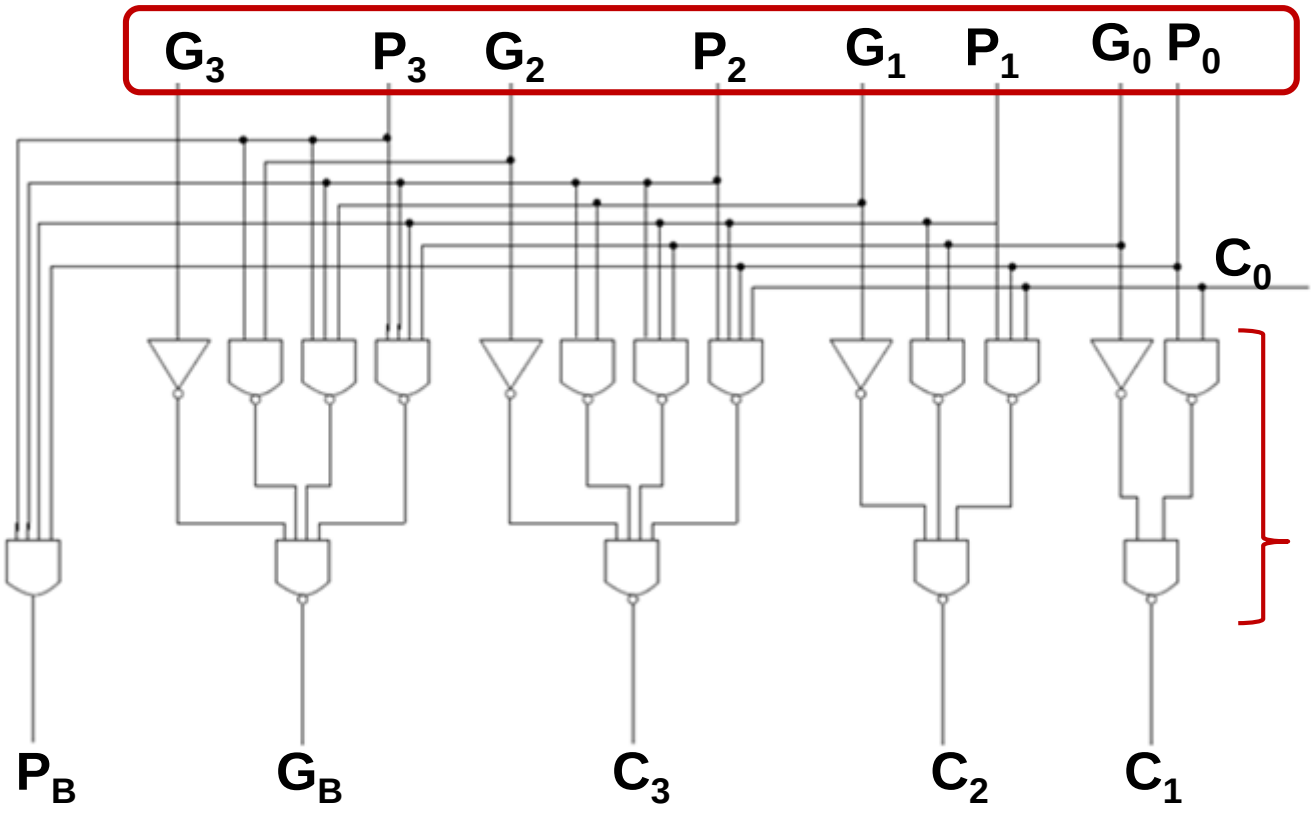
Somador binário de n bits

- Circuito completo com 4 bits

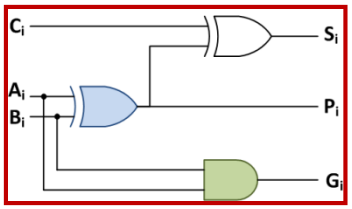


Somador binário de n bits

- Circuito completo com 4 bits: custo

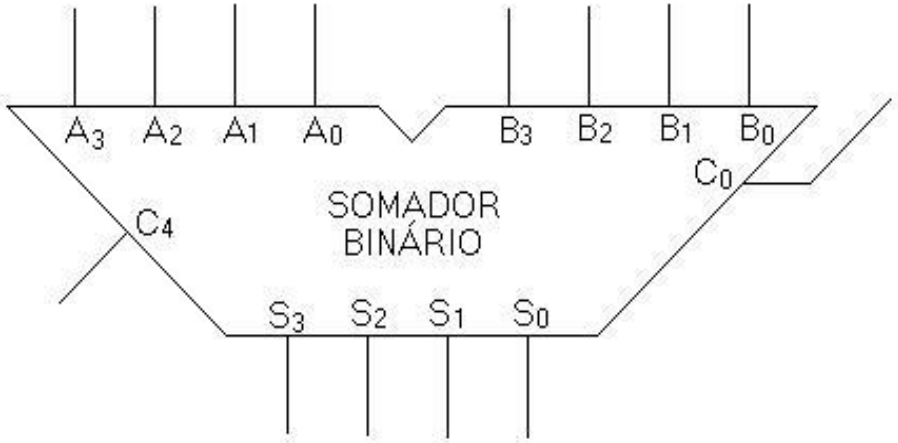


P e G: latência de 1 porta



Latência extra de 2

Representação de um somador



Alternativa para meia soma

$$
\begin{aligned}
hs_i &= x_i \oplus y_i \\
&= x_i \cdot y_i' + x_i' \cdot y_i \\
&= x_i \cdot y_i' + x_i \cdot x_i' + x_i' \cdot y_i + y_i \cdot y_i' \\
&= (x_i + y_i) \cdot (x_i' + y_i') \\
&= (x_i + y_i) \cdot (x_i \cdot y_i)' \\
&= p_i \cdot g_i'
\end{aligned}
$$

- Porta AND tem menos transistores que a XOR
- Implementação mais comum em somadores comerciais

Somador 74x283

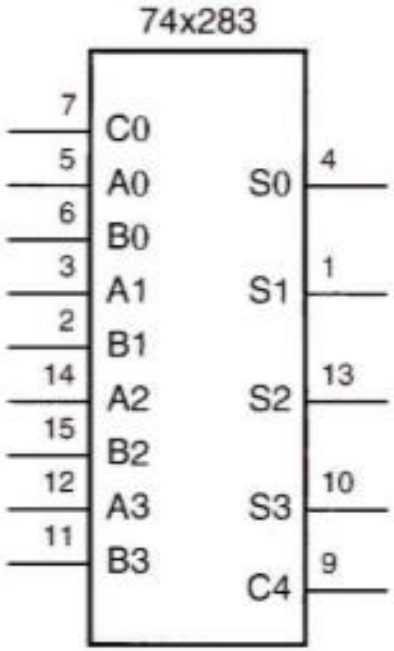


Figure 6-88
Traditional logic symbol for the 74x283 4-bit binary adder.

- Somador de 4 bits com vai-um antecipado.

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

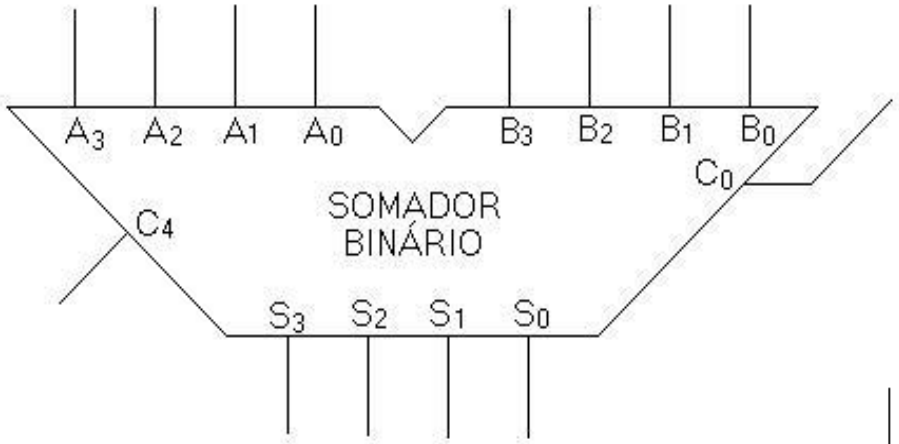
Lógica adicional sobre saída do somador hexa

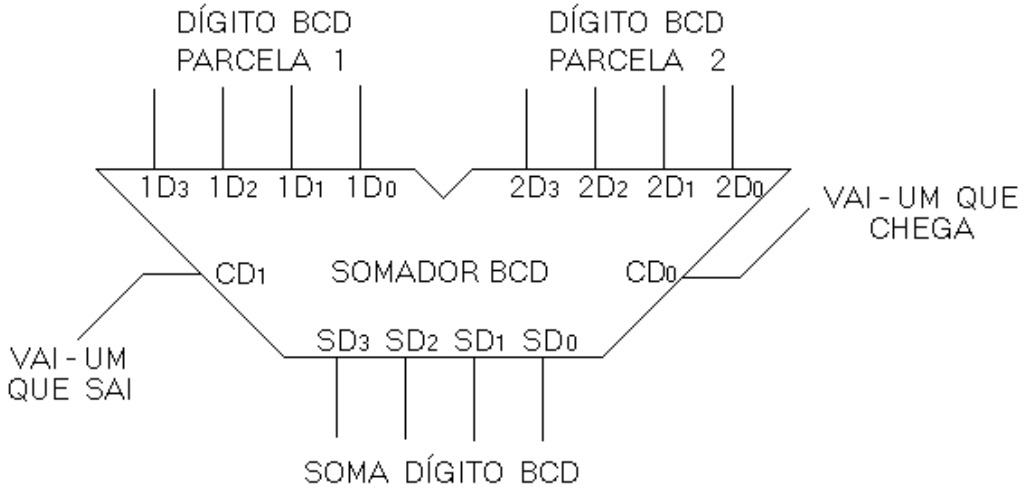
- Nesse caso, ação a tomar:
  - **Somar 6** (0110) à soma hexa
  - Vai-um$_{BCD}$=1

Necessário um segundo somador

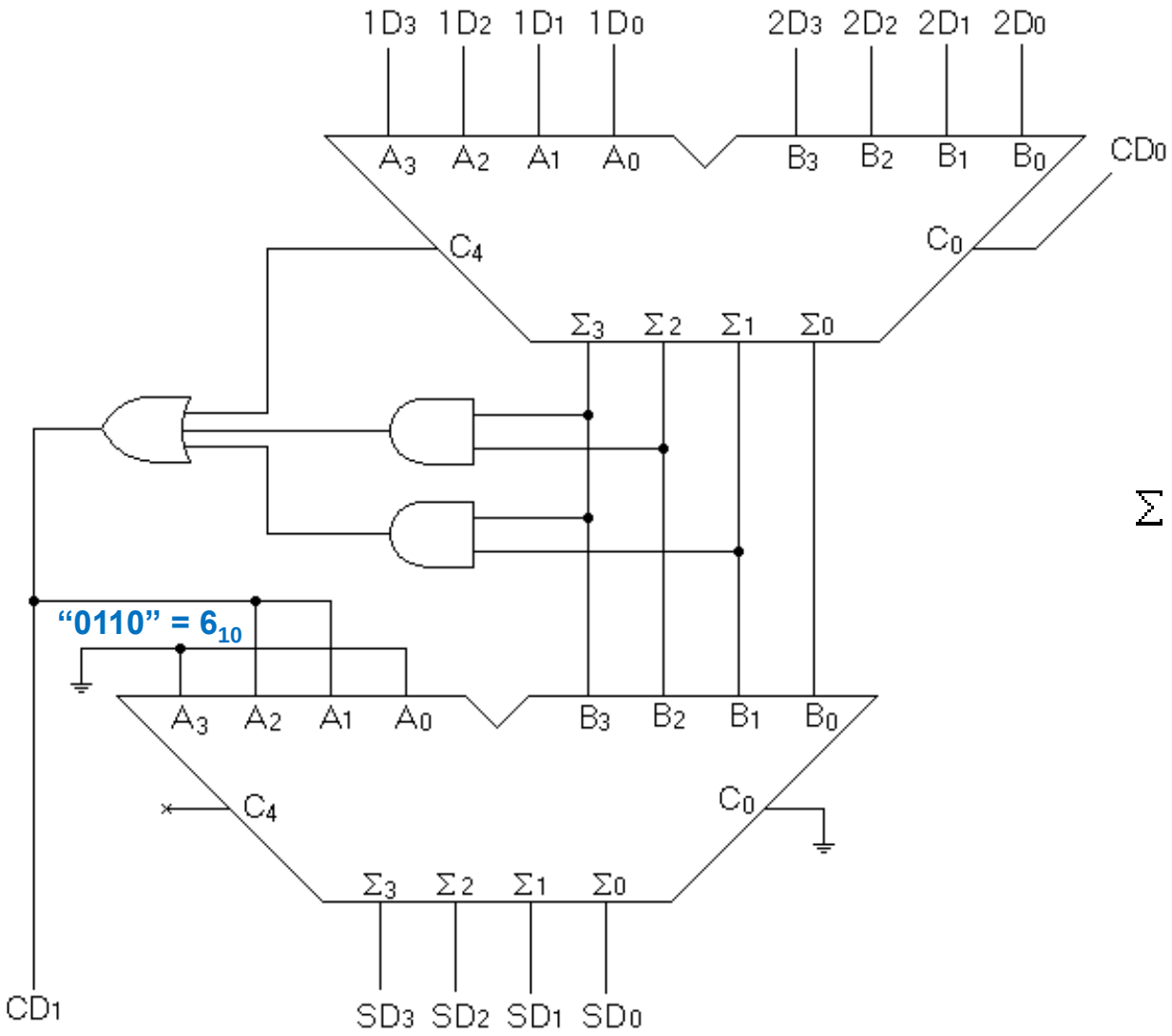
Somador BCD

- 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

Subtrator binário

- Pode ser construído da mesma forma que construímos o somador binário: a partir da tabela verdade.
- Poderíamos fazer um meio-subtrator, depois um complete, depois projetar um circuito para calcular o empresta-1 antecipado
- Alternativamente, podemos utilizar o circuito do 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”)

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

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

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

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

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

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

Comparando com somador

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

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

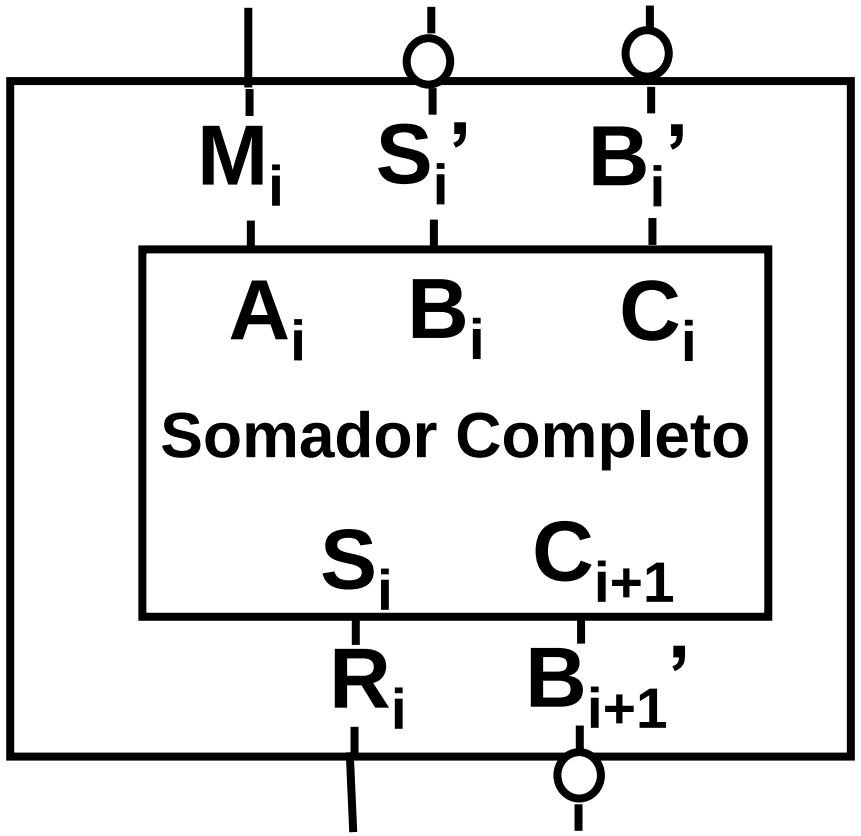
$$C_{i+1} = A_i \bullet B_i + A_i \bullet C_i + B_i \bullet C_i$$

Subtrator binário

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

$Si = Ai \oplus Bi \oplus Ci$

Subtrator Completo



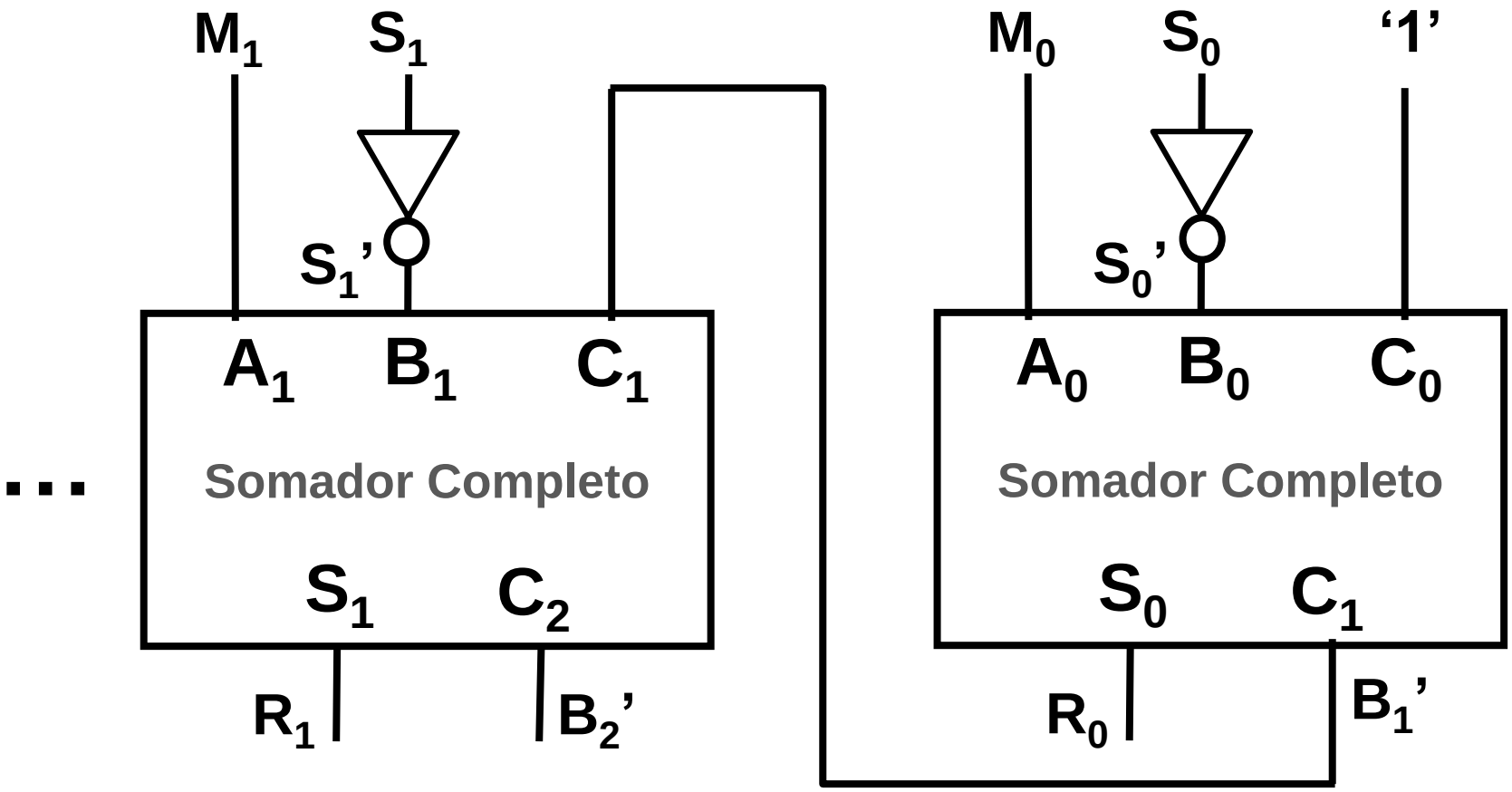
$B_{i+1}' = M_i \cdot S_i' + M_i \cdot B_i' + S_i' \cdot B_i'$

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

Subtrator binário

- Essa correspondência tem uma razão simples: m – s = m + (–s)
  - Logo: **operação de subtração** pode ser construída **a partir** de uma **operação de adição** em um **somador completo**
- Procedimento:
  - **Complementar *bit* a *bit*** o subtraendo (**S_i**)
  - Somar resultado a minuendo (**M_i**), lembrando de fazer **$C_0 = 1$** no somador completo para obter o complemento de 2 de (**S_i**)

Subtrator binário



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