



Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
Curso de Graduação em Ciências da Computação



Sistemas Digitais

INE 5406

Aula 1-T

- 1. Projeto de Unidade Lógico-Aritmética (ULA). Circuitos digitais e níveis de abstração. Representação de números inteiros em binário. Adição de números sem e com sinal, o somador paralelo *carry-ripple* e *overflow*.**

Prof. José Luís Güntzel
guntzel@inf.ufsc.br

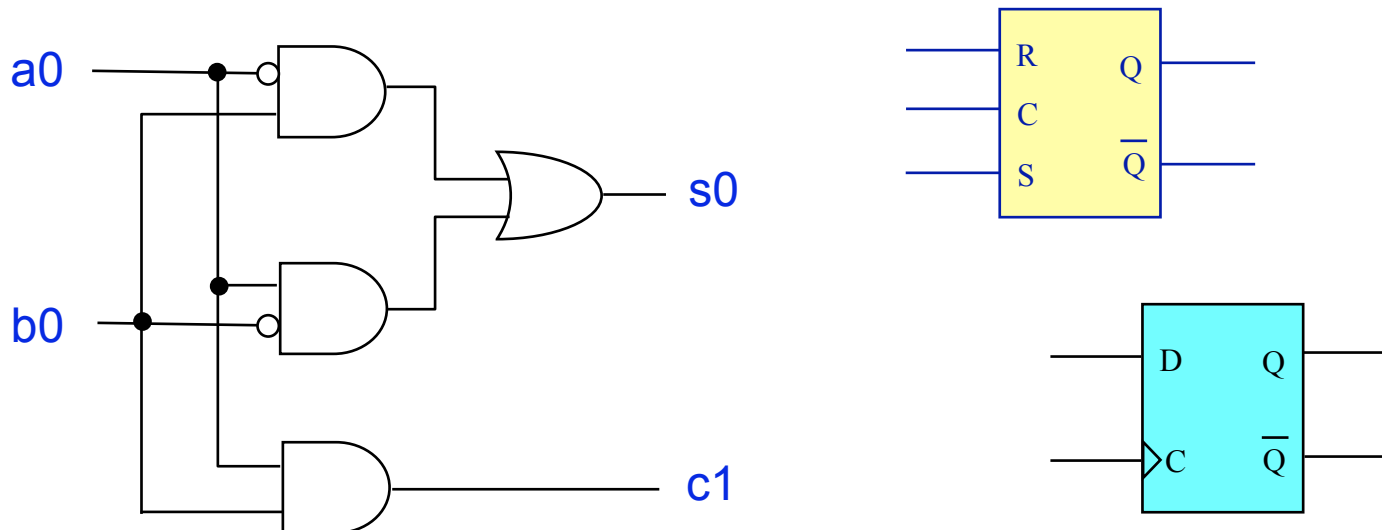
www.inf.ufsc.br/~guntzel/ine5406/ine5406.html

1. Projeto de Unidade Lógico-Aritmética

► Circuitos Digitais e Níveis de Abstração

Nível Lógico:

- Circuitos vistos como associações de portas lógicas, latches e flip-flops (conforme visto em EEL5105).
- Variáveis binárias de um bit cada.

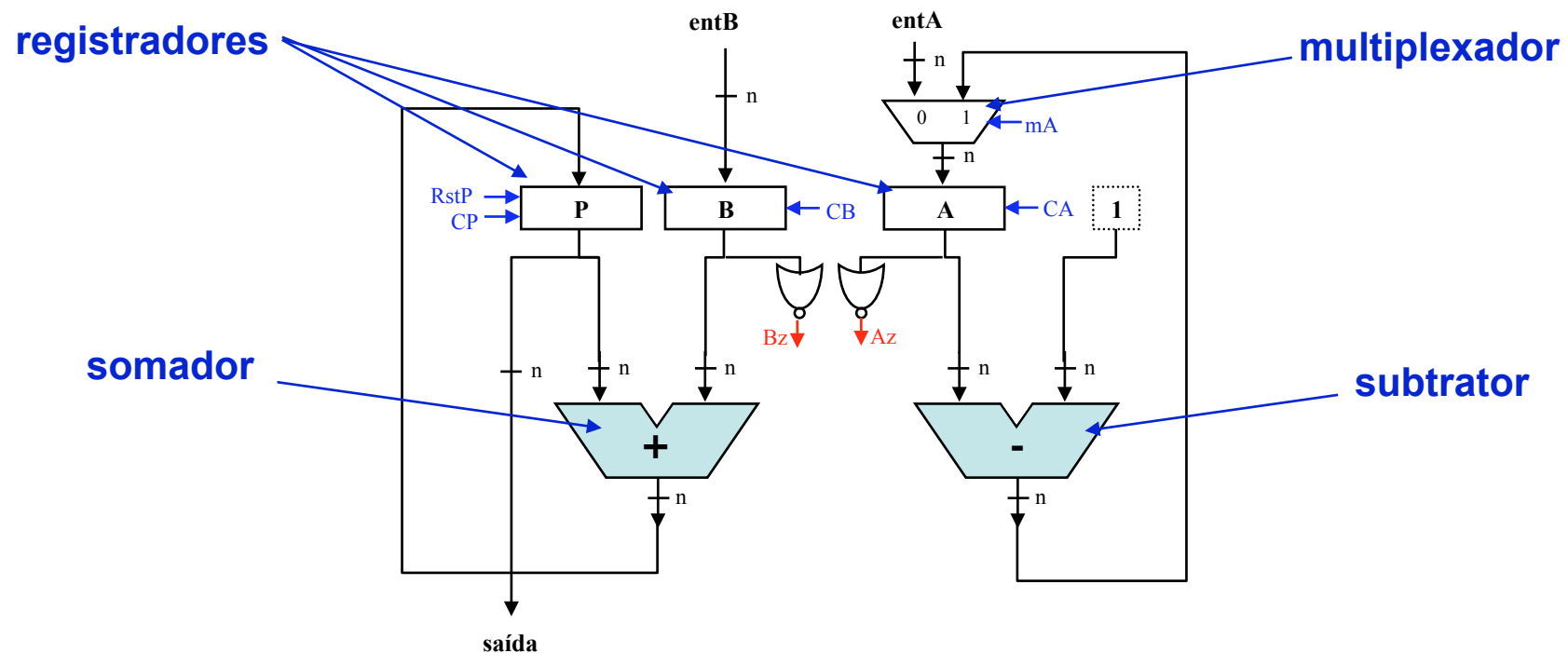


1. Projeto de Unidade Lógico-Aritmética

► Circuitos Digitais e Níveis de Abstração

Nível RT (Register Transfer):

- Circuitos vistos como associações de componentes, feitos de portas lógicas (objeto desta disciplina).

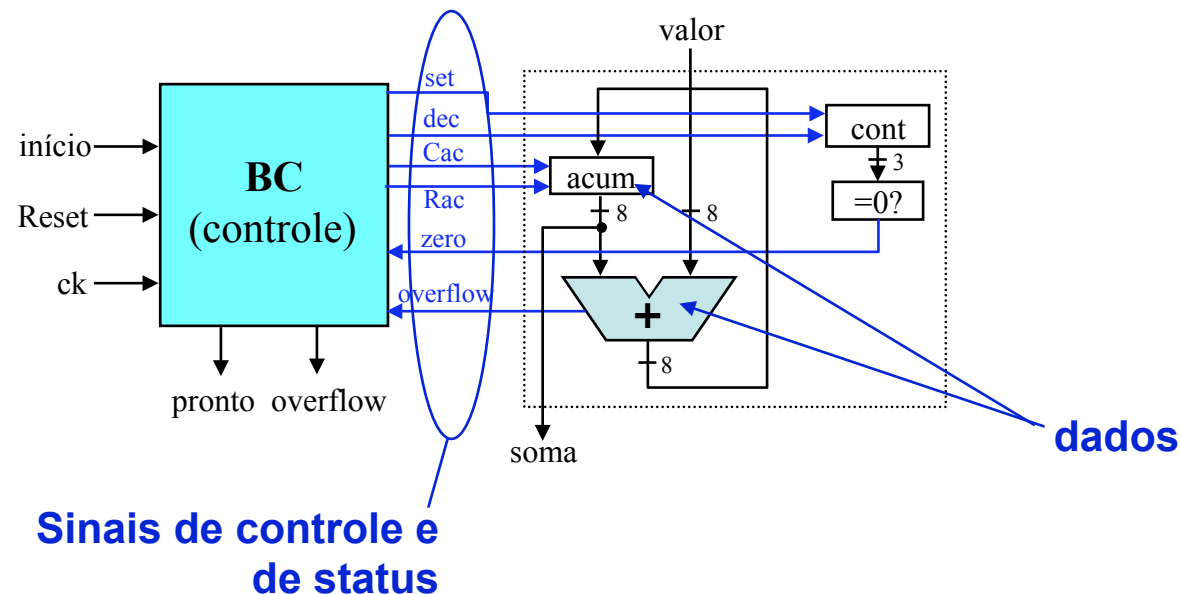


1. Projeto de Unidade Lógico-Aritmética

► Circuitos Digitais e Níveis de Abstração

Nível RT (Register Transfer):

- Operações lógicas e aritméticas sobre vetores de bits, os quais representam números.
- Sinais de controle e de status.



1. Projeto de Unidade Lógico-Aritmética

► Circuitos Digitais e Níveis de Abstração

Matéria	Nível de abstração	Componentes
Circuitos Digitais	Lógico	<ul style="list-style-type: none">• Portas lógicas• Latches• Flip-flops
Sistemas Digitais	RT (register-transfer)	<ul style="list-style-type: none">• Somadores• Subtratores• ULAs (= unidades funcionais compostas)• Registradores• Memórias• Multiplexadores• Decodificadores• Bloco de controle

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros sem Sinal (Naturais)

- assumindo-se números com 4 bits

	binário	decimal
Menor número	0000	0
Maior número	1111	15

Intervalo de representação: [0 , 15]

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros sem Sinal (Naturais)

Observe que:

$$\begin{aligned} 1111_2 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = \\ &= 1 + 2 + 4 + 8 = \\ &= \mathbf{15} \end{aligned}$$

Observe também que:

$$\begin{aligned} 10000_2 &= 0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 = \\ &= 0 + 0 + 0 + 0 + 16 = \\ &= \mathbf{16 (= 15 + 1)} \end{aligned}$$

Então, podemos nos referir ao **15** como **“2⁴ - 1”**, para efeitos de generalização.

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros sem Sinal (naturais)

- assumindo-se números com 8 bits

	binário	decimal
Menor número	00000000	0
Maior número	11111111	255

Intervalo de representação: [0 , 255]

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros sem Sinal (Naturais)

Observe que:

$$\begin{aligned} 11111111_2 &= \\ &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 = \\ &= 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = \mathbf{255} \end{aligned}$$

Observe também que:

$$\begin{aligned} 100000000_2 &= \\ &= 0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 0 \times 2^7 + 1 \times 2^8 = \\ &= 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 256 = \mathbf{256 (= 255 + 1)} \end{aligned}$$

Então, podemos nos referir ao **255** como **"2⁸ - 1"**, para efeitos de generalização.

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros sem Sinal (naturais)

- Generalizando-se para n bits

	binário	decimal
Menor número	0000...0	0
Maior número	1111...1	2^n-1

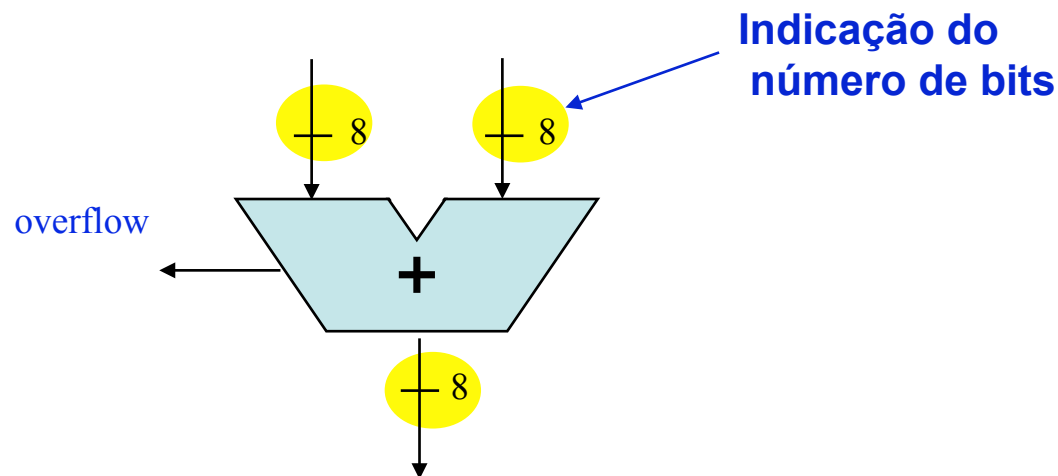
Intervalo de representação: $[0, 2^n-1]$

1. Projeto de Unidade Lógico-Aritmética

► Representando Dados em Circuitos Digitais

Exemplo: um circuito somador.

- Neste caso, cada operando tem 8 bits e o resultado é calculado com precisão de 8 bits. (Há um indicador de estouro de campo, *overflow*)

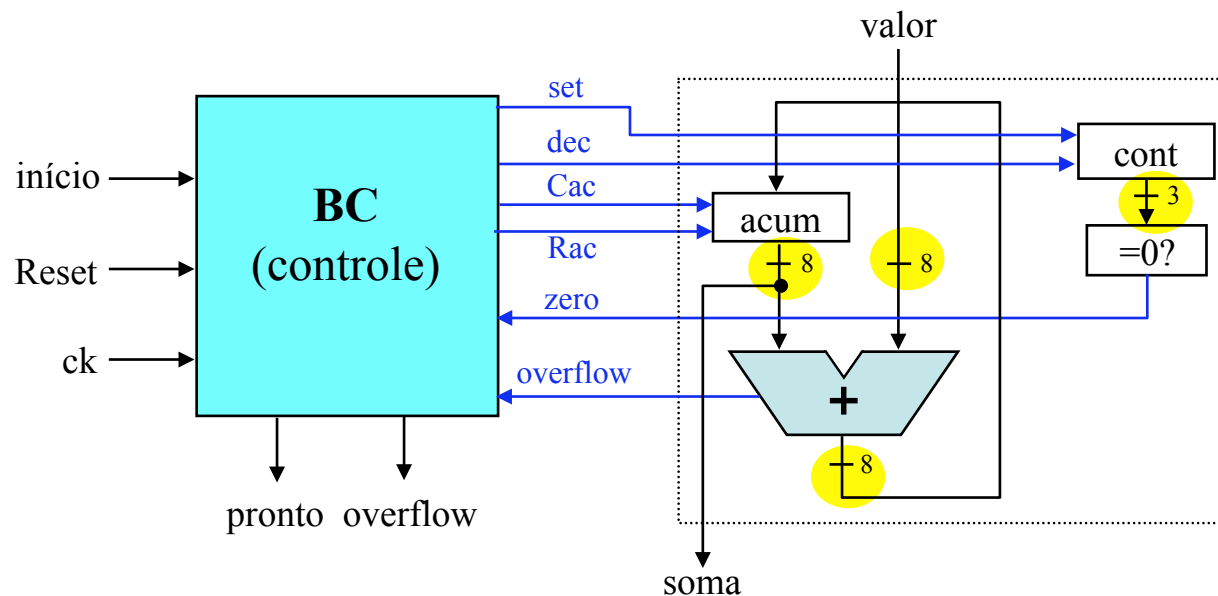


1. Projeto de Unidade Lógico-Aritmética

► Representando Dados em Circuitos Digitais

Exemplo: um circuito mais complexo.

- Fios que representam variáveis de mais de um bit devem ser identificados!

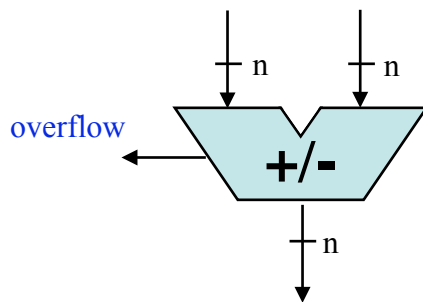


1. Projeto de Unidade Lógico-Aritmética

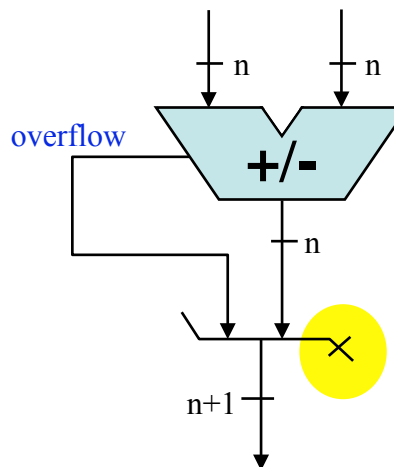
► Representando Dados em Circuitos Digitais

Convenções

Somador-subtrator para operandos com n bits cada



Indicando como um número de $n+1$ bits é composto



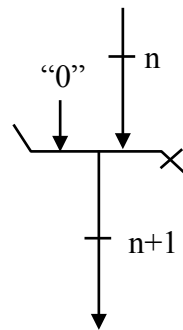
Convenção para indicar que o bit menos significativo está mais à direita

1. Projeto de Unidade Lógico-Aritmética

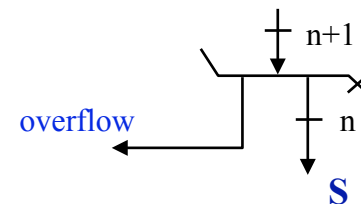
► Representando Dados em Circuitos Digitais

Convenções (continuação)

Indicando como um número de $n+1$ bits é composto (outro exemplo)



"Decompondo" um número de $n+1$ bits em um número de n bits e mais um sinal



1. Projeto de Unidade Lógico-Aritmética

► Circuitos Aritméticos

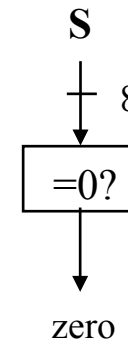
São Circuitos Combinacionais que operam sobre dados numéricos.

Exercício 1: Projetar um circuito combinacional capaz de testar se um número de 8 bits vale zero ou não. Este circuito deve operar conforme descrito na tabela abaixo.

Tabela de funcionamento:

zero	significado
0	$S \neq 0$
1	$S = 0$

Interfaces:



1. Projeto de Unidade Lógico-Aritmética

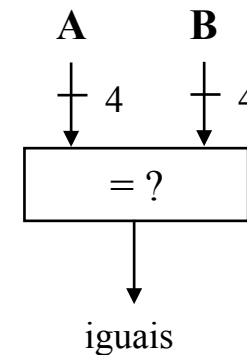
► Circuitos Aritméticos

Exercício 2: Projetar um circuito combinacional que compara dois números de 4 bits cada. Este circuito deve operar conforme descrito na tabela abaixo.

Tabela de funcionamento:

iguais	significado
0	$A \neq B$
1	$A = B$

Interfaces:



1. Projeto de Unidade Lógico-Aritmética

► Circuitos Aritméticos

Exercício 2: solução...

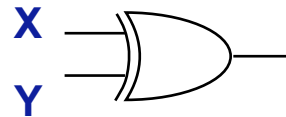
Função XOR (Exclusive OR, “OU” Exclusivo)

- Resulta “1” se as duas entradas forem iguais...

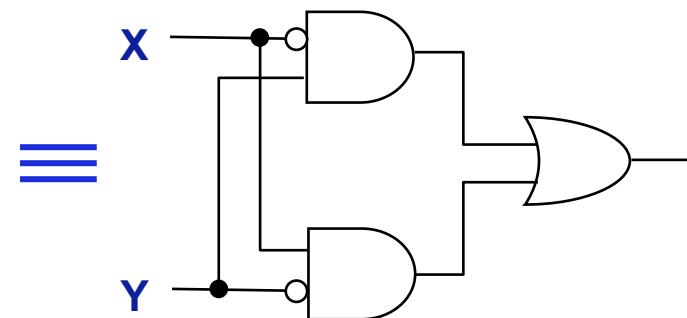
X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

$$X \oplus Y = \bar{X} \cdot Y + X \cdot \bar{Y}$$

Porta lógica



Circuito em soma de produtos



1. Projeto de Unidade Lógico-Aritmética

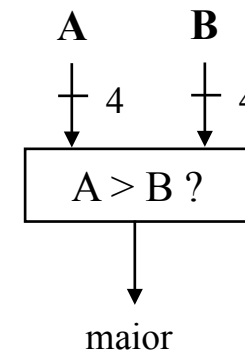
► Circuitos Aritméticos

Exercício 3: Projetar um circuito combinacional que compara dois números de 4 bits cada. Este circuito deve operar conforme descrito na tabela abaixo.

Tabela de funcionamento:

maior	significado
0	$A \leq B$
1	$A > B$

Interfaces:



1. Projeto de Unidade Lógico-Aritmética

► Circuitos Aritméticos

Qual é a operação aritmética mais importante?

Resposta: a adição!

Por quê?

Porque ela serve de base para outras operações aritméticas mais frequentes nos sistemas digitais.

1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

Adição de Números Sem Sinal

Exemplo 1:

Notação genérica

Análise supondo A e B com 4 bits

$$\begin{array}{r} A \\ + B \\ \hline S \end{array}$$

$$\begin{array}{r} \text{transportes (carries)} \\ 0100 \\ 0110 \quad (6) \\ + 0100 \quad (4) \\ \hline 1010 \quad (10) \quad \text{resultado} \end{array}$$

Supondo um somador que trabalhe com operandos de 4 bits, então o resultado S deverá pertencer ao intervalo [0, 15]

1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

Adição de Números Sem Sinal

Exemplo 2:

Diagram illustrating the addition of two 4-bit numbers A and B to produce a 5-bit result S. The carry-out (Cout) is 1, indicating an overflow.

A	1 1 0 0	(12)
+ B	0 1 1 0	(6)
<hr/>		
S	1 0 0 1 0	(18)

The carry-out (Cout) is 1, indicating an overflow.

Caso o resultado não puder ser representado com 4 bits, o sinal “Cout” indicará que houve um “estouro de representação” (em inglês, *overflow*).

1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

Generalizando a Adição de Números Sem Sinal

Notação genérica

$$\begin{array}{r} A \\ + B \\ \hline S \end{array}$$

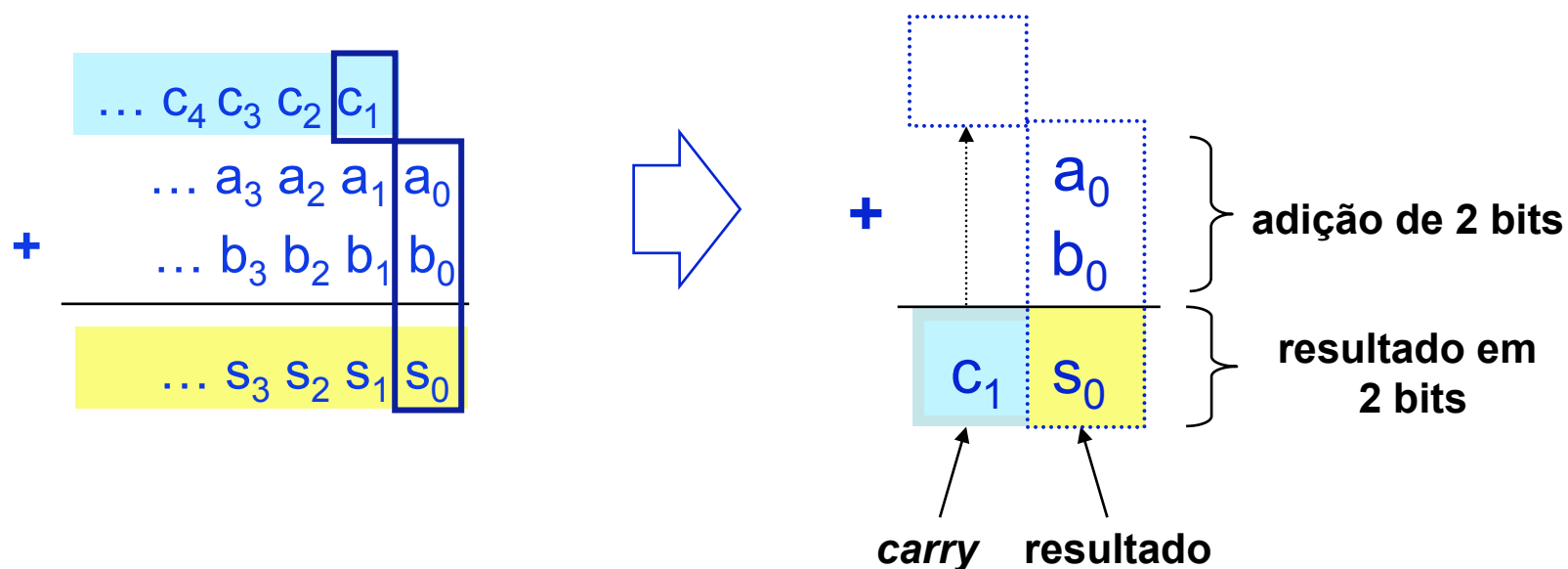
Análise detalhada

$$\begin{array}{r} \dots c_4 c_3 c_2 c_1 \quad \text{transportes (carries)} \\ \dots a_3 a_2 a_1 a_0 \\ + \dots b_3 b_2 b_1 b_0 \\ \hline \dots s_3 s_2 s_1 s_0 \quad \text{resultado} \end{array}$$

1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

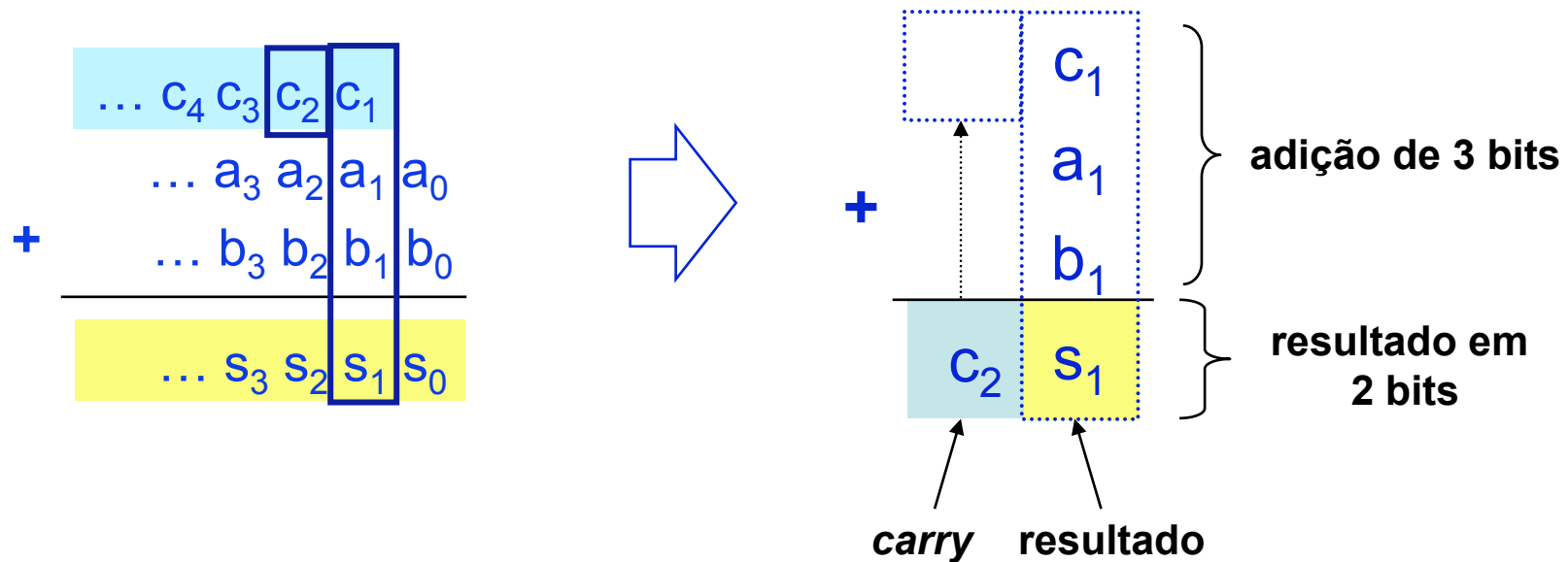
Adicionando os bits menos significativos



1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

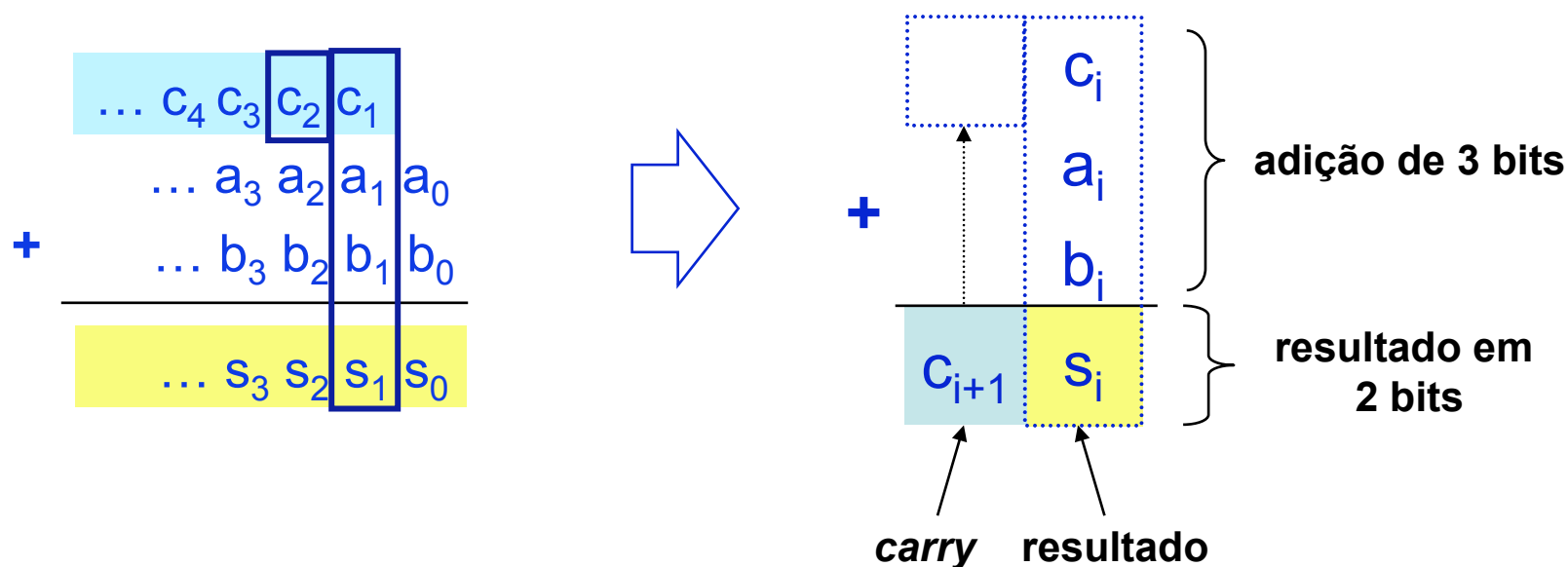
Porém, a partir do 2ª coluna ...



1. Projeto de Unidade Lógico-Aritmética

► Revisão de Adição Binária

Generalizando, a partir do 2ª coluna ...

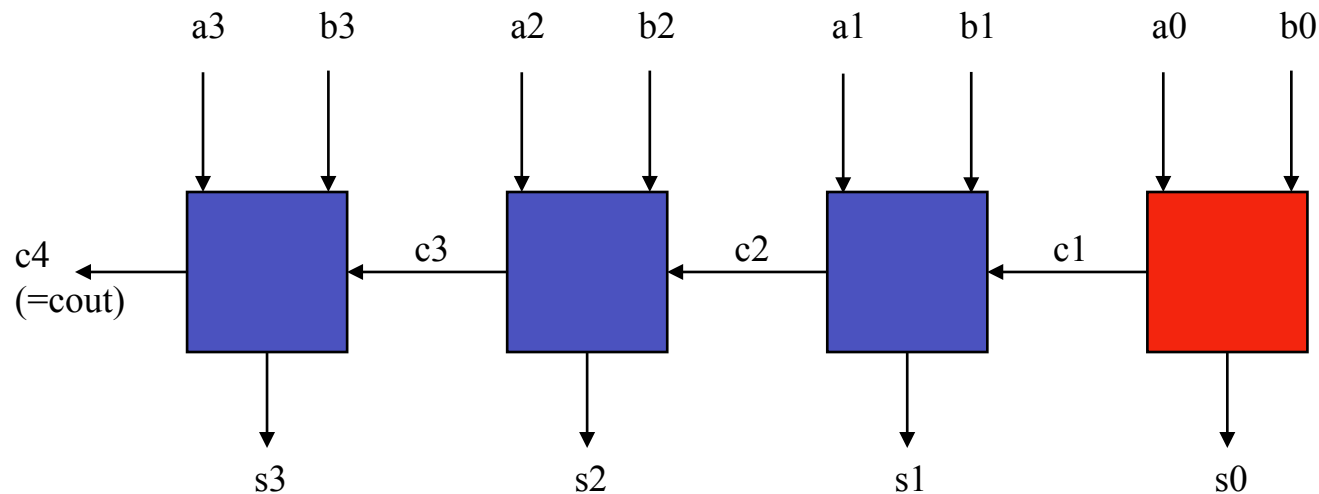


Obs: $i \geq 1$

1. Projeto de Unidade Lógico-Aritmética

► Esquema da Adição Paralela

Considerando dois números (A e B) com 4 bits cada

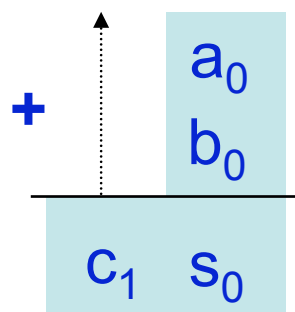
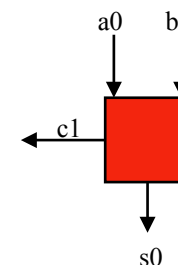


- Há um elemento para cada coluna da soma.
- O sinal de transporte mais à esquerda (c_4 , neste caso), também recebe o nome de *cout*.
- Se este somador operar sobre inteiros sem sinal, então *cout* também servirá para indicar a ocorrência de *overflow*

1. Projeto de Unidade Lógico-Aritmética

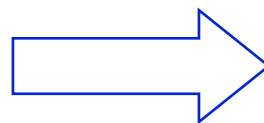
► Somador Para a Primeira Coluna

Este circuito é conhecido por “Meio Somador” (*Half-Adder*, em inglês)



Criação da tabela-verdade:

- Listar todas as combinações de entradas (a_0 , b_0)
- Preencher os valores das saídas s_0 e c_1 baseado no resultado da adição entre a_0 e b_0



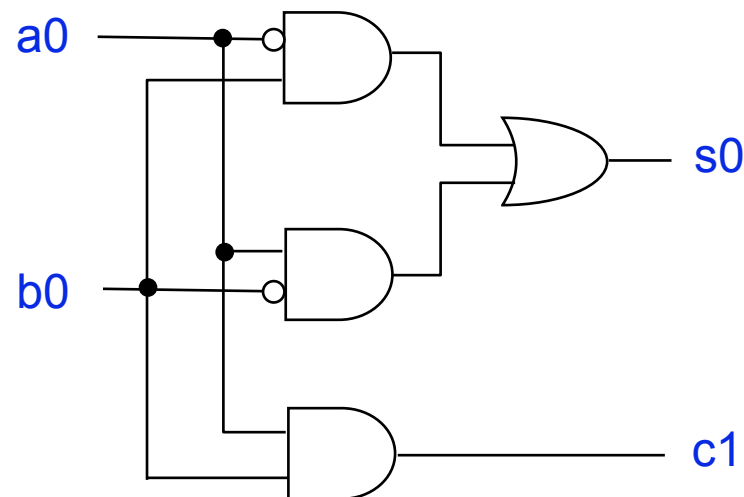
entradas		saídas	
a_0	b_0	c_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

1. Projeto de Unidade Lógico-Aritmética

► O Meio Somador

entradas		saídas	
a0	b0	c1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$s0 = \overline{a0} \cdot b0 + a0 \cdot \overline{b0} = a0 \oplus b0$$
$$c1 = a0 \cdot b0$$

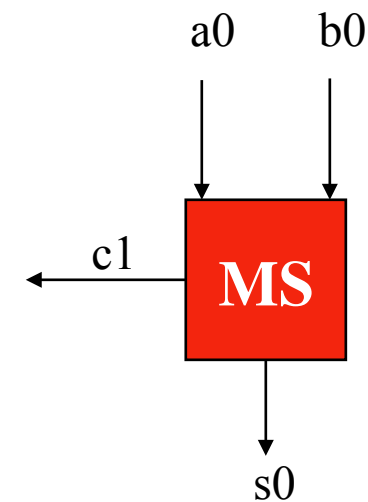
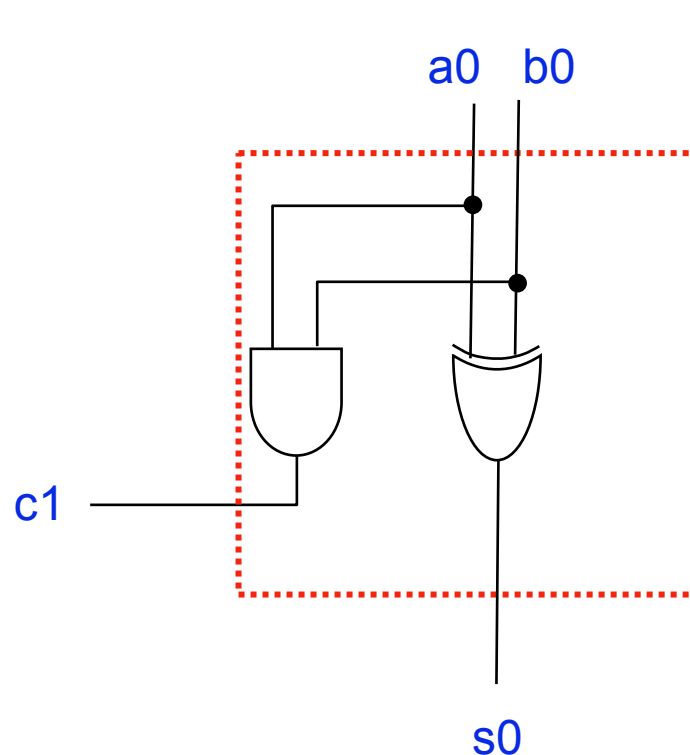


Obs: circuito independente de tecnologia de fabricação

1. Projeto de Unidade Lógico-Aritmética

► O Meio Somador

Redesenhando, usando porta XOR

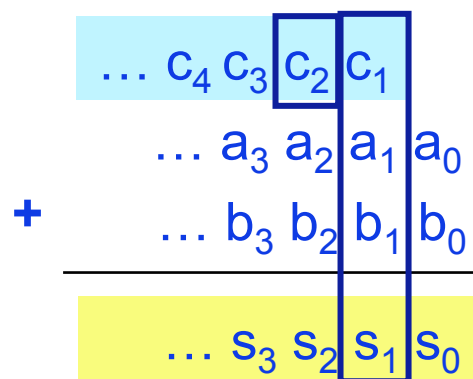
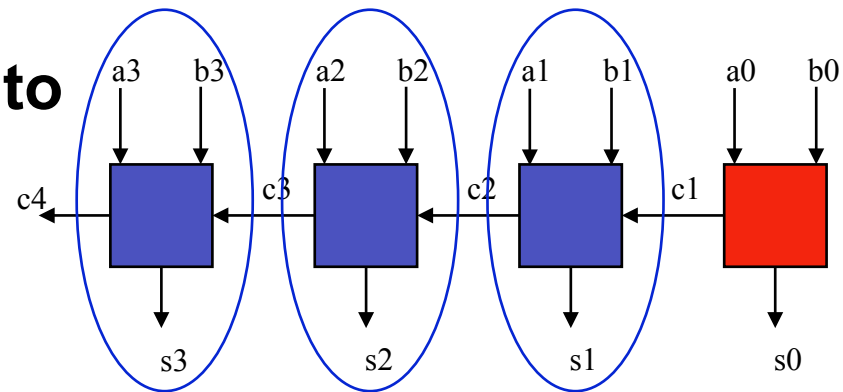


$$s0 = \overline{a0} \cdot b0 + a0 \cdot \overline{b0} = a0 \oplus b0$$
$$c1 = a0 \cdot b0$$

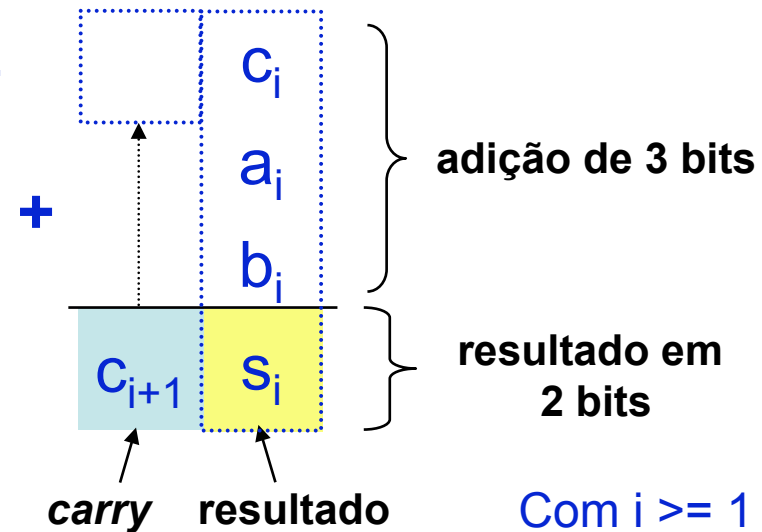
1. Projeto de Unidade Lógico-Aritmética

► Somador Para as Demais Colunas

Projetando um tipo de circuito para as demais colunas:



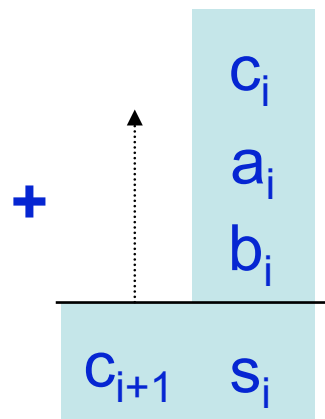
Generalizando...



1. Projeto de Unidade Lógico-Aritmética

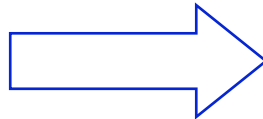
► Somador Para as Demais Colunas

Este circuito é conhecido como “Somador Completo” (*Full-Adder*, em inglês)



Criação da tabela-verdade:

- Listar todas as combinações de entradas (c_i , a_i , b_i)
- Preencher os valores das saídas s_i e c_{i+1} baseado no resultado da adição entre a_i , b_i e c_i



entradas			saídas	
c_i	a_i	b_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

1. Projeto de Unidade Lógico-Aritmética

► O Somador Completo

entradas			saídas	
c_i	a_i	b_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

Mapa de Karnaugh para C_{i+1}

c_{i+1}	$\overline{a_i} \overline{b_i}$	$\overline{a_i} b_i$	$a_i b_i$	$a_i \overline{b_i}$	
$\overline{c_i}$	0	0	1	0	$a_i \cdot b_i$
c_i	0	1	1	1	$a_i \cdot c_i$

$b_i \cdot c_i$

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

1. Projeto de Unidade Lógico-Aritmética

► O Somador Completo

entradas			saídas	
ci	ai	bi	ci+1	si
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

Mapa de Karnaugh para s_i

s_i	$\overline{a_i} \overline{b_i}$	$\overline{a_i} b_i$	$a_i b_i$	$a_i \overline{b_i}$
$\overline{c_i}$	0	1	0	1
c_i	1	0	1	0

Não é possível simplificar, logo, usaremos todos os produtos do tipo mintermo!

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

1. Projeto de Unidade Lógico-Aritmética

► O Somador Completo

Manipulando a expressão para si

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

$$= \overline{c_i} (\underbrace{\overline{a_i} \cdot b_i + a_i \cdot \overline{b_i}}) + c_i (\underbrace{\overline{a_i} \cdot \overline{b_i} + a_i \cdot b_i})$$

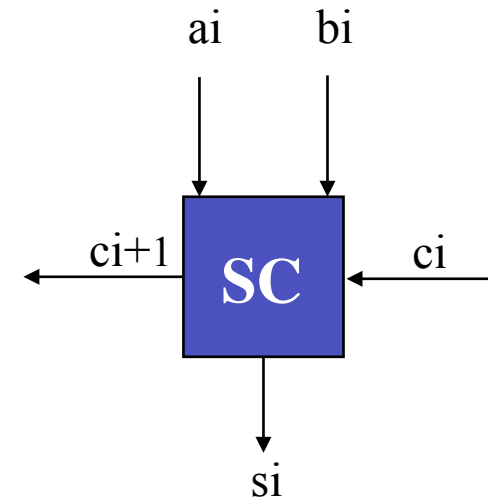
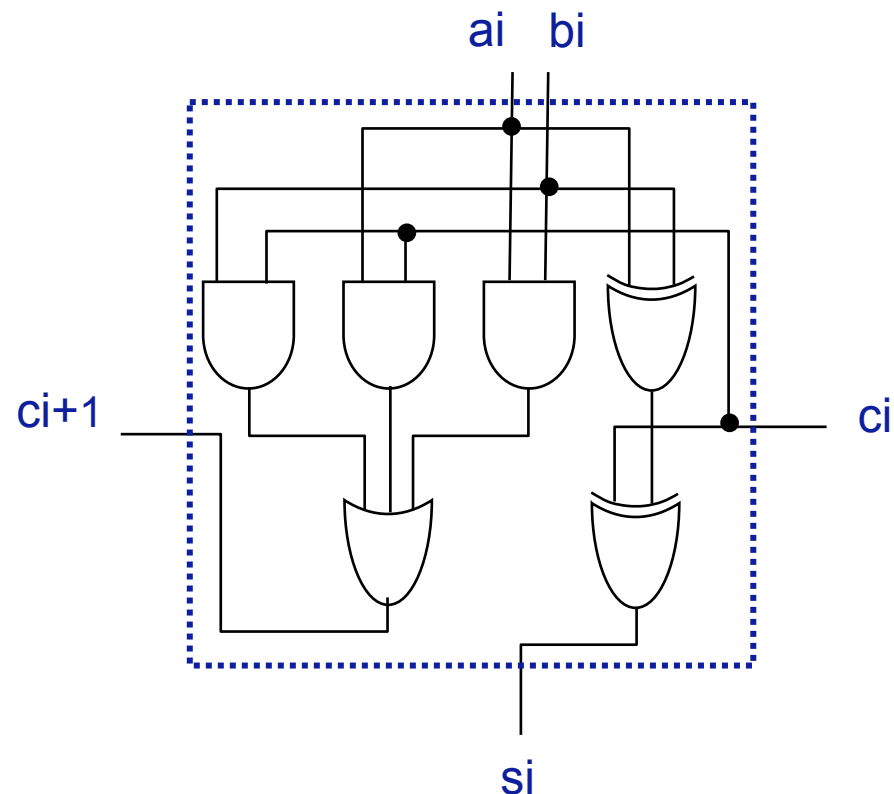
$$= \overline{c_i} (a_i \oplus b_i) + c_i (\overline{a_i \oplus b_i})$$

$$= c_i \oplus a_i \oplus b_i$$

Normalmente, se assumem portas xor com duas entradas, as quais podem ser fabricadas diretamente usando tecnologia CMOS.

1. Projeto de Unidade Lógico-Aritmética

► O Somador Completo



$$s_i = c_i \oplus a_i \oplus b_i$$

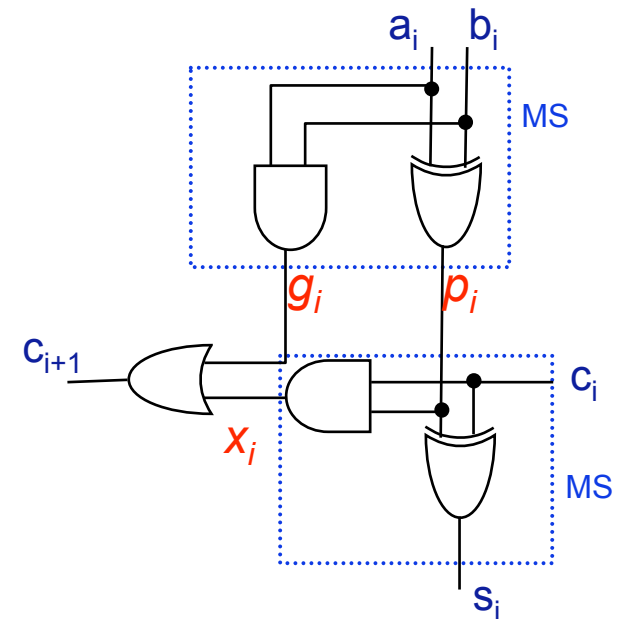
$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$

1. Projeto de Unidade Lógico-Aritmética

► O Somador Completo

Uma Outra Versão, usando dois MS...

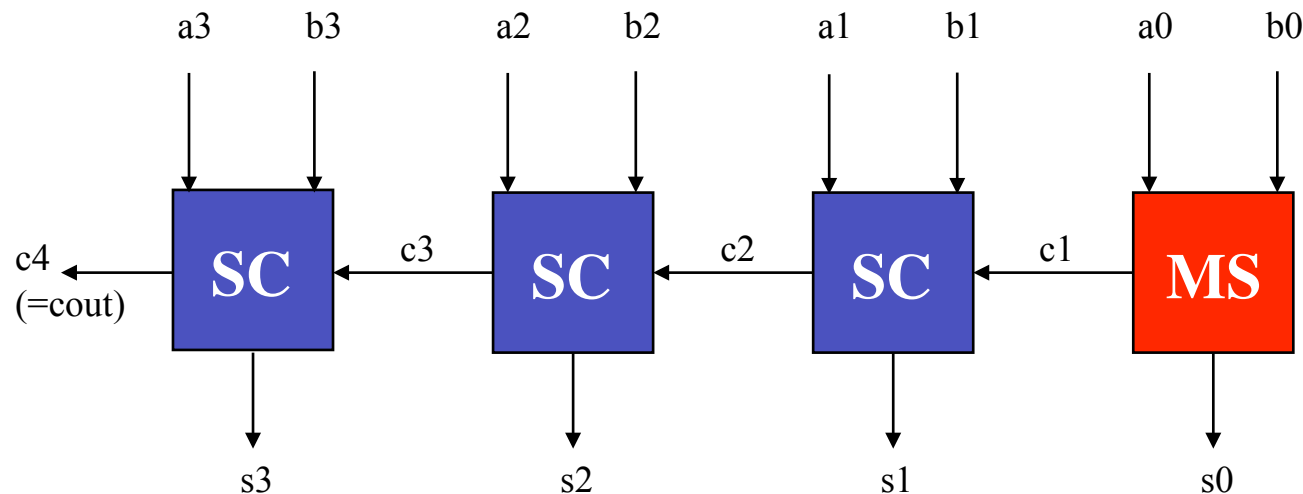
ai	bi	ci	<i>pi</i>	<i>xi</i>	<i>gi</i>	ci+1	si
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	1	1	0	1	0
1	0	0	1	0	0	0	1
1	0	1	1	1	0	1	0
1	1	0	0	0	1	1	0
1	1	1	0	0	1	1	1



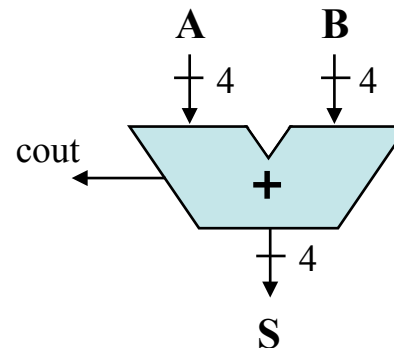
Vantagem: necessita menos portas lógicas

1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits) Diagrama de Blocos (Nível Lógico)

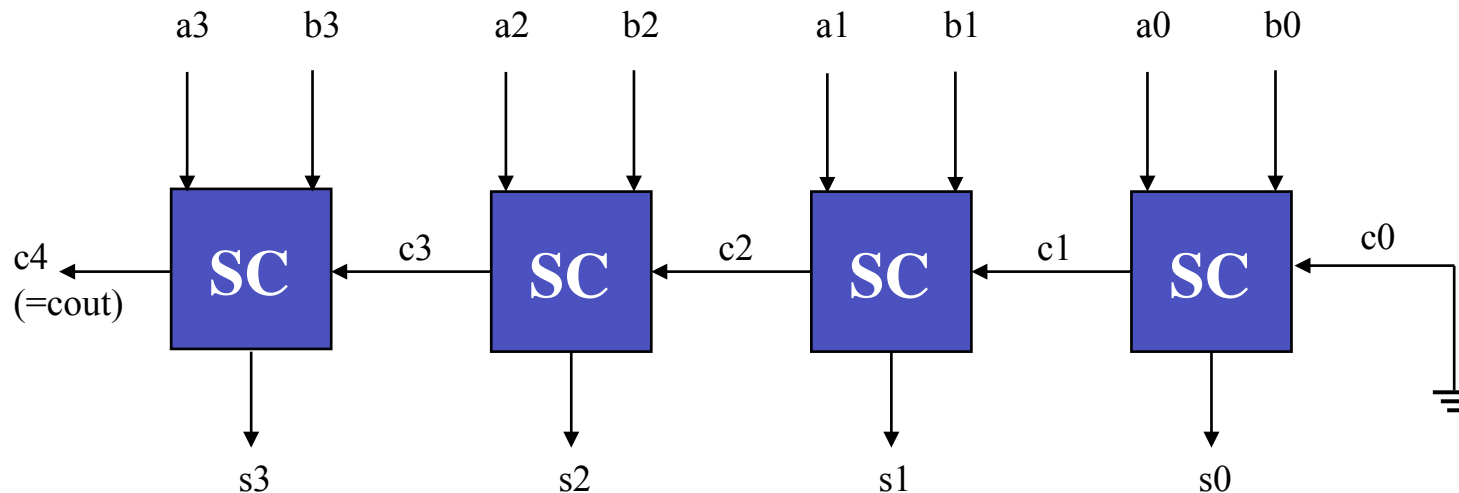


Símbolo no Nível RT

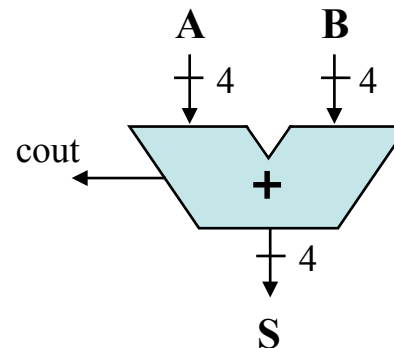


1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits) Diagrama de Blocos (Nível Lógico): versão 2



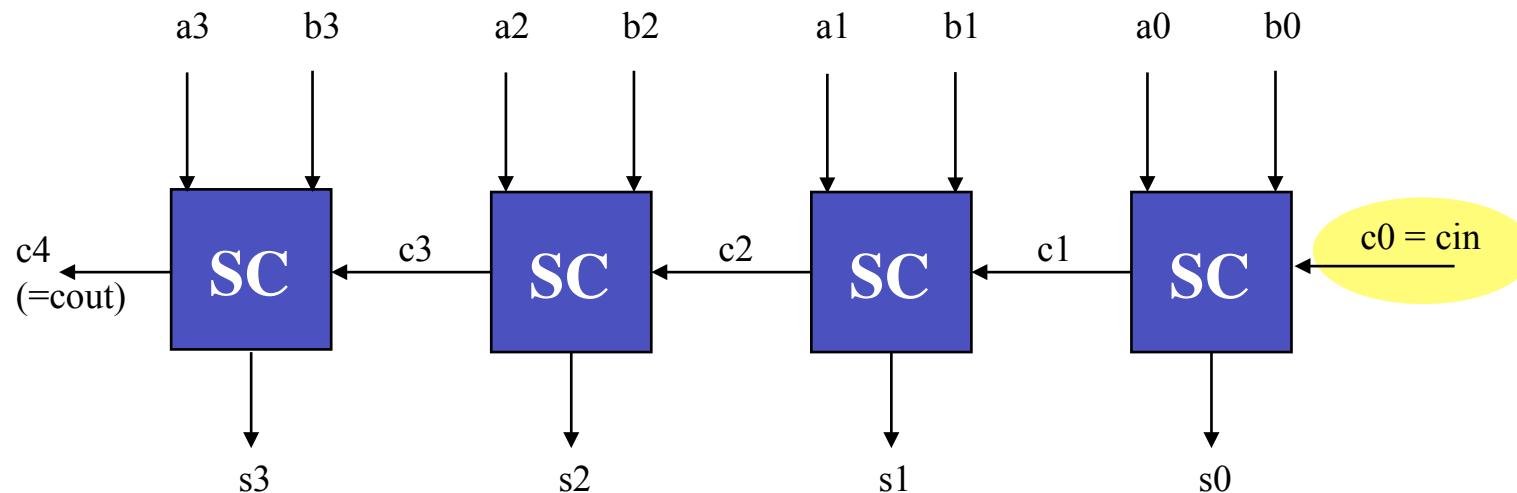
Símbolo no Nível RT



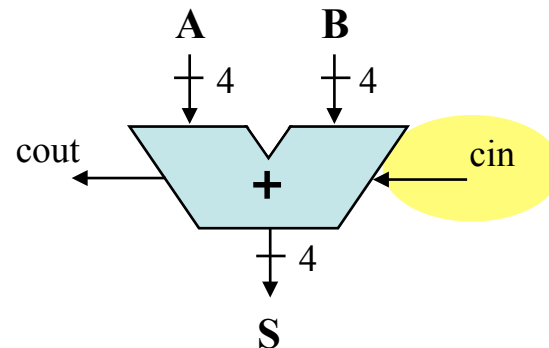
1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits)

Diagrama de Blocos (Nível Lógico): versão 3



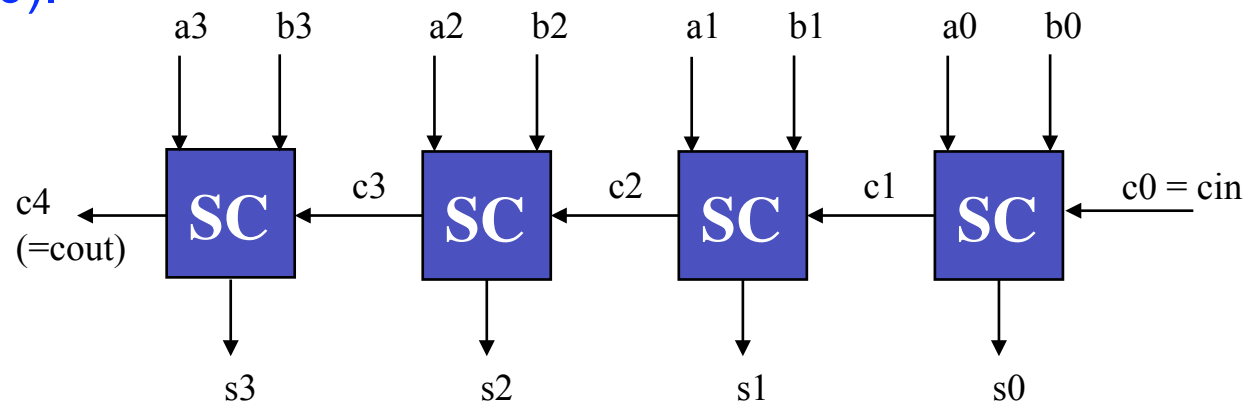
Símbolo no Nível RT



1. Projeto de Unidade Lógico-Aritmética

► Somador *Carry-Ripple*: Custo x Desempenho

- A estrutura do somador *carry-ripple* baseia-se na fatoração da expressão do *carry*. A consequência disto é:
 - Redução do custo (i.e., menor número de portas lógicas)
 - Aumento do atraso (o cálculo de c_4 depende de c_3 , que depende de c_2 , que depende de c_1 , que depende apenas das entradas c_0 , a_0 e b_0).



Os slides complementares “somadores rápidos” mostra outros tipos de somadores.

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Como Representar Inteiros Negativos?

Sinal-magnitude

sinal
↓
+3 = 0 0 1 1
-3 = 1 0 1 1

Complemento de 1

sinal
↓
+3 = 0 0 1 1
-3 = 1 1 0 0 ← { Obtido a partir do +3,
aplicando-se a inversão
bit a bit

Complemento de 2

sinal
↓
+3 = 0 0 1 1
-3 = 1 1 0 1 ← { Obtido a partir do +3, aplicando
-se a inversão bit a bit e após,
somando-se 1 (uma unidade)

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros com Sinal

- Para facilitar a construção de circuitos aritméticos, os negativos são representados em complemento de dois
- Assumindo-se números com **4 bits**

	binário	decimal
Menor número	1000	-8
Zero	0000	0
Maior número	0111	+7

Intervalo de representação: [-8 , +7]

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros com Sinal

Observe que:

$$\begin{aligned} 0111_2 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 = \\ &= 1 + 2 + 4 + 0 = \\ &= \mathbf{+7} \end{aligned}$$

Observe também que:

$$\begin{aligned} 1111_2 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 - 1 \times 2^3 = \\ &= 1 + 2 + 4 - 8 = \\ &= \mathbf{-1} \end{aligned}$$

O bit mais à esquerda representa o sinal: se ele valer 1, o número é negativo. Caso contrário, o número é positivo ou zero.

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros com Sinal

- assumindo-se números com 8 bits

	binário	decimal
Menor número	10000000	-128
Zero	00000000	0
Maior número	01111111	+127

Intervalo de representação: [-128 , +127]

1. Projeto de Unidade Lógico-Aritmética

► Representação de Inteiros em Binário

Números Inteiros com Sinal

- Generalizando-se para n bits

	binário	decimal
Menor número	1000...0	-2^{n-1}
Zero	0000...0	0
Maior número	0111...1	$+(2^{n-1}-1)$

Intervalo de representação: $[-2^{n-1}, +(2^{n-1}-1)]$

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal

Assumindo:

- Negativos são representados em complemento de dois
- Números com **4 bits**

	binário	decimal
Menor número	1000	-8
Zero	0000	0
Maior número	0111	+7

Intervalo de representação: [-8 , +7]

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 3: dois números positivos, cuja soma $\in [-8,+7]$

	0 0 0 0	transporte (<i>carry</i>)
	0 0 1 0 (+2)	
+	0 1 0 0 (+4)	
<hr/>		
	0 1 1 0 (+6)	resultado correto

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 4: dois números negativos, cuja soma seja ≥ -8

Apesar deste último *carry* valer 1, não houve *overflow*

	1	1	0	0		transporte (<i>carry</i>)
		1	1	1	0	(-2)
+		1	1	0	0	(-4)
<hr/>						
	1	0	1	0		(-6) resultado correto

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 5: um número positivo e um número negativo, tais que o resultado é positivo

Novamente...

	1	1	1	1		transporte (<i>carry</i>)
		0	1	1	1	(+7)
+		1	1	1	1	(-1)
<hr/>						
	0	1	1	0		(+6) resultado correto

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 6: um número positivo e um número negativo, tais que o resultado é negativo

	0 0 0 1	transporte (<i>carry</i>)
	1 0 0 1 (-7)	
+	0 0 0 1 (+1)	
<hr/>		
	1 0 1 0 (-6)	resultado correto

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 7: um positivo e um negativo, iguais em módulo

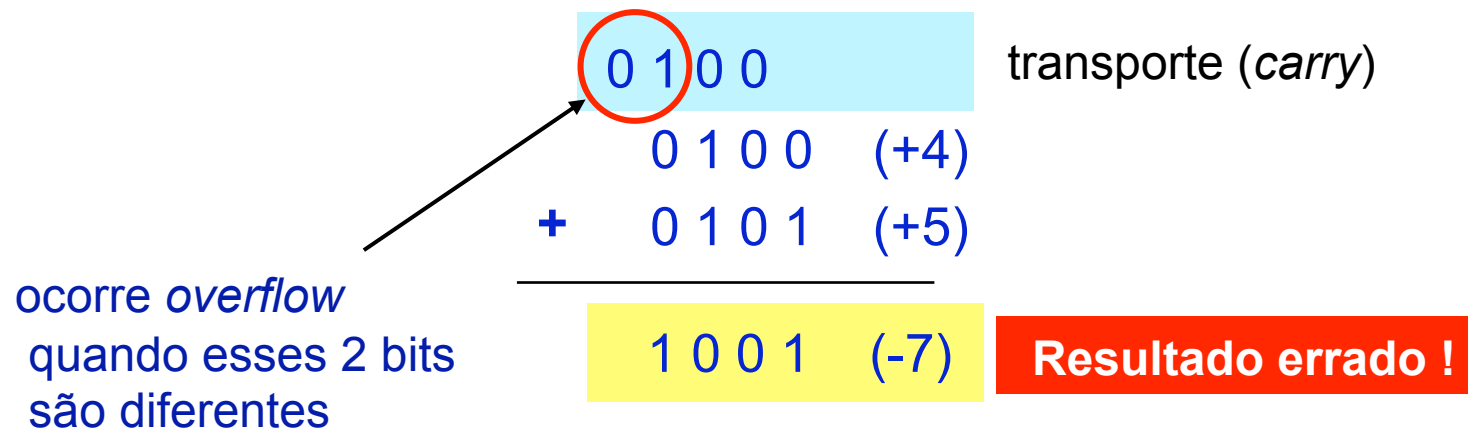
E novamente...

	1	1	1	1	transporte (<i>carry</i>)
	0	1	0	1	(+5)
+	1	0	1	1	(-5)
<hr/>					
	0	0	0	0	(0) resultado correto

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 8: 2 números positivos



o resultado excede o intervalo de representação = *overflow*

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal (Assumindo Negativos em Complemento de 2)

Exemplo 9: 2 números negativos

ocorre *overflow*
quando esses 2 bits
são diferentes

	1	0	0	0	transporte (<i>carry</i>)
	1	1	0	0	(-4)
+	1	0	1	1	(-5)
<hr/>					
	0	1	1	1	(+7)

Resultado errado !

o resultado excede o intervalo de representação = *overflow*

1. Projeto de Unidade Lógico-Aritmética

► Adição de Inteiros com Sinal

(Assumindo Negativos em Complemento de 2)

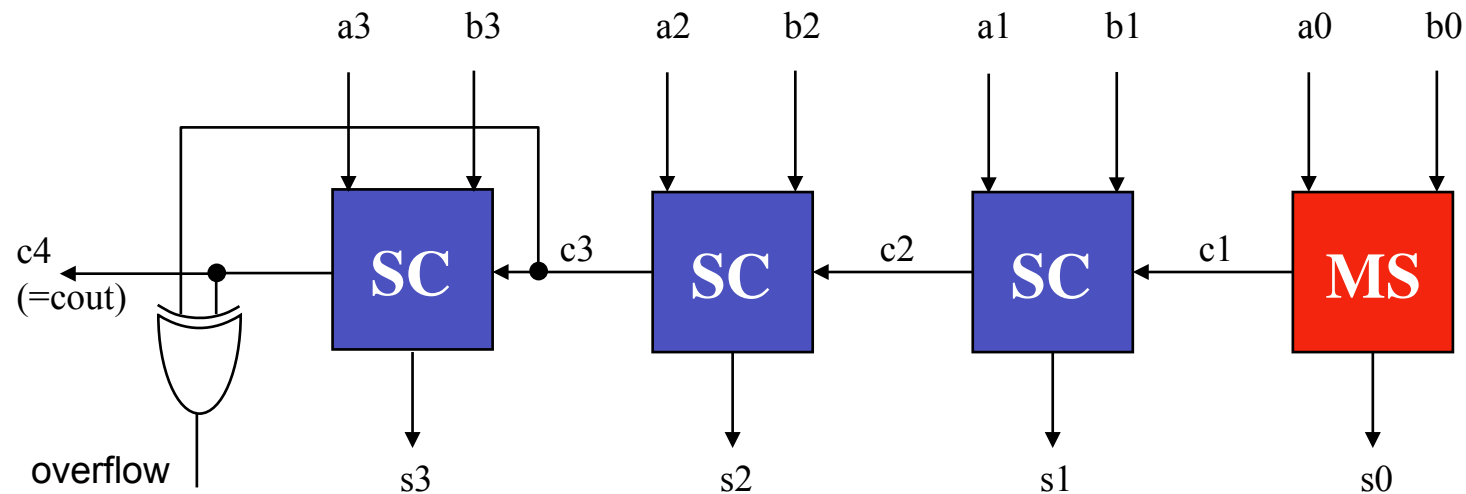
Conclusões:

- Números binários em **complemento de 2** podem ser adicionados como se fossem números binários sem sinal!
- Neste caso, a detecção de **overflow** se dá comparando-se os dois últimos sinais de *carry*

1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits) Modificado para Operar Sobre Números com Sinal (Assumindo negativos em complemento de 2)

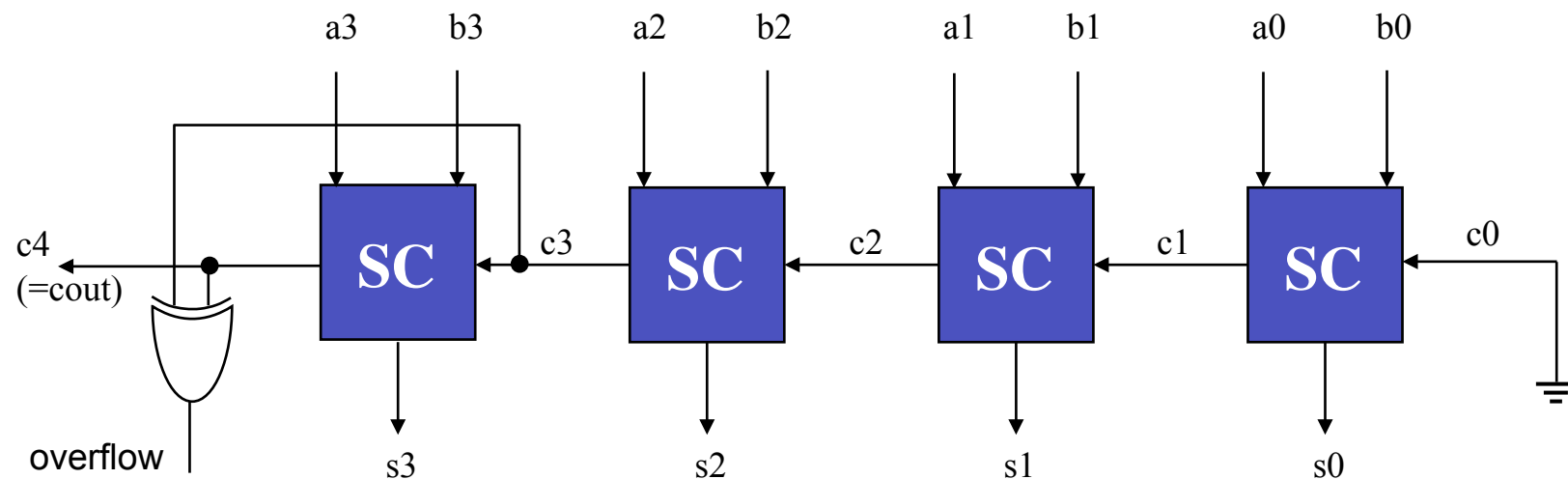
Diagrama de Blocos (Nível Lógico)



1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits) Modificado para Operar Sobre Números com Sinal (Assumindo negativos em complemento de 2)

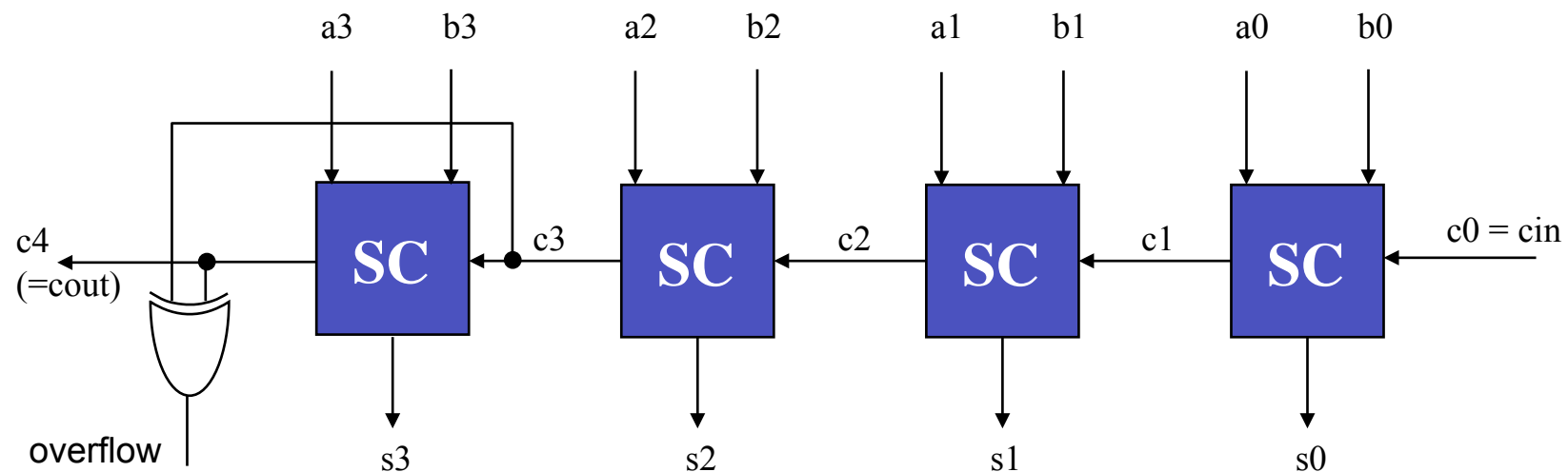
Diagrama de Blocos (Nível Lógico): versão 2



1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits) Modificado para Operar Sobre Números com Sinal (Assumindo negativos em complemento de 2)

Diagrama de Blocos (Nível Lógico): versão 3



1. Projeto de Unidade Lógico-Aritmética

► O Somador Paralelo *Carry-Ripple* (de 4 Bits)

Símbolos no Nível RT

