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

Sistemas Digitais

INE 5406

Revisão

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

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:

$1111_2 = 1x2^0 + 1x2^1 + 1x2^2 + 1x2^3 =$

$= 1 + 2 + 4 + 8 =$

$=$ **15**

Observe também que:

$10000_2 = 0x2^0 + 0x2^1 + 0x2^2 + 0x2^3 + 1x2^4 =$

$= 0 + 0 + 0 + 0 + 16 =$

$=$ **16 (= 15 + 1)**

Então, podemos nos referir ao 15 como "2^4 - 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:

11111111_2 =

$= 1x2^0 + 1x2^1 + 1x2^2 + 1x2^3 + 1x2^4 + 1x2^5 + 1x2^6 + 1x2^7 =$

$= 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 =$ **255**

Observe também que:

100000000_2 =

$= 0x2^0 + 0x2^1 + 0x2^2 + 0x2^3 + 0x2^4 + 0x2^5 + 0x2^6 + 0x2^7 + 1x2^8 =$

$= 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 256 =$ **256 (= 255 + 1)**

Então, podemos nos referir ao 255 como "2^8 - 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

Revisão de Adição Binária

Adição de Números Sem Sinal

Exemplo 1:

Notação genérica **Análise supondo A e B com 4 bits**

		0 1 0 0		transportes (*carries*)
	A	0 1 1 0	(6)	
+	B	+ 0 1 0 0	(4)	
	S	1 0 1 0	(10)	resultado

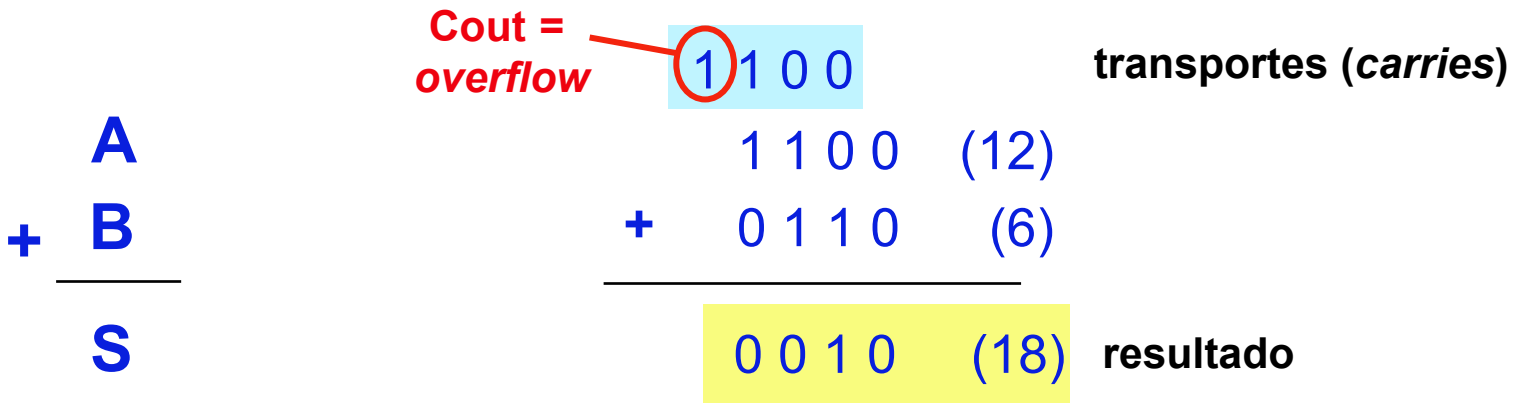
Supondo um somador que trabalhe com operandos de 4 bits, então o resultado S deverá pertencer ao intervalo [0, 15]

1. Projeto de Unidade Lógico-Aritmética

Revisão de Adição Binária

Adição de Números Sem Sinal

Exemplo 2:



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

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

Análise detalhada

$$\begin{array}{rr} & \ldots\ c_4\ c_3\ c_2\ c_1 \\ & \ldots\ a_3\ a_2\ a_1\ a_0 \\ + & \ldots\ b_3\ b_2\ b_1\ b_0 \\ \hline & \ldots\ s_3\ s_2\ s_1\ s_0 \end{array}$$

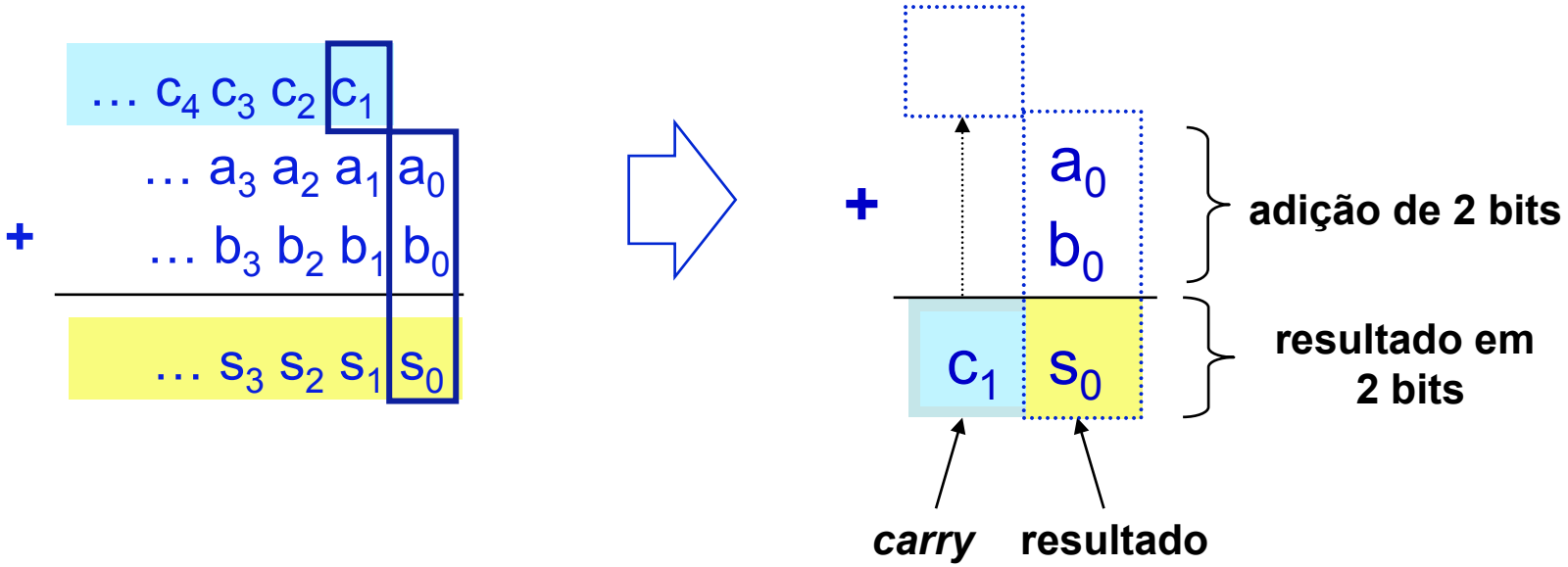
transportes (*carries*)

resultado

1. Projeto de Unidade Lógico-Aritmética

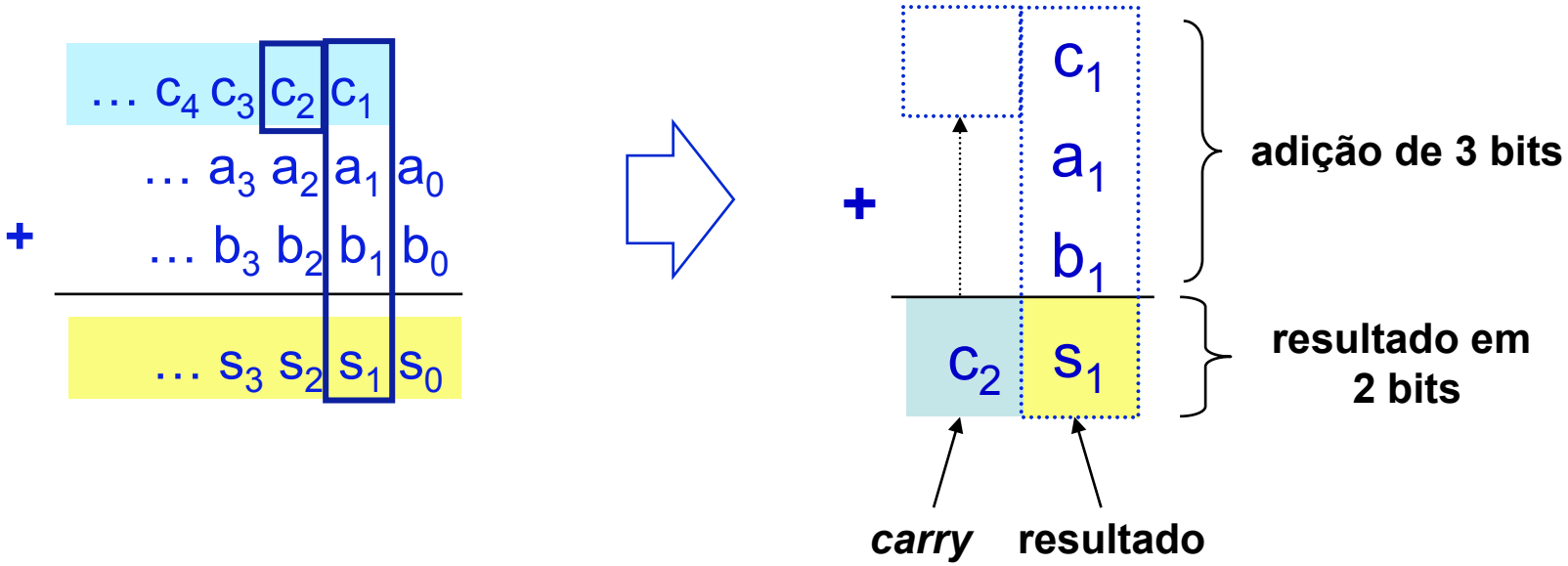
Revisão de Adição Binária

Adicionando os bits menos significativos



Revisão de Adição Binária

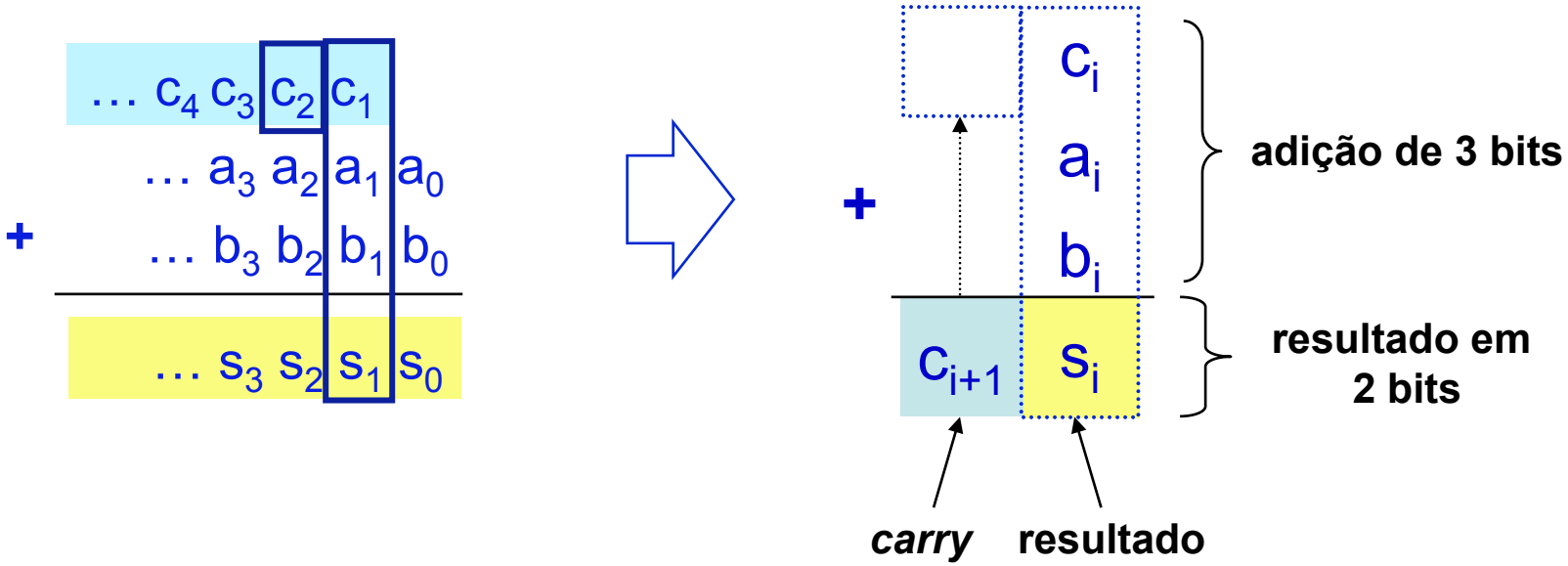
Porém, a partir do 2ª coluna …



1. Projeto de Unidade Lógico-Aritmética

Revisão de Adição Binária

Generalizando, a partir do 2ª coluna …

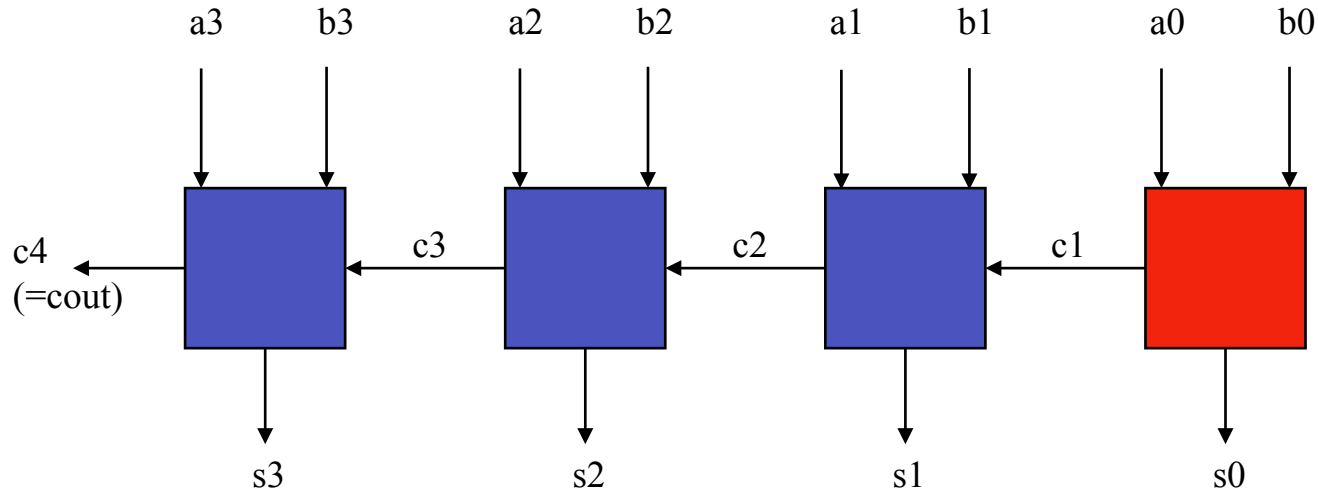


Obs: i >= 1

1. Projeto de Unidade Lógico-Aritmética

Esquema da Adição Paralela

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

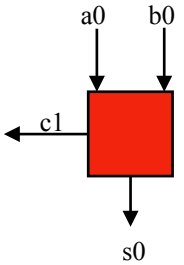


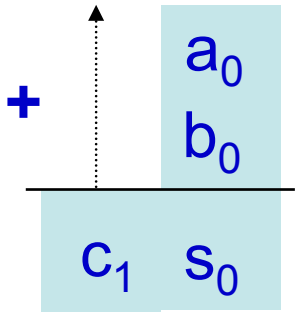
- **Há um elemento para cada coluna da soma.**
- **O sinal de transporte mais à esquerda (c4, neste caso), também recebe o nome de cout.**
- **Se este somador operar sobre inteiros sem sinal, então cout também servirá para indicar a ocorrência de *overflow***

1. Projeto de Unidade Lógico-Aritmética

Somador Para a Primeira Coluna

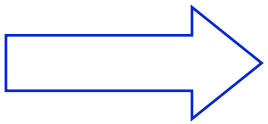
Este circuito é conhecido por "Meio Somador" (*Half-Adder*, em inglês)





Criação da tabela-verdade:

- Listar todas as combinações de entradas (a0, b0)
- Preencher os valores das saídas s0 e c1 baseado no resultado da adição entre a0 e b0



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

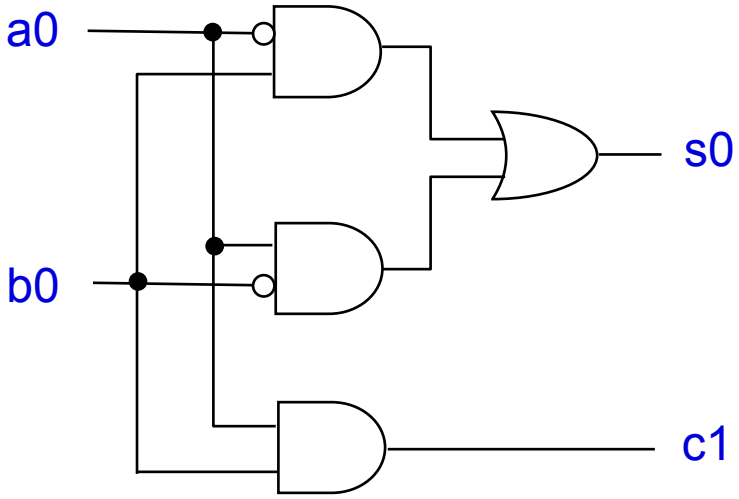
1. Projeto de Unidade Lógico-Aritmética

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

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

$c1 = a0\cdot b0$

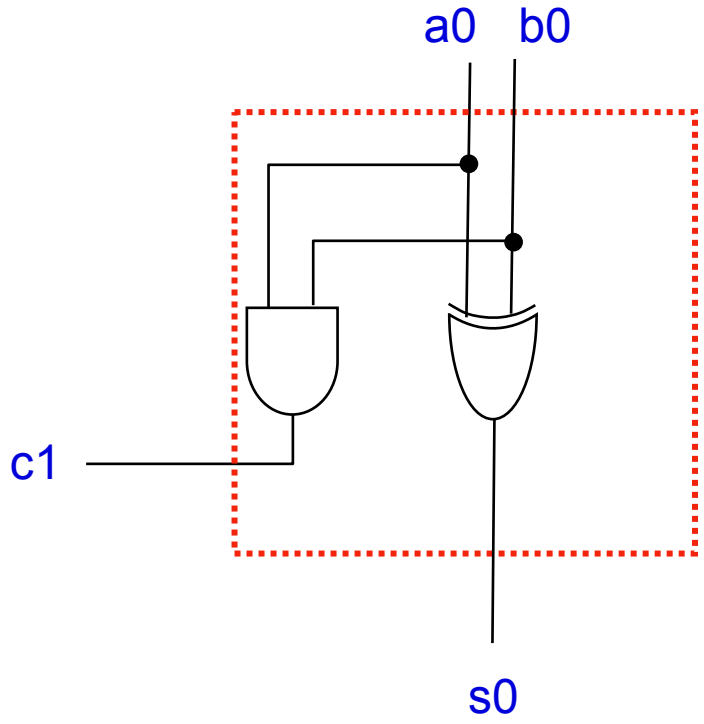


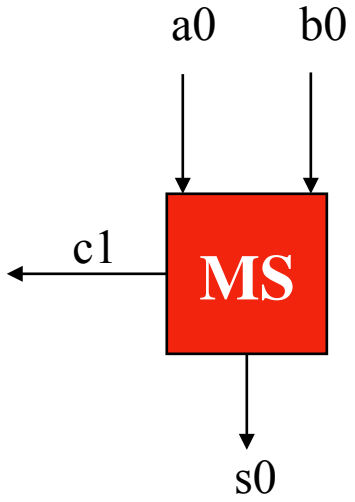
Obs: circuito independente de tecnologia de fabricação

1. Projeto de Unidade Lógico-Aritmética

O Meio Somador

Redesenhando, usando porta XOR





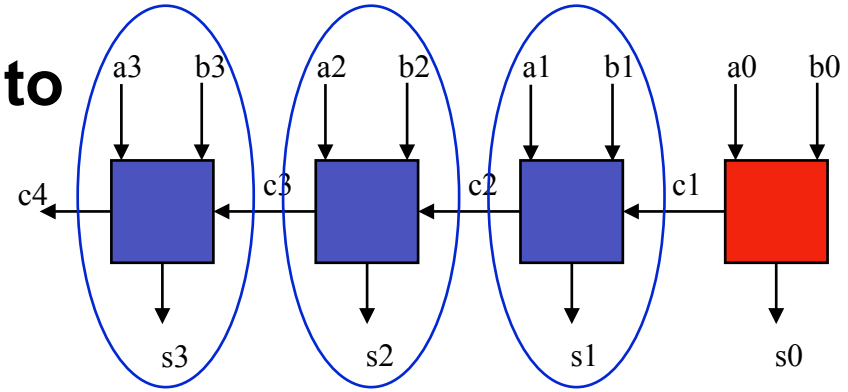
$s0 = \overline{a0}\cdot b0 + a0\cdot\overline{b0} = a0 \oplus b0$

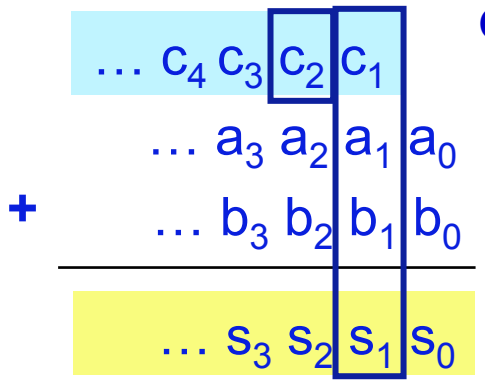
$c1 = a0\cdot b0$

1. Projeto de Unidade Lógico-Aritmética

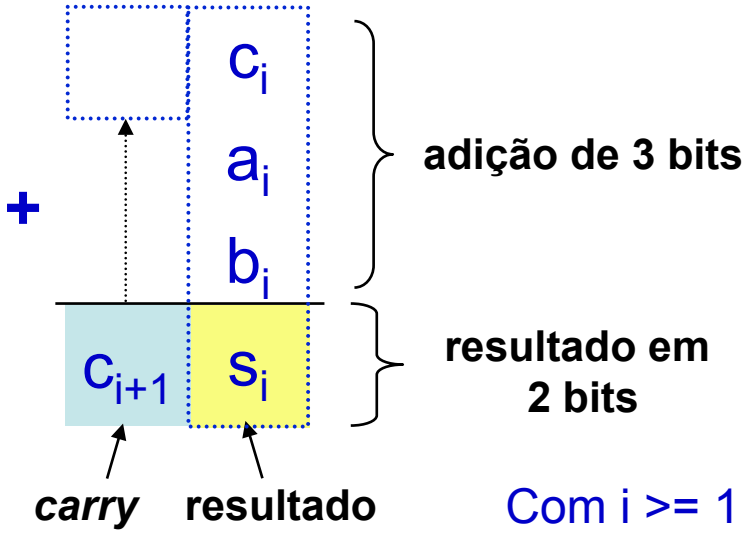
Somador Para as Demais Colunas

Projetando um tipo de circuito para as demais colunas:





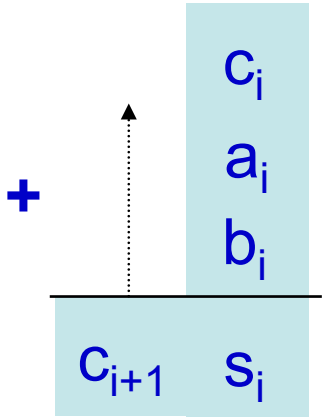
Generalizando…



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 (ci, ai, bi)
- Preencher os valores das saídas si e ci+1 baseado no resultado da adição entre ai, bi e ci

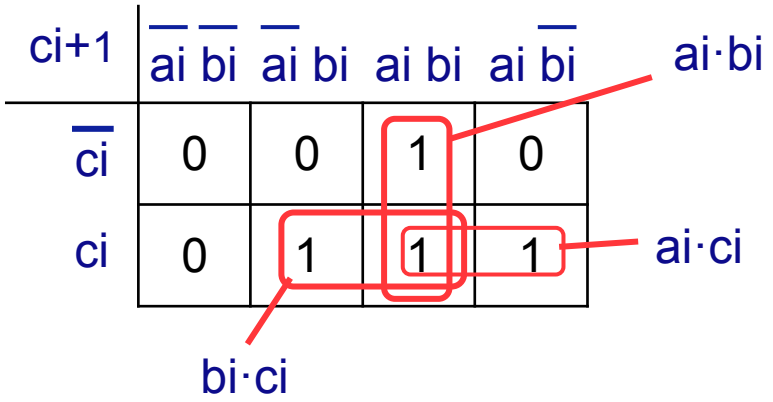
entradas			saídas	
ci	ai	bi	ci+1	si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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 Ci+1



$ci+1 = ai \cdot bi + ai \cdot ci + bi \cdot ci$

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 si

si	$\overline{ai}\,\overline{bi}$	$\overline{ai}\,bi$	$ai\,bi$	$ai\,\overline{bi}$
$\overline{ci}$	0	1	0	1
ci	1	0	1	0

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

$$si = \overline{ai}\cdot bi\cdot\overline{ci} + ai\cdot\overline{bi}\cdot\overline{ci} + \overline{ai}\cdot\overline{bi}\cdot ci + ai\cdot bi\cdot ci$$

1. Projeto de Unidade Lógico-Aritmética

O Somador Completo

Manipulando a expressão para si

$$si = \overline{ai}\cdot bi\cdot\overline{ci} + ai\cdot\overline{bi}\cdot\overline{ci} + \overline{ai}\cdot\overline{bi}\cdot ci + ai\cdot bi\cdot ci$$

$$= \overline{ci}\,(\,\overline{ai}\cdot bi + ai\cdot\overline{bi}\,) + ci\,(\,\overline{ai}\cdot\overline{bi} + ai\cdot bi\,)$$

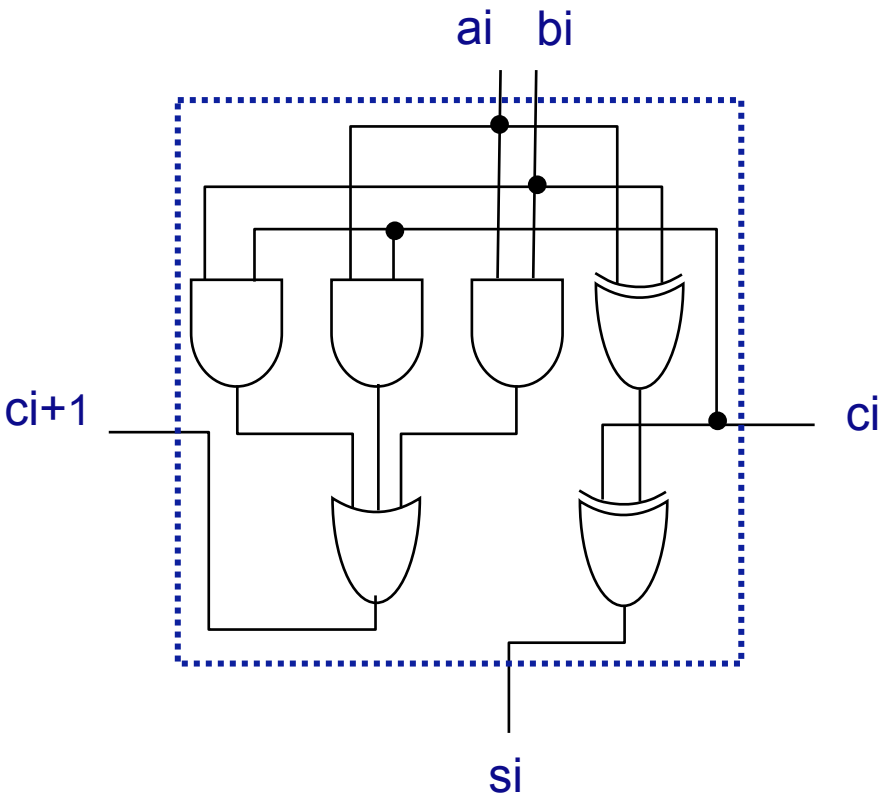
$$= \overline{ci}\,(\,ai \oplus bi\,) \quad + \quad ci\,(\,\overline{ai \oplus bi}\,)$$

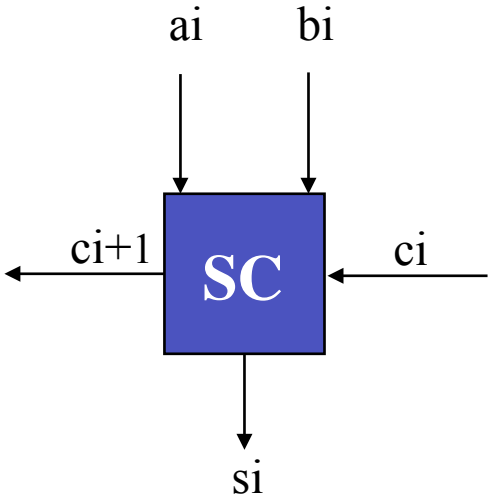
$$= ci \oplus ai \oplus bi$$

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





$si = ci \oplus ai \oplus bi$

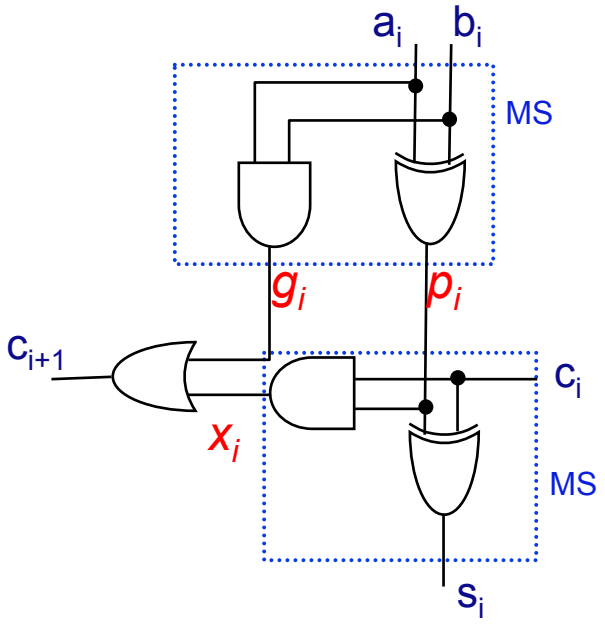
$ci+1 = ai \cdot bi + ai \cdot ci + bi \cdot ci$

1. Projeto de Unidade Lógico-Aritmética

O Somador Completo

Uma Outra Versão, usando dois MS…

ai	bi	ci	*pi*	*xi*	*gi*	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

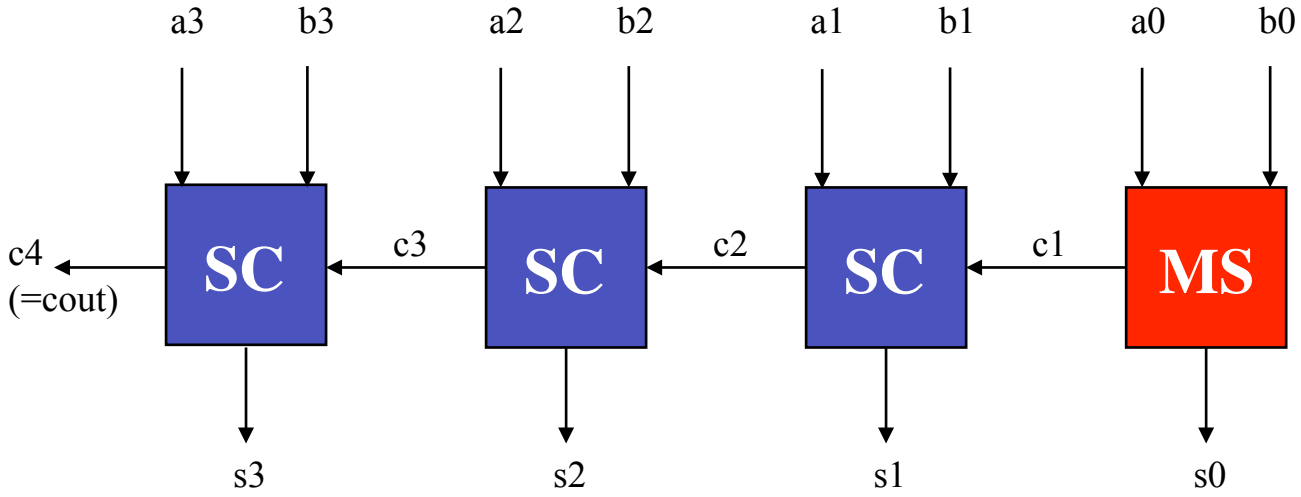


Vantagem: necessita menos portas lógicas

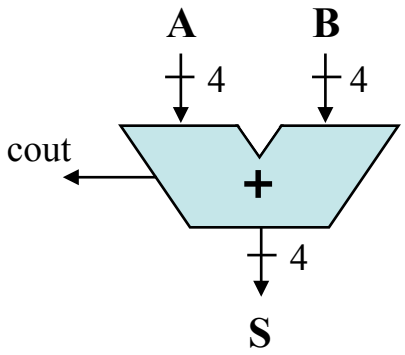
1. Projeto de Unidade Lógico-Aritmética

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

Diagrama de Blocos (Nível Lógico)



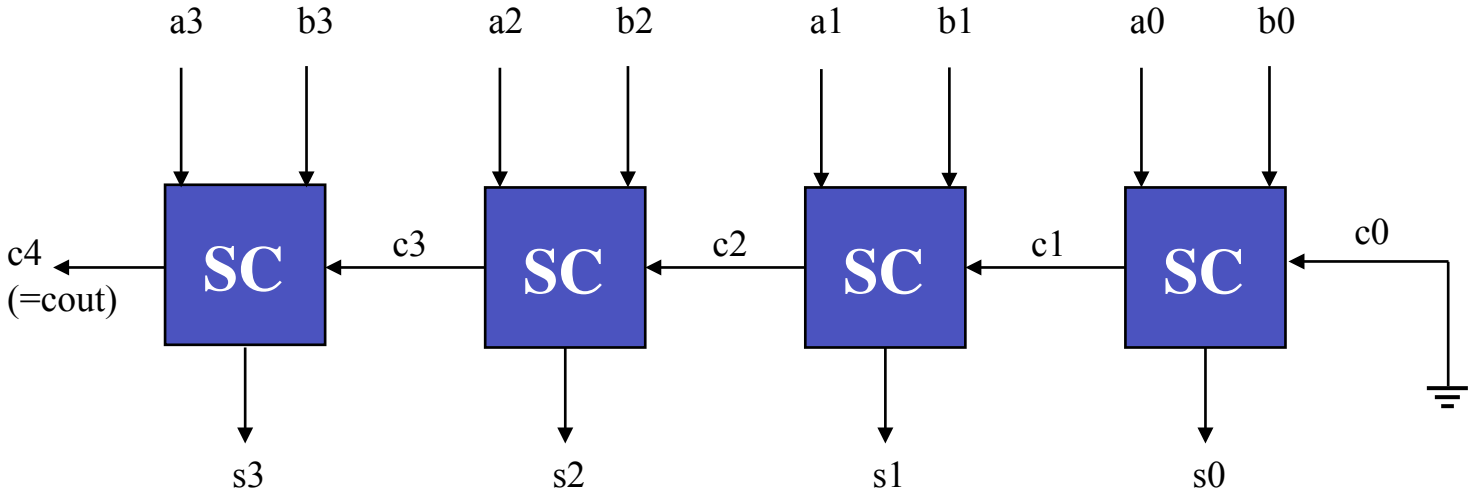
Símbolo no Nível RT



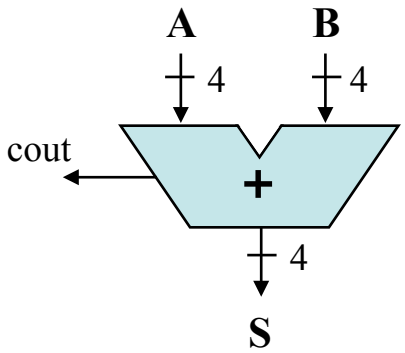
1. Projeto de Unidade Lógico-Aritmética

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

Diagrama de Blocos (Nível Lógico): versão 2



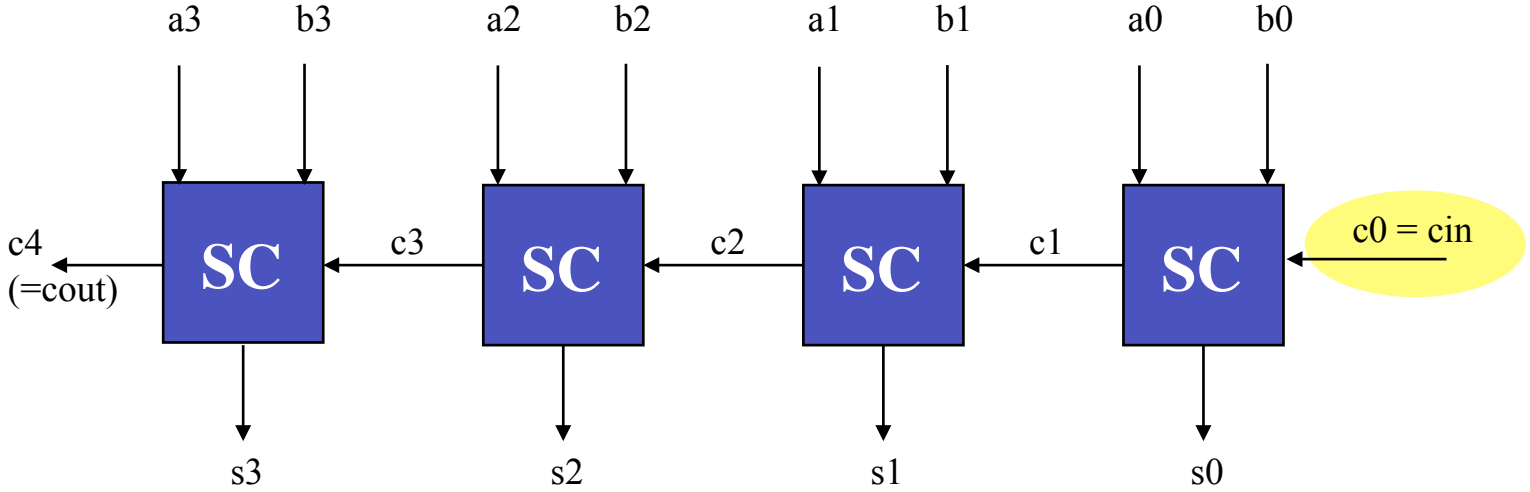
Símbolo no Nível RT



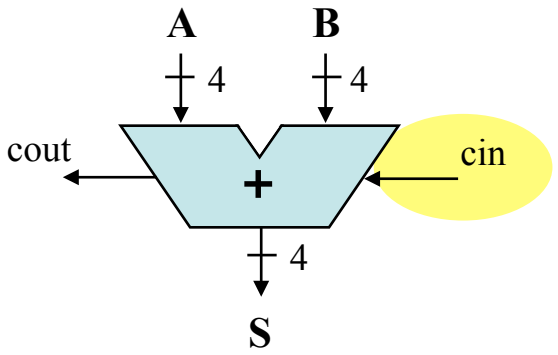
1. Projeto de Unidade Lógico-Aritmética

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

Diagrama de Blocos (Nível Lógico): versão 3



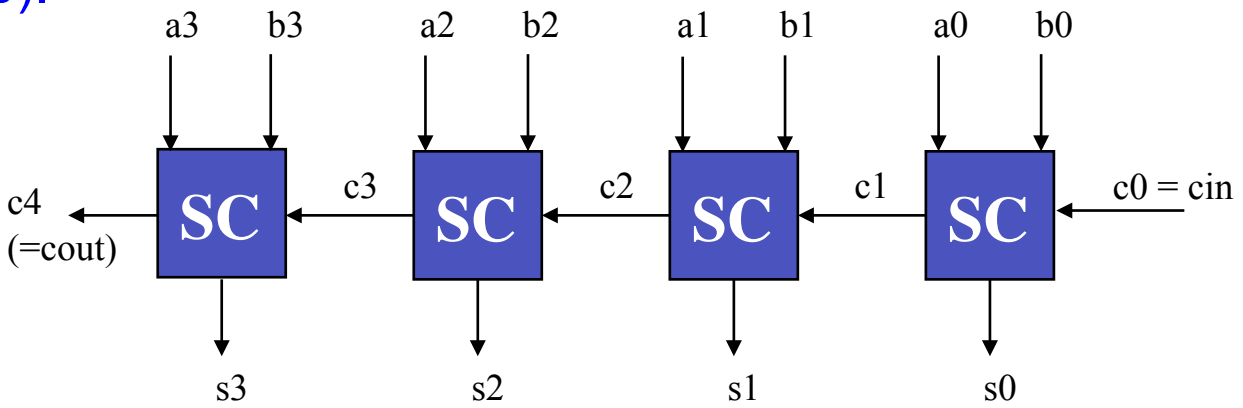
Símbolo no Nível RT



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 c4 depende de c3, que depende de c2, que depende de c1, que depende apenas das entradas c0, a0 e b0).



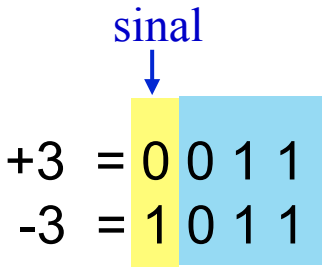
Os slides complementares “somadores rápidos” mostra outros tipos de somadores.

1. Projeto de Unidade Lógico-Aritmética

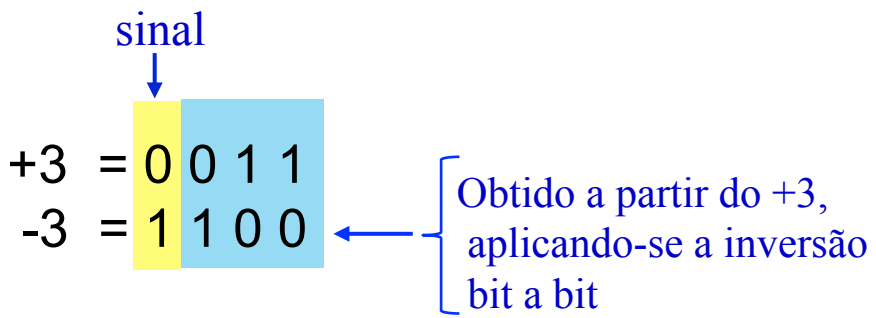
Representação de Inteiros em Binário

Como Representar Inteiros Negativos?

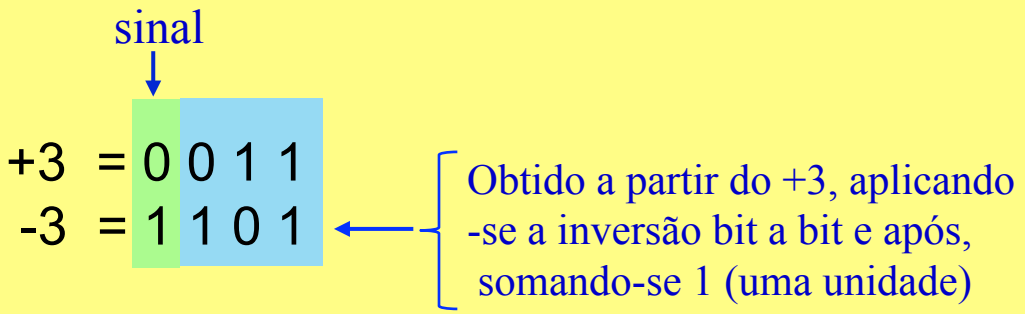
Sinal-magnitude



Complemento de 1



Complemento de 2



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:

$0111_2 = 1x2^0 + 1x2^1 + 1x2^2 + 0x2^3 =$

$= 1 + 2 + 4 + 0 =$

$= \mathbf{+7}$

Observe também que:

$1111_2 = 1x2^0 + 1x2^1 + 1x2^2 - 1x2^3 =$

$= 1 + 2 + 4 - 8 =$

$= \mathbf{-1}$

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 ∈ [-8,+7]

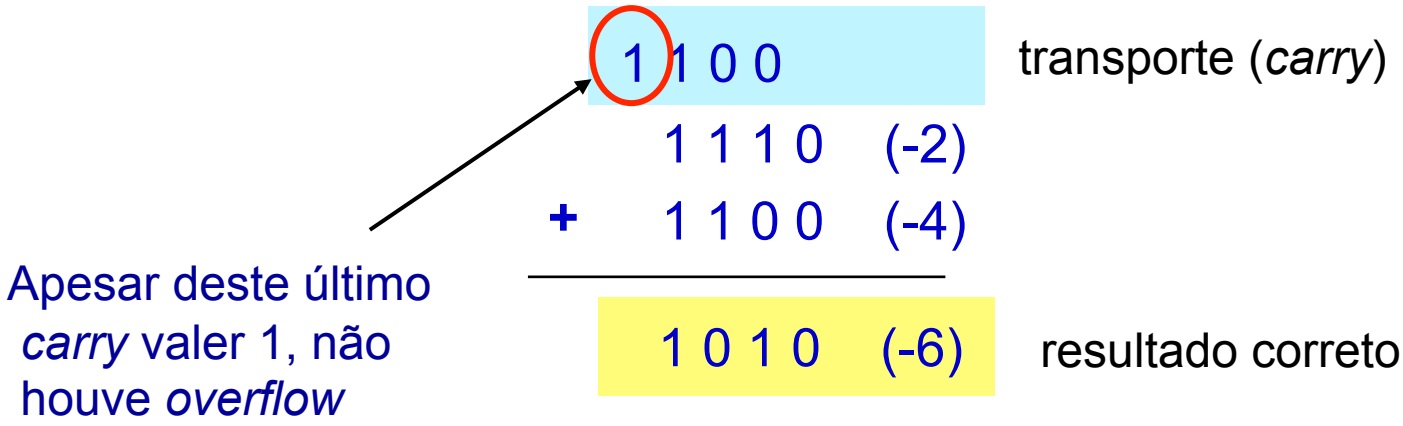
	0 0 0 0	transporte (*carry*)
	0 0 1 0 (+2)	
+	0 1 0 0 (+4)	
	0 1 1 0 (+6)	resultado correto

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

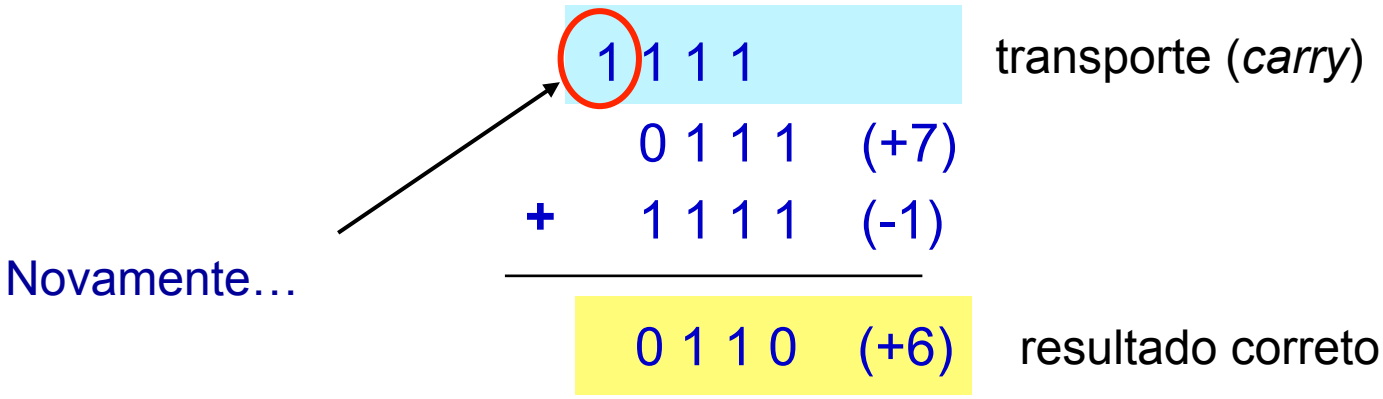


1. Projeto de Unidade Lógico-Aritmética

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



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

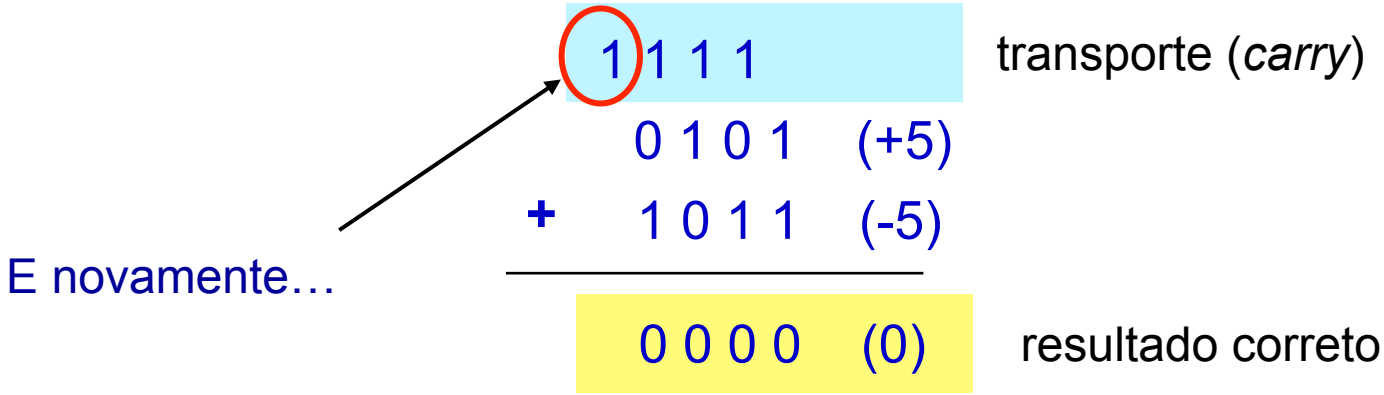
	0 0 0 1		transporte (*carry*)
	1 0 0 1	(-7)	
+	0 0 0 1	(+1)	
	1 0 1 0	(-6)	resultado correto

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

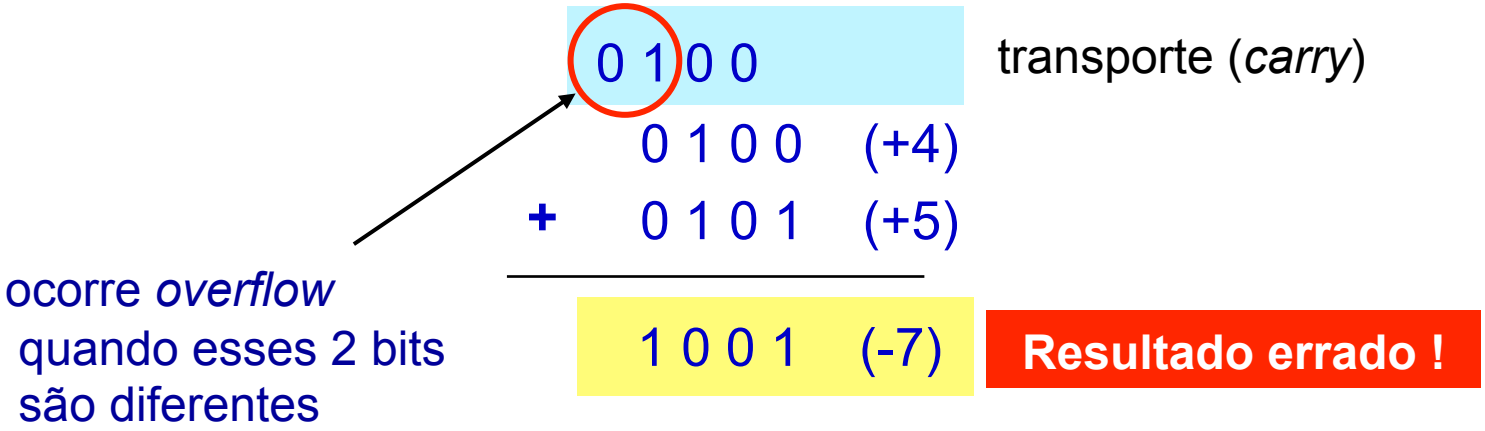


1. Projeto de Unidade Lógico-Aritmética

Adição de Inteiros com Sinal

(Assumindo Negativos em Complemento de 2)

Exemplo 8: 2 números positivos



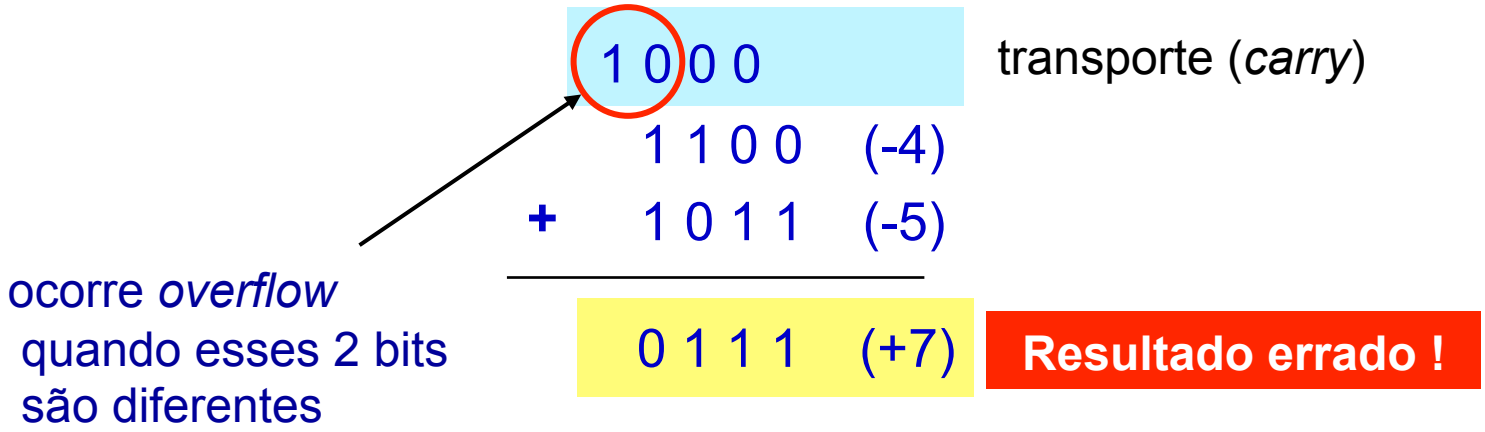
o resultado excede o intervalo de representação = *overflow*

1. Projeto de Unidade Lógico-Aritmética

Adição de Inteiros com Sinal

(Assumindo Negativos em Complemento de 2)

Exemplo 9: 2 números negativos



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***

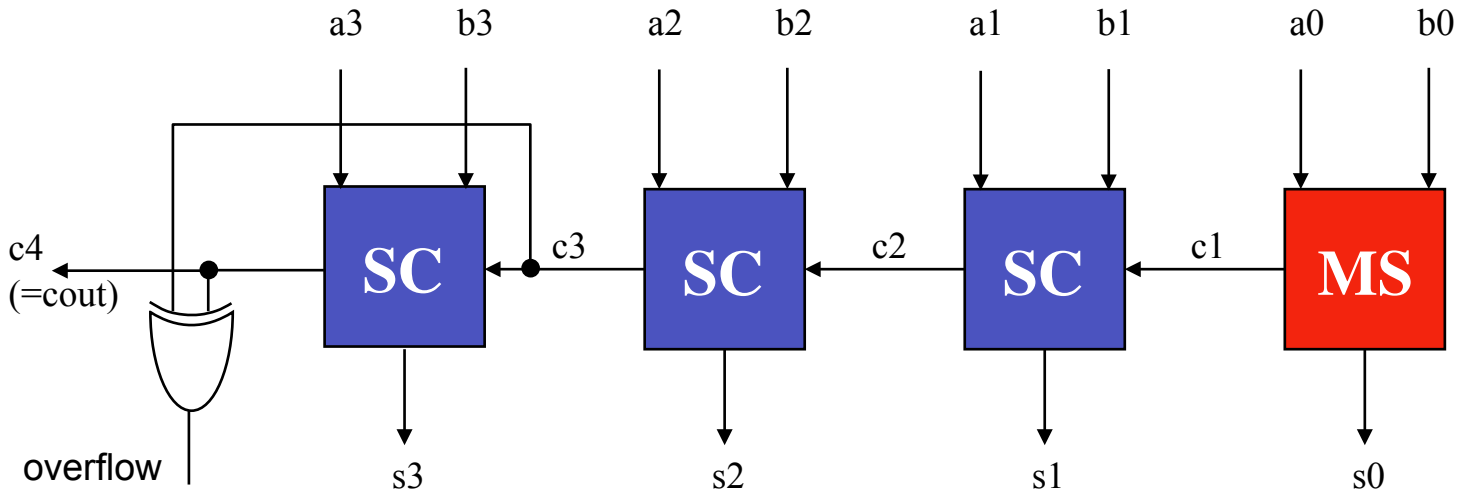
1. Projeto de Unidade Lógico-Aritmética

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)



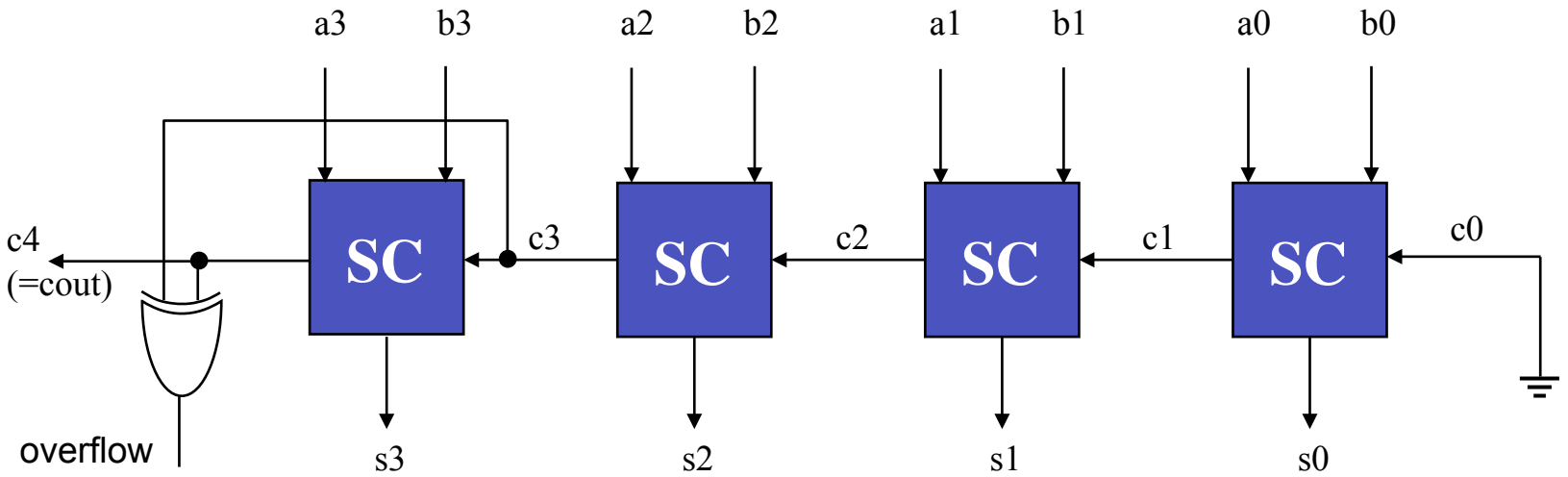
1. Projeto de Unidade Lógico-Aritmética

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



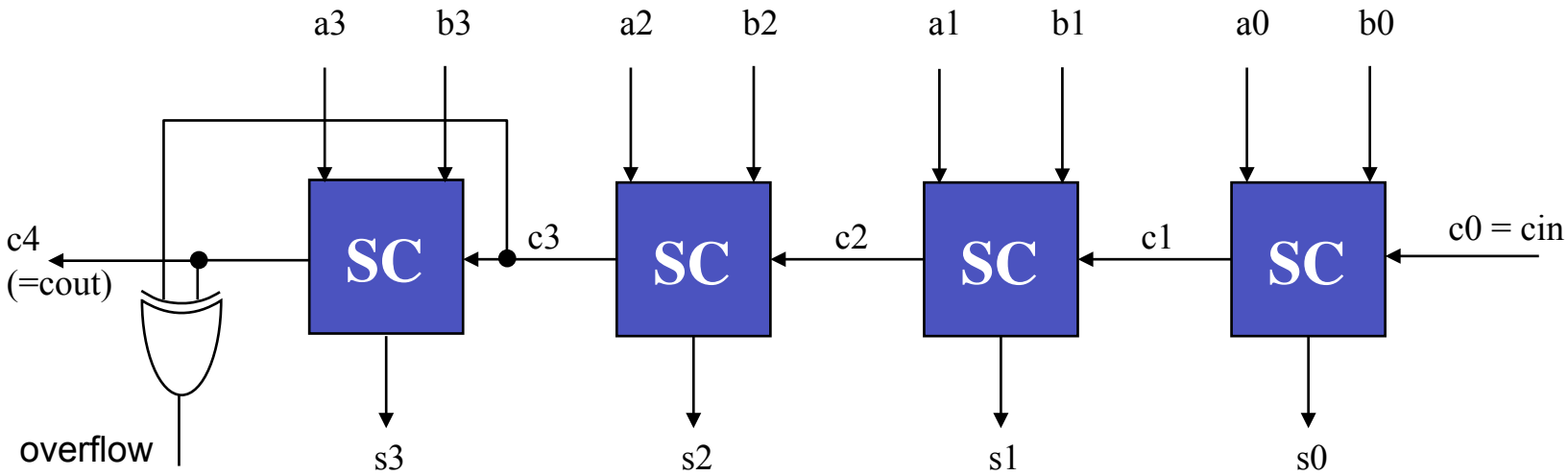
1. Projeto de Unidade Lógico-Aritmética

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