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

Sistemas Digitais

INE 5406

Aula 2-T

2. circuitos aritméticos: adição de números sem sinal, adição de números com sinal, circuitos somador (*ripple-carry*), subtrator e somador-subtrator, *overflow*.

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

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

2. Circuitos Aritméticos

Revisão da Adição Binária

Adição de Números Sem Sinal

Exemplo

overflow	1 1 0 0		transportes (“carry”)
	1 1 0 0	(12)	
+	0 1 1 0	(6)	
	0 0 1 0	(18)	resultado

- **Note que o maior binário que se pode representar com 4 bits é 15**

Revisão da Adição Binária

Adição de Números Sem Sinal

Outro Exemplo

representação simplificada

	0 1 1 0 1 1 0 0		transportes	
	0 1 1 0 1 1 0 1	(109)	$\rightarrow$	A
+	0 1 1 0 0 1 1 0	(102)	$\rightarrow$	+ B
	1 1 0 1 0 0 1 1	(211)	resultado $\rightarrow$	S

2. Circuitos Aritméticos

Revisão da Adição Binária

```
     0 1 1 0 1 1 0 0
       0 1 1 0 1 1 0 1
  +    0 1 1 0 0 1 1 0
     ------------------
       1 1 0 1 0 0 1 1    (211)
```

$$\begin{array}{rcc} & & a_0 \\ + & & b_0 \\ \hline & c_1 & s_0 \end{array}$$

adição de 2 bits

resultado em 2 bits

2. Circuitos Aritméticos

Revisão da Adição Binária

Porém, a partir do 2º bit …

```
    0 1 1 0 1 1 0 0
      0 1 1 0 1 1 0 1
+     0 1 1 0 0 1 1 0
  ___________________
      1 1 0 1 0 0 1 1   (211)
```

$$\begin{array}{rcc} & & c_1 \\ & & a_1 \\ + & & b_1 \\ \hline & c_2 & s_1 \end{array}$$

adição de 3 bits

resultado em 2 bits

2. Circuitos Aritméticos

Revisão da Adição Binária

Generalizando, para bits a partir do 2º bit

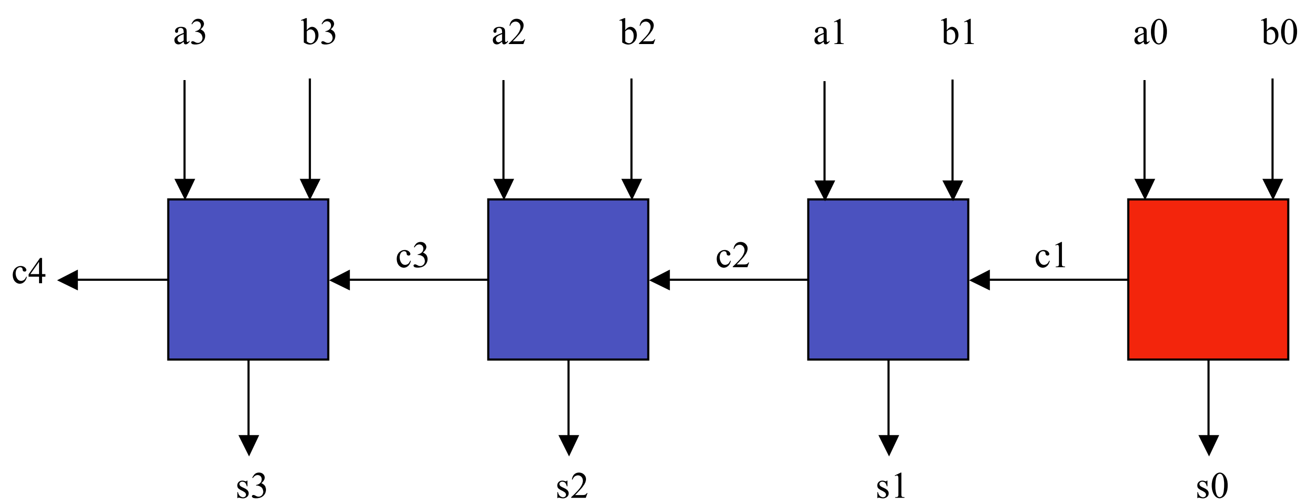
```
   0 1 1 0 1 1 0 0
     0 1 1 0 1 1 0 1
+    0 1 1 0 0 1 1 0
   ------------------
     1 1 0 1 0 0 1 1    (211)
```

		c_i	
		a_i	adição de 3 bits
+		b_i	
	C_{i+1}	s_i	resultado em 2 bits

2. Circuitos Aritméticos

Esquema da Soma Paralela

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

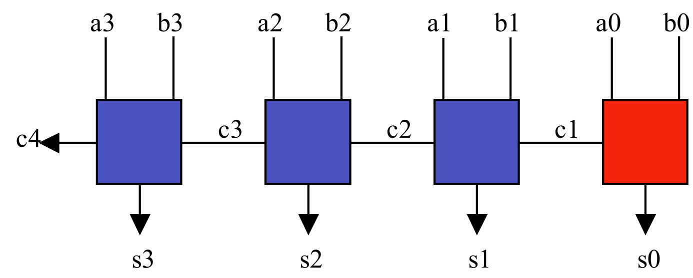


Note que:

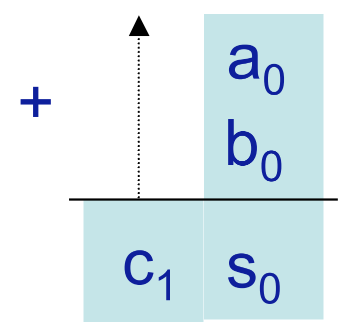
- **há um elemento para cada coluna da soma**
- **o sinal de *overflow* será o *carry* mais significativo**

Projetando um Somador Paralelo

Projetando um circuito para a primeira coluna




```
    0 1 1 0 1 1 0 0
      0 1 1 0 1 1 0 1
+     0 1 1 0 0 1 1 0
  ---------------------
      1 1 0 1 0 0 1 1    (211)
```



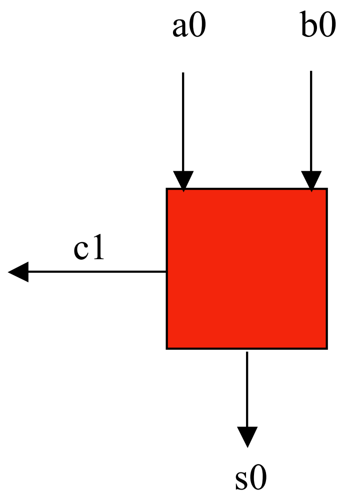
Quantas combinações de 2 bits existem?

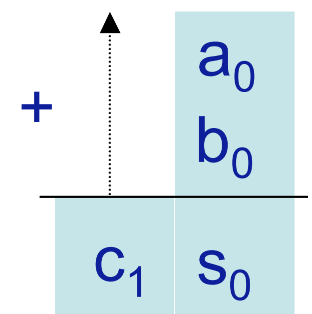
2. Circuitos Aritméticos

Projetando um Somador Paralelo

Projetando um circuito para a primeira coluna

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





Projetando um Somador Paralelo

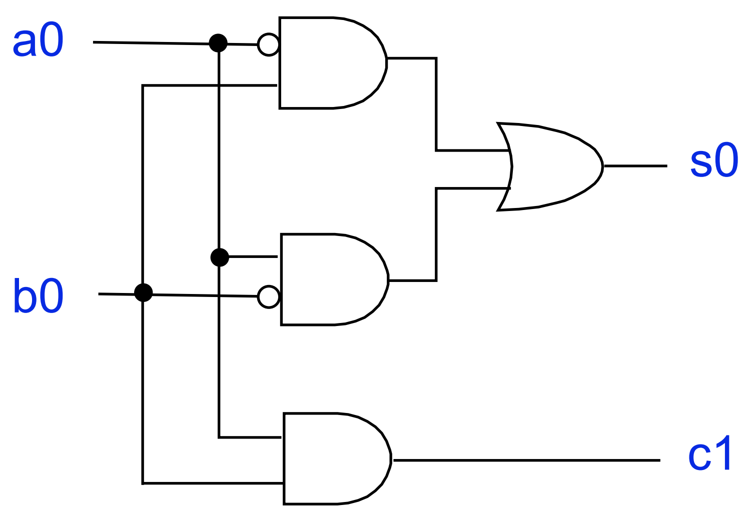
Projetando um circuito para a primeira coluna:

O Meio Somador (MS)

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

$c1 = a0 \cdot b0$

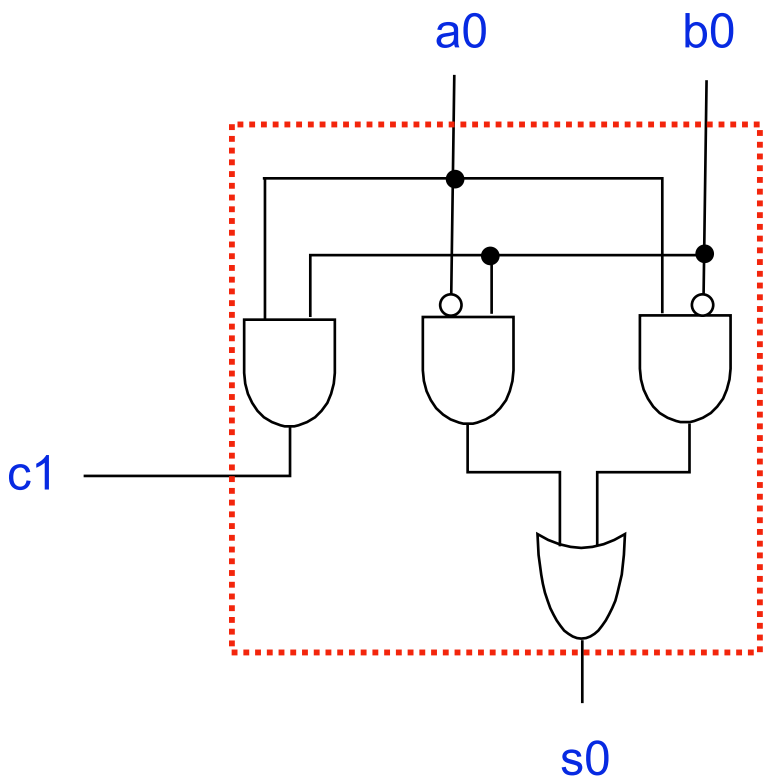
entradas		saídas	
a0	b0	c1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

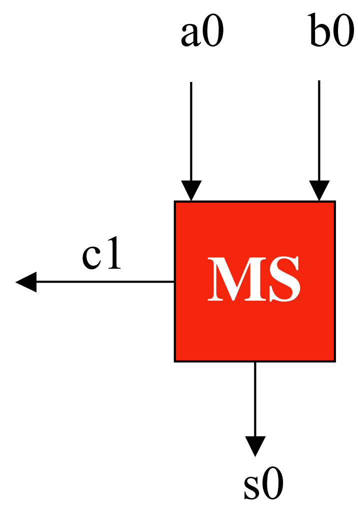


Obs: circuito independente de tecnologia

O Meio Somador (*Half Adder*)

Ou, se preferir





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

$$c1 = a0\cdot b0$$

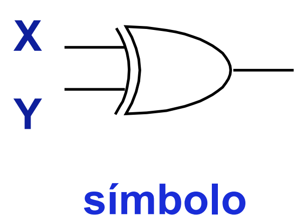
A Função OU Exclusivo

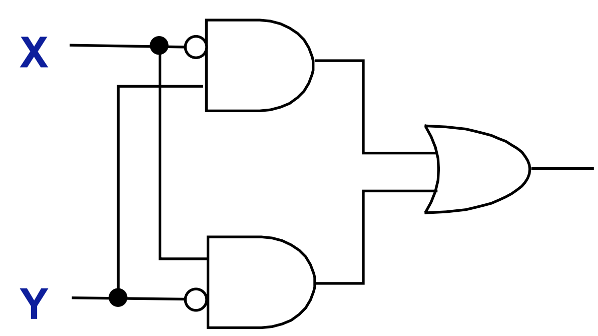
***EXclusive OR* - XOR**

- **A função XOR resulta 1 se um número ímpar de entradas valer 1**
- **Para duas entradas, temos a seguinte tabela-verdade**

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

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





A Função OU Exclusivo (XOR)

Aplicações 1

A função XOR pode ser usada para comparar dois sinais, da seguinte forma:

- se $X \oplus Y = 0 \Rightarrow X = Y$
- se $X \oplus Y = 1 \Rightarrow X \neq Y$

X	Y	$X \oplus Y$	
0	0	0	$\Rightarrow$ X=Y
0	1	1	
1	0	1	
1	1	0	$\Rightarrow$ X=Y

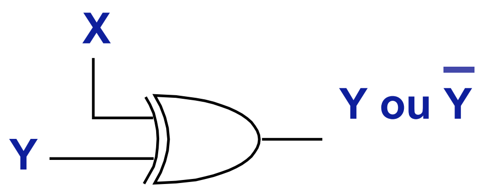
A Função OU Exclusivo (XOR)

Aplicações 2

A função XOR pode ser usada como um **negador controlado**

Assumindo (p.ex.) que X é o controle

- se $X=0 \Rightarrow X \oplus Y = Y$
- se $X=1 \Rightarrow X \oplus Y = \overline{Y}$



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

A Função OU Exclusivo (XOR)

Aplicações 3

A função XOR realiza uma adição sem o *carry*

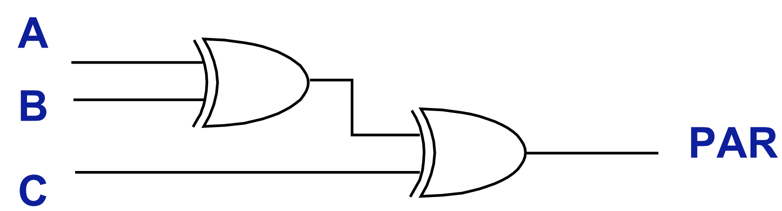
X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

A Função OU Exclusivo (XOR)

Aplicações 4

A função XOR pode ser usada para calcular a paridade (seguindo sua própria definição...)

A	B	C	PAR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



2. Circuitos Aritméticos

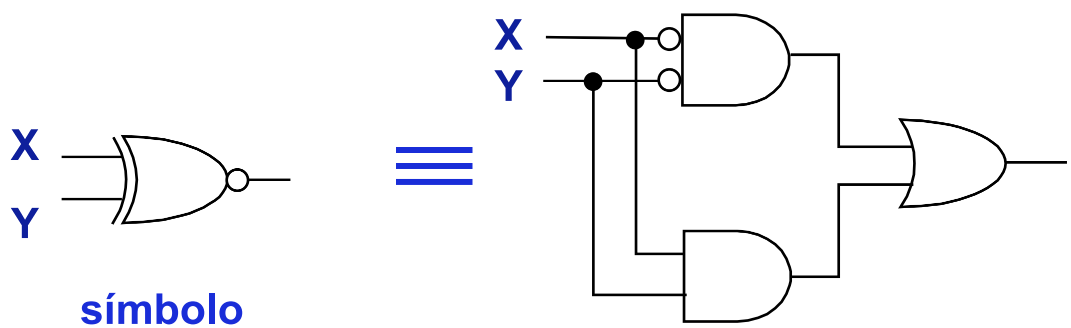
A Função Equivalência (ou Concidência)

***EXclusive NOR* - XNOR**

- **A função XNOR corresponde à função XOR negada**

X	Y	$X \odot Y$
0	0	1
0	1	0
1	0	0
1	1	1

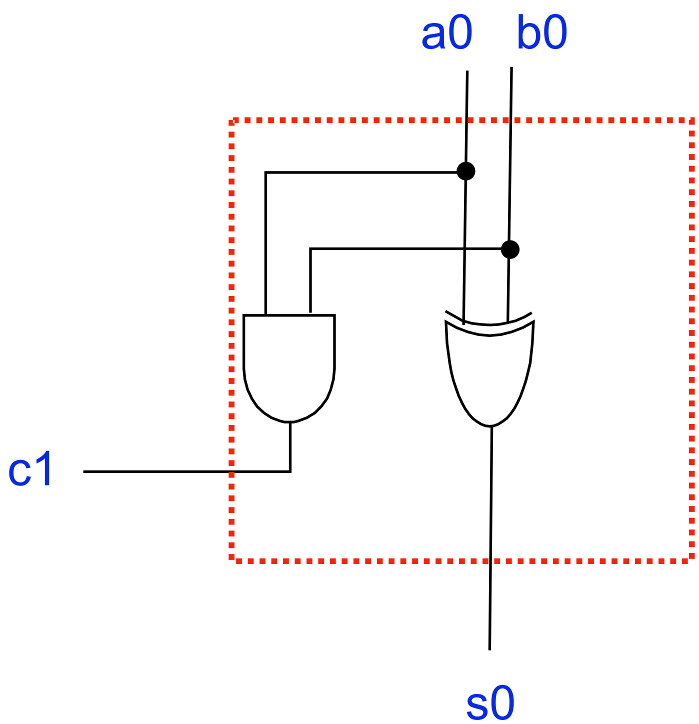
$$X \odot Y = \overline{X} \cdot \overline{Y} + X \cdot Y = \overline{X \oplus Y}$$

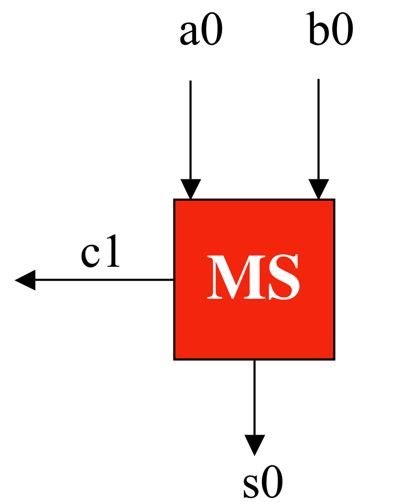


2. Circuitos Aritméticos

Voltando ao Meio Somador (*Half Adder*)

Redesenhando o meio somador, usando porta XOR





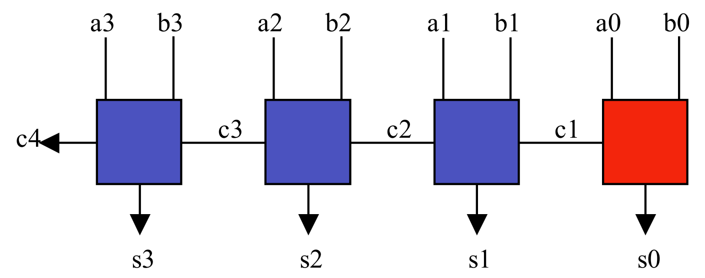
$s0 = \overline{a0} \cdot b0 + a0 \cdot \overline{b0}$

$c1 = a0 \cdot b0$

2. Circuitos Aritméticos

Projetando um Somador Paralelo

Projetando o circuito para as demais colunas




```
  0 1 1 0 1 1 0 0
    0 1 1 0 1 1 0 1
+   0 1 1 0 0 1 1 0
  -----------------
    1 1 0 1 0 0 1 1   (211)
```

$$\begin{array}{rcc} & & c_i \\ & & a_i \\ + & & b_i \\ \hline & c_{i+1} & s_i \end{array}$$

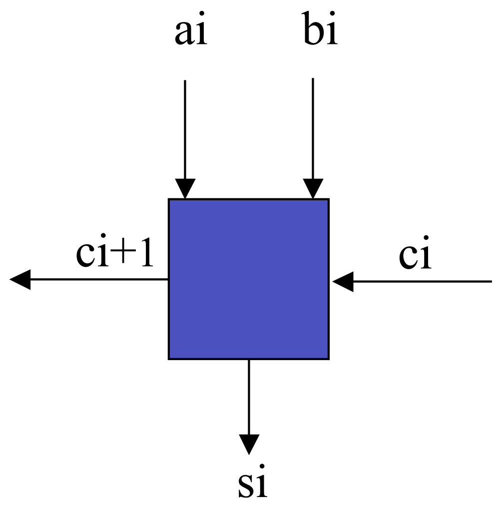
Quantas combinações de 3 bits existem?

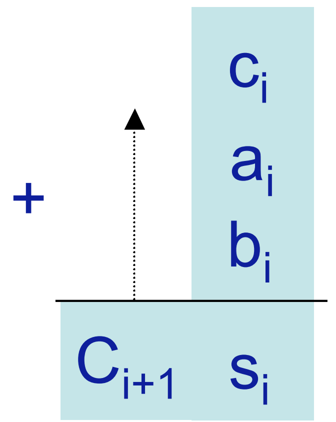
2. Circuitos Aritméticos

Projetando um Somador Paralelo

Projetando um circuito para as demais colunas

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





2. Circuitos Aritméticos

Projetando um Somador Paralelo

Projetando um circuito para as demais colunas

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

ai·bi

ai·ci

bi·ci

ci+1 = ai·bi+ ai·ci+ bi·ci

Projetando um Somador Paralelo

Projetando um circuito para as demais colunas

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!

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

Projetando um Somador Paralelo

Manipulando a expressão para si

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

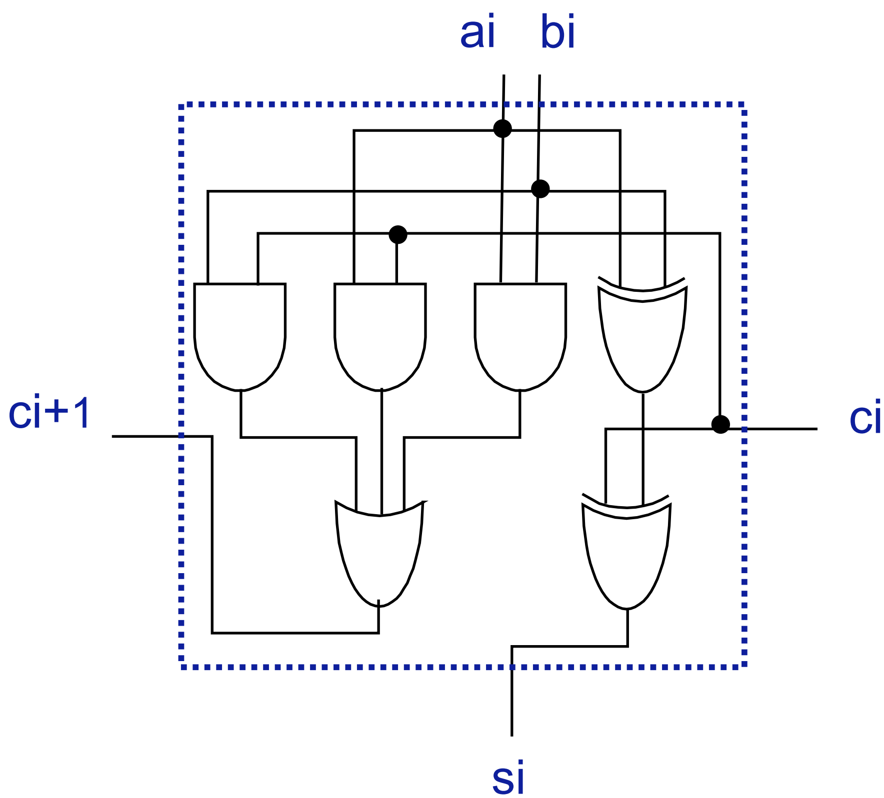
$$= \overline{ci}\ \underbrace{(\ \overline{ai}\cdot bi + ai\cdot \overline{bi}\ )} + ci\ \underbrace{(\ \overline{ai}\cdot \overline{bi} + ai\cdot bi\ )}$$

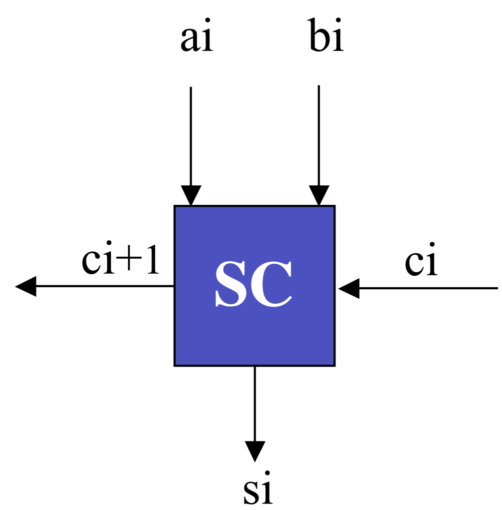
$$= \overline{ci}\ (\ ai \oplus bi\ ) \quad + \quad ci\ (\ \overline{ai \oplus bi}\ )$$

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

Normalmente, se assumem portas xor com duas entradas, o que aliás, corresponde à realidade da implementação em tecnologia CMOS.

O Somador Completo (*Full Adder*)





$si = ci \oplus ai \oplus bi$

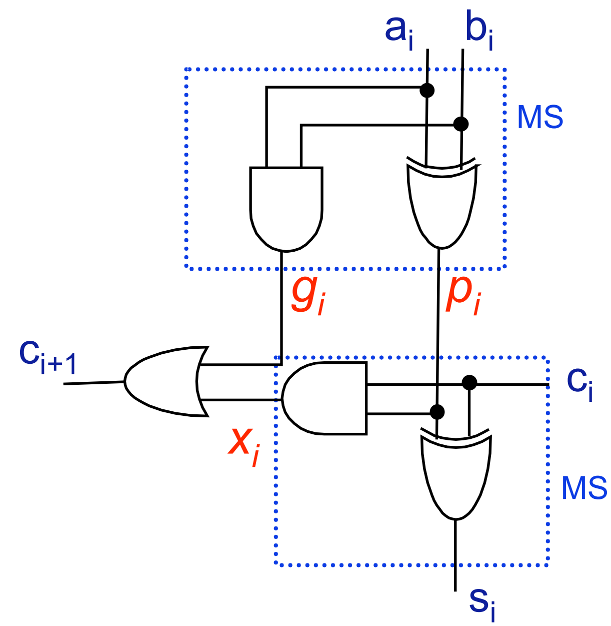
$ci+1 = ai \cdot bi + ai \cdot ci + bi \cdot ci$

2. Circuitos Aritméticos

O Somador Completo (*Full Adder*)

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

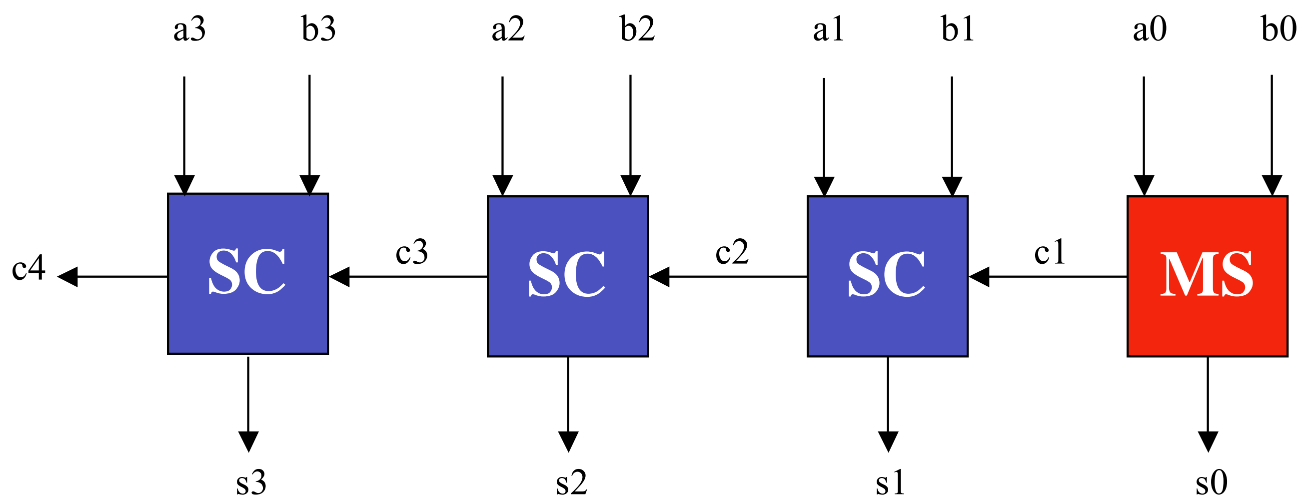


Vantagem: menos portas lógicas

2. Circuitos Aritméticos

Somador Binário Paralelo (de 4 bits)

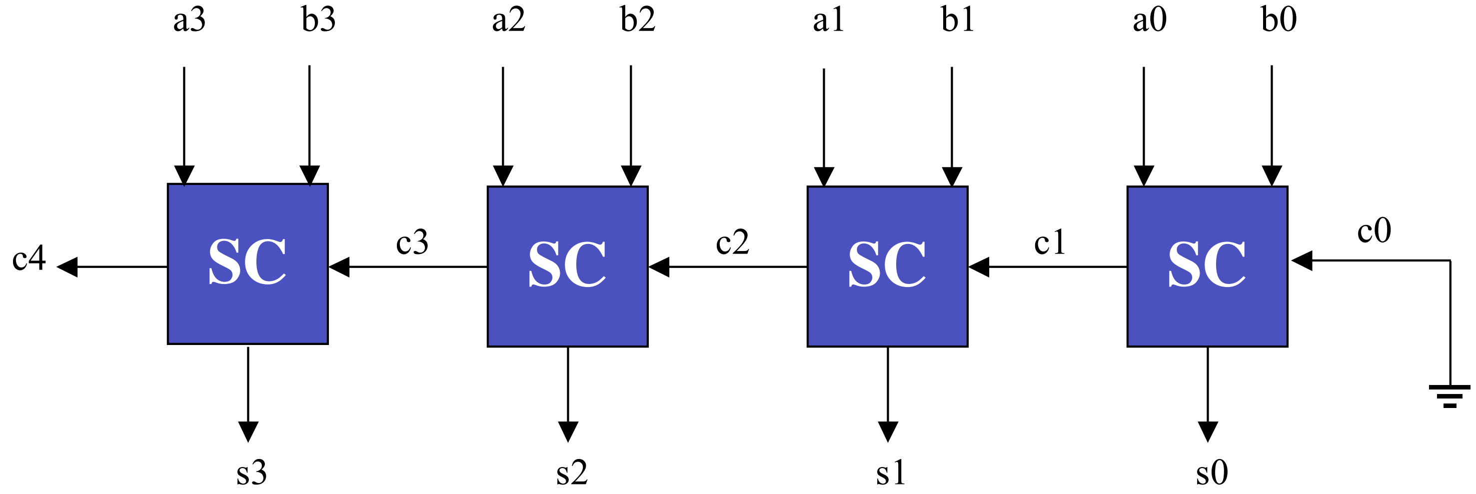
Considerando dois números (A e B) com 4 bits cada



Note que o sinal c4 somente estabiliza depois que c1, c2 e c3 estabilizarem

Somador Binário Paralelo (de 4 bits)

Versão 2: somente com somadores completos



O Custo é ligeiramente maior, porém funciona!

2. Circuitos Aritméticos

E se os números que quisermos operar tiverem sinal?

- **Precisaremos considerar uma representação que sirva tanto para binários positivos quanto binários negativos**
- **A representação mais usada, neste caso, é complemento de 2**
- **Lembremos do porquê disto…**

Revisão da Adição Binária

Representação de Números Positivos e Negativos em Binário

Representação em sinal-magnitude

Exemplos: +9 e -9 representados com 8 bits

		sinal	magnitude
+9	=	0	0 0 0 1 0 0 1
-9	=	1	0 0 0 1 0 0 1

Revisão da Adição Binária

Representação de Números Positivos e Negativos em Binário

Representação em complemento de 2

Exemplos: +9 e -9 representados com 8 bits

sinal

+9 =	0	0	0	0	1	0	0	1
-9 =	1	1	1	1	0	1	1	1

Revisão da Adição Binária

Representação em complemento de 2

Sinal	Regra de formação	Exemplo
positivos	= sinal-magnitude	+9= 0 0 0 0 1 0 0 1
negativos	1. Toma-se a representação em sinal-magnitude	+9= 0 0 0 0 1 0 0 1
	2. Inverte-se o número, bit a bit	1 1 1 1 0 1 1 0
	3. Soma-se 1	1 1 1 1 0 1 1 1 = -9

Revisão da Adição Binária

Adição de números binários em complemento de 2

Para os próximos exemplos, considere números com 4 bits (ou seja, o intervalo de representação será **[-8,+7]**)

Exemplo 1: dois números positivos, cuja soma seja ≤ +7

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

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 2: dois números negativos, cuja soma seja ≥ -8

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

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

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 3: um número positivo e um número negativo, tais que o resultado é positivo

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

Novamente…

2. Circuitos Aritméticos

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 4: 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

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 5: um positivo e um negativo, iguais em módulo

1 1 1 1		transporte ("carry")
0 1 0 1	(+5)	
+ 1 0 1 1	(-5)	
0 0 0 0	(0)	resultado correto

E novamente…

2. Circuitos Aritméticos

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 6: 2 números positivos

	0 1 0 0		transporte (“carry”)
	0 1 0 0	(+4)	
+	0 1 0 1	(+5)	
	1 0 0 1	(-7)	**Resultado errado !**

ocorre “overflow” quando esses 2 bits são diferentes

o resultado excede o intervalo de representação = overflow

Revisão da Adição Binária

Adição de números binários em complemento de 2

Exemplo 7: 2 números negativos

	1 0 0 0	transporte ("carry")
	1 1 0 0	(-4)
+	1 0 1 1	(-5)
	0 1 1 1	(+7)

Resultado errado !

ocorre "overflow" quando esses 2 bits são diferentes

o resultado excede o intervalo de representação = overflow

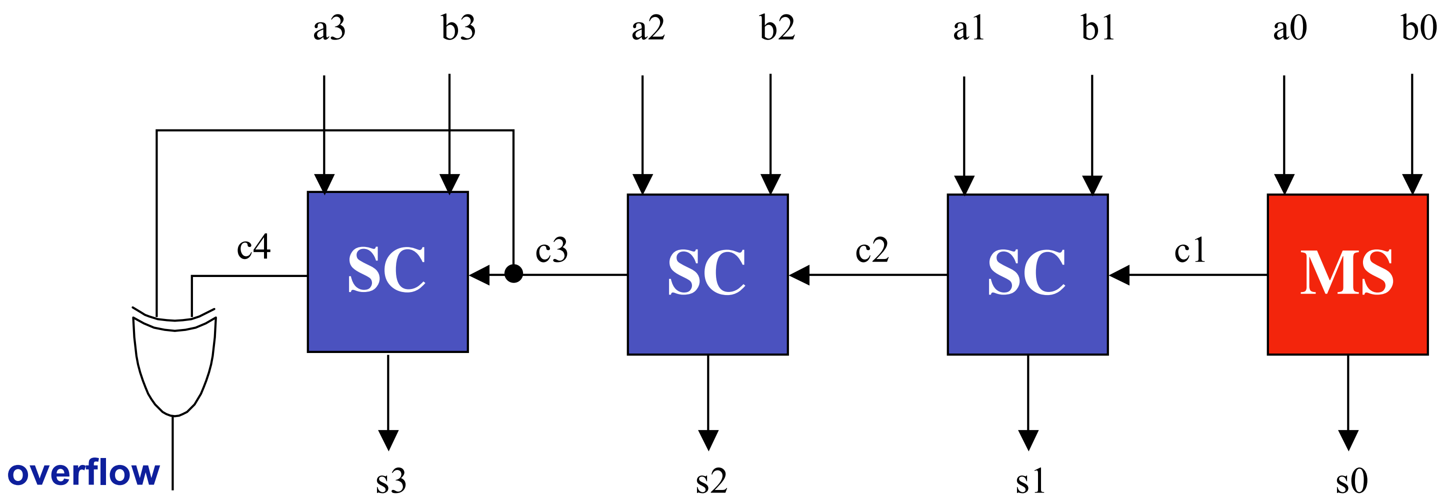
Revisão da Adição Binária

Adição de números binários em complemento de 2

Conclusões:

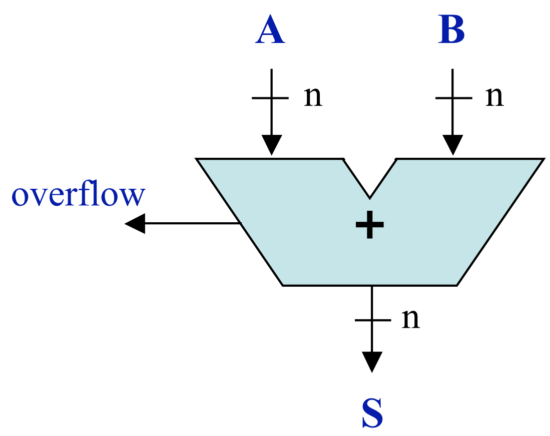
- **Números binários em complemento de 2 podem ser adicionados como se fossem número binários sem sinal**
- **Neste caso, a detecção de *overflow* se dá comparando-se os dois últimos sinais de *carry***

Somador Binário Paralelo para Números em Complemento de 2



esquemático de blocos

Somador Binário Paralelo (para Números em Complemento de 2)



símbolo no nível RT

Subtração Binária

Princípio Básico

$$A - B = A + (-B)$$

onde -B é o número B de sinal trocado

Subtração Binária

Princípio Básico

Trocar o sinal equivale a **Determinar o complemento de 2**

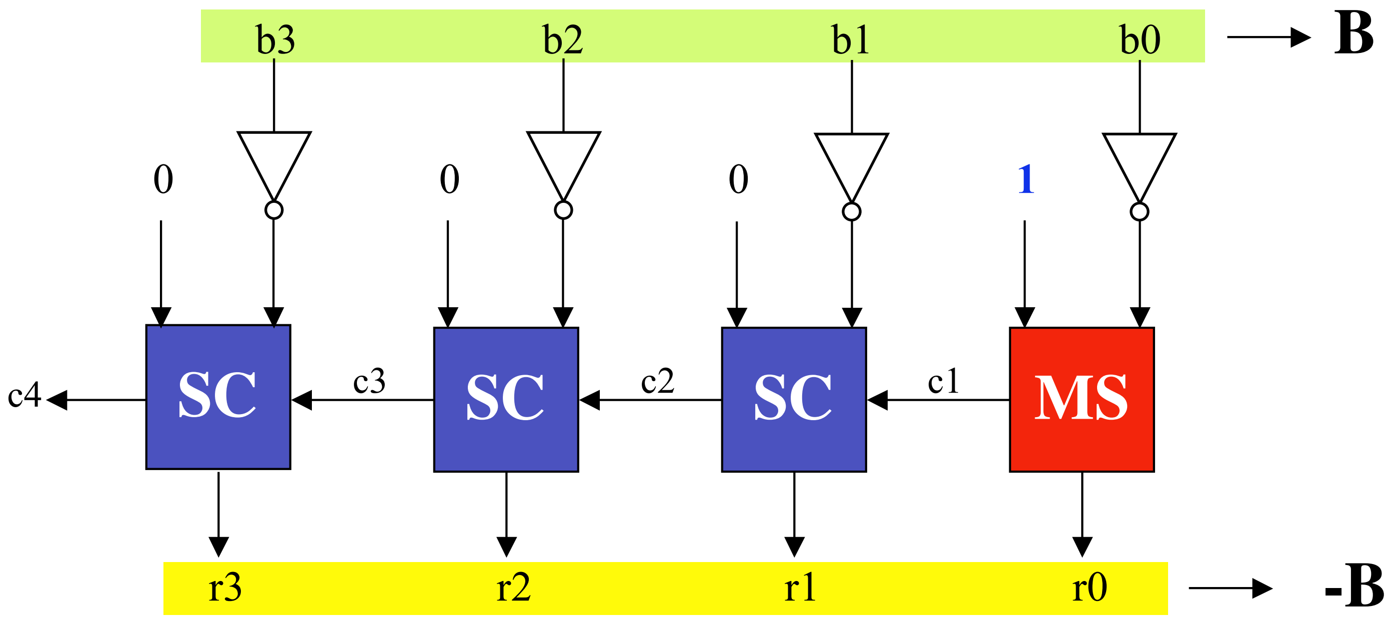
Então

A - B = A + (- B) = A + (B em complemento de 2**)**

Subtração Binária

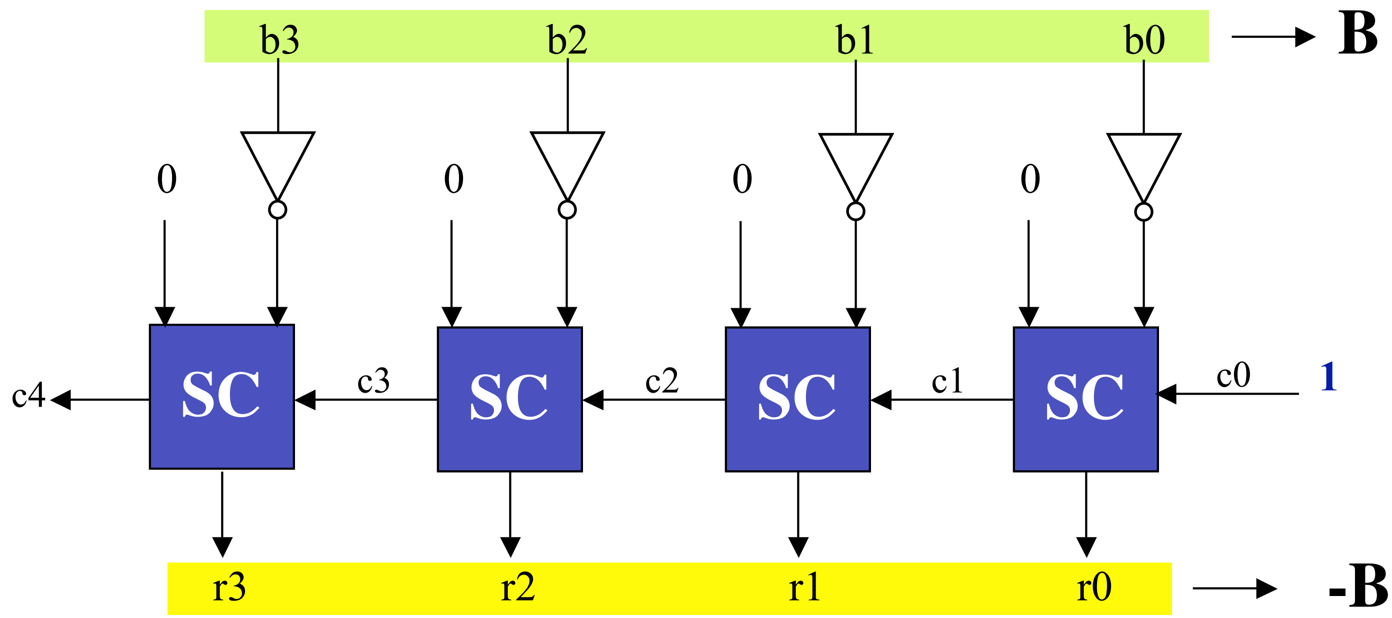
Como determinar o complemento de 2 de um número?

1. Toma-se a representação em sinal-magnitude
2. Inverte-se o número, bit a bit
3. Soma-se 1



Subtração Binária

Outra configuração de circuito…



Subtração Binária

Trocar o sinal → equivale a → **Determinar o complemento de 2**

Mas será que isso funciona se o número é negativo e queremos trocar seu sinal? Vejamos um exemplo…

1 0 0 1 = -7 (com 4 bits, complemento de 2)

Aplicando as regras do complemento de 2…

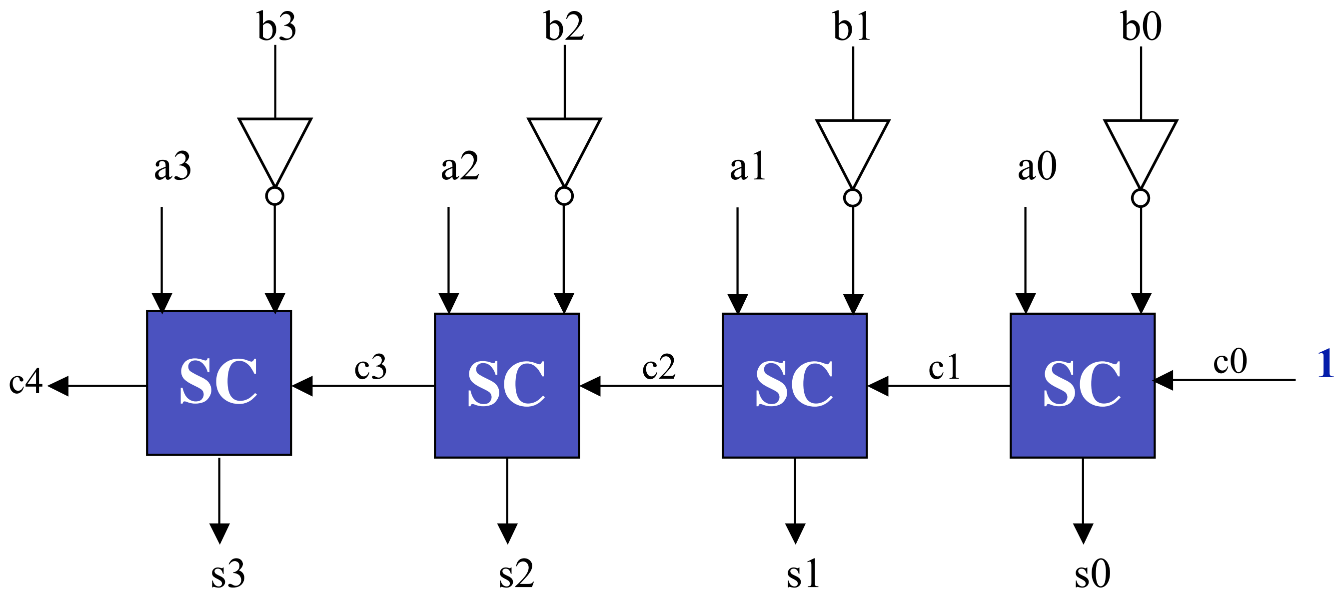
invertendo, bit a bit	0 1 1 0
somando 1	0 1 1 1 = +7 (com 4 bits)

Funciona !!

Subtração Binária

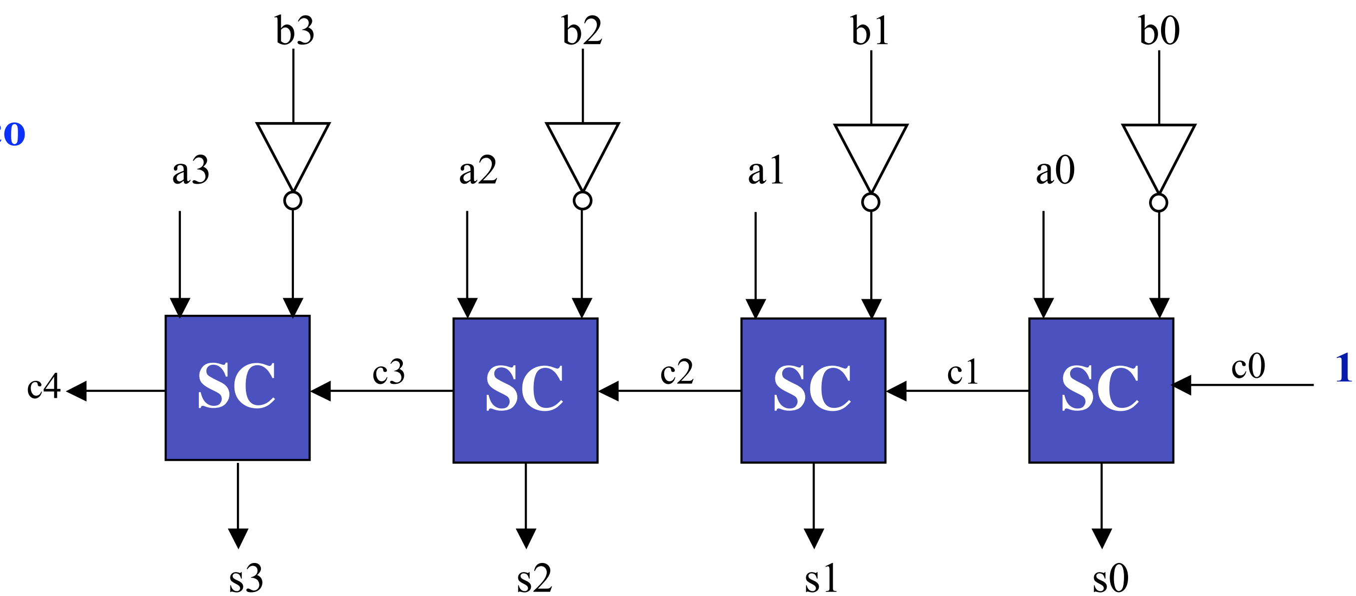
Voltando à subtração:

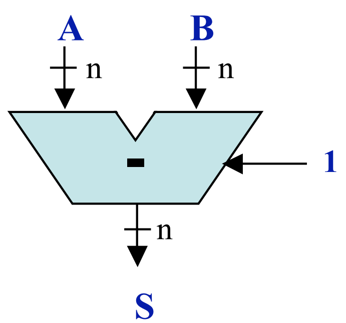
A - B = A + (B em complemento de 2**)**



Subtrator Binário Paralelo (de 4 bits)

esquemático
de blocos

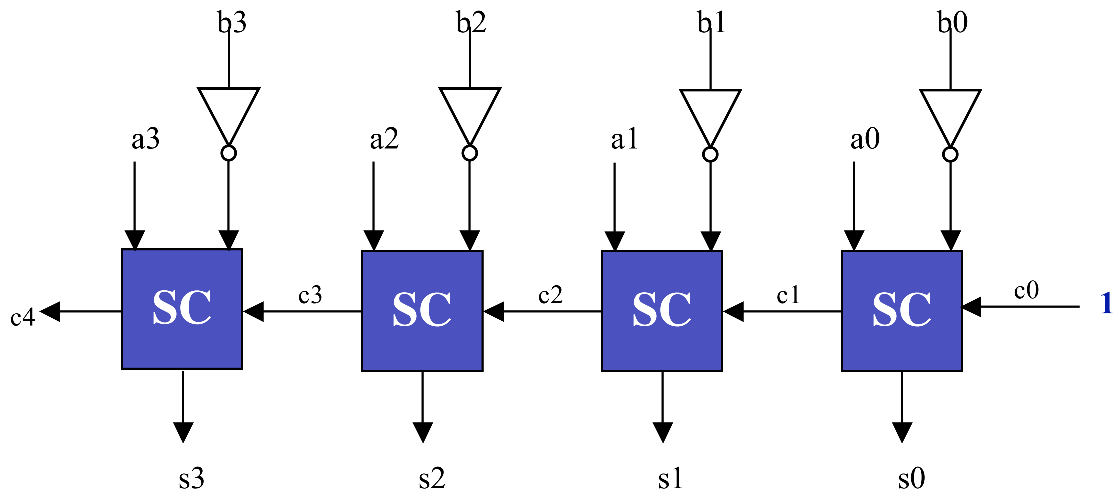




símbolo no
nível RT

Somador/Subtrator Paralelo

- Seria possível modificar este circuito, de modo que ele possa ser "configurado" para ser somador ou subtrator?



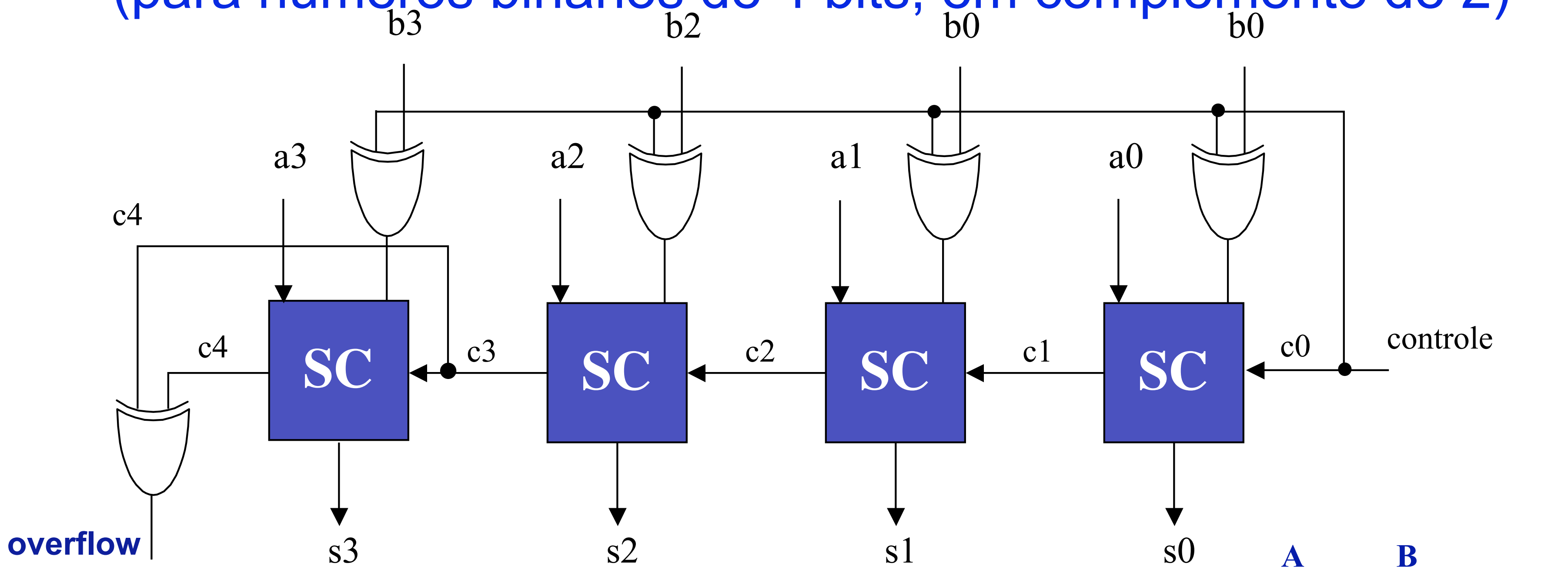
Resposta: Positivo! Modificações necessárias:

- Substituir os inversores por "negadores controlados" (xors)
- Controlar o valor de c0 (0 para adição/1 para subtração)

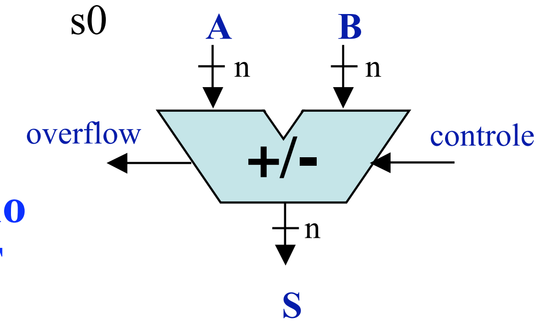
2. Circuitos Aritméticos

Somador/Subtrator Paralelo

(para números binários de 4 bits, em complemento de 2)

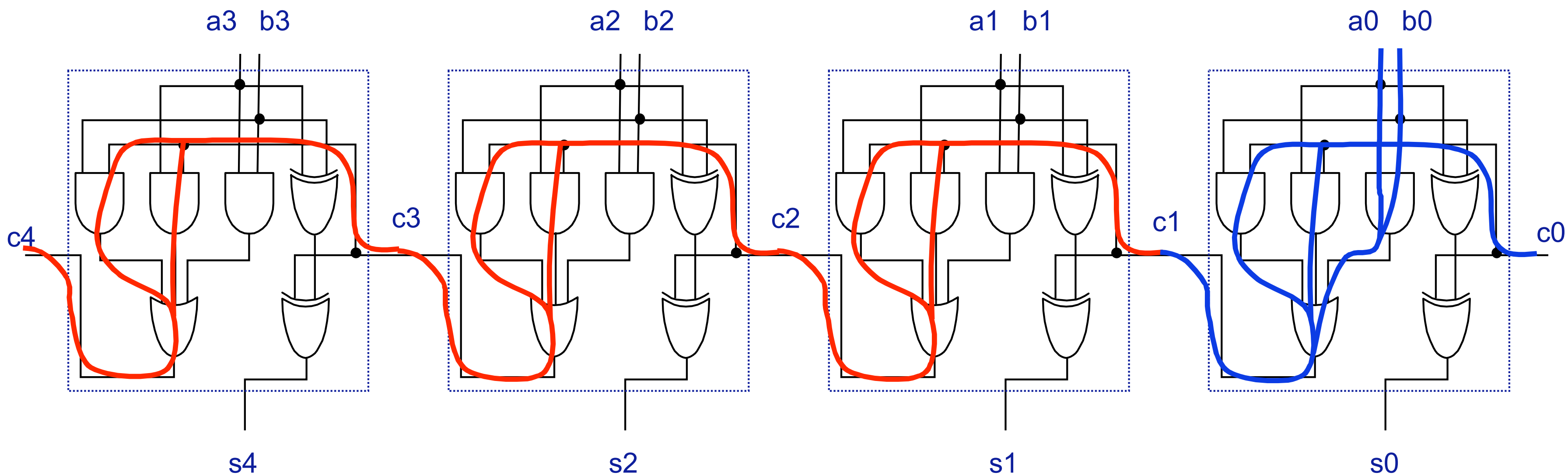


controle	operação
0	S=A+B
1	S=A-B



símbolo no nível RT

Somador Paralelo Tipo *Ripple Carry*

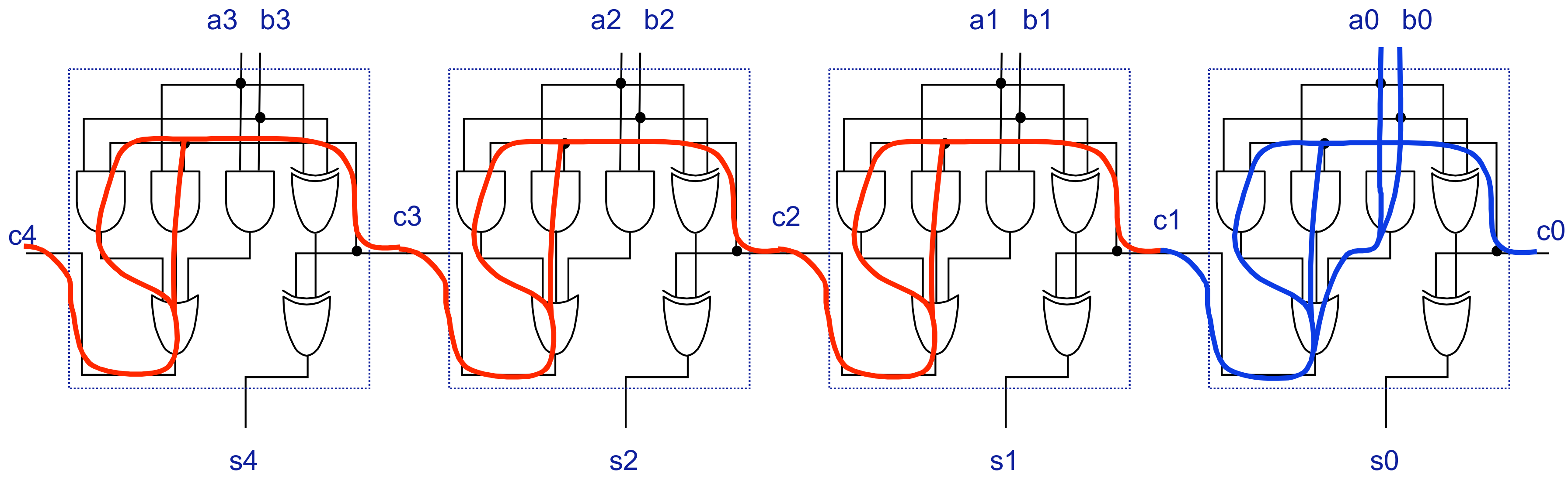


Suponha que no tempo t=0 um par de valores é aplicado às entradas (A,B):

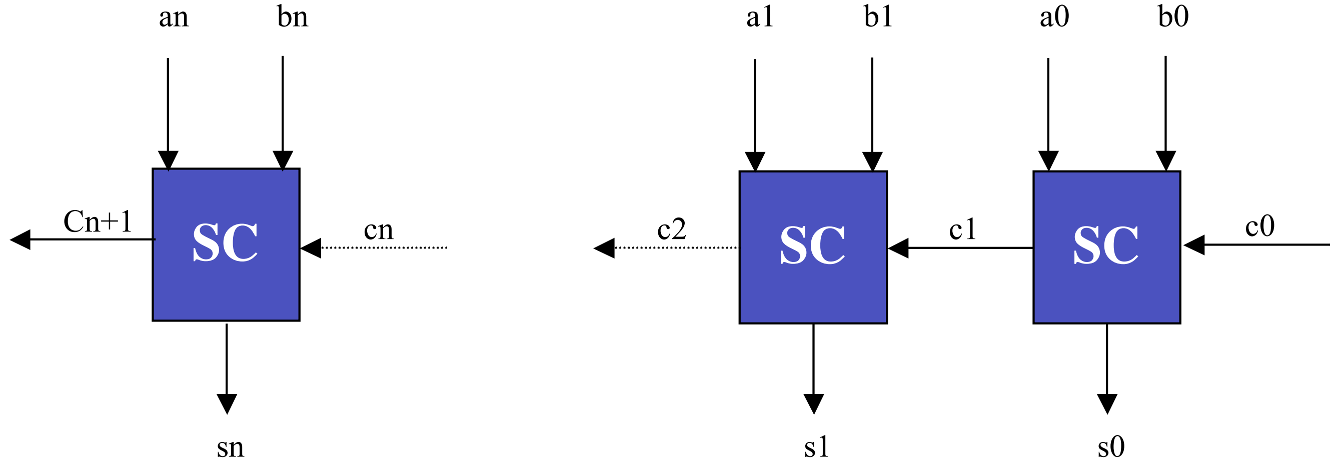
- O resultado só estará pronto quando todas as saídas tiverem estabilizado
- si e ci+1 só estabilizam após ci estabilizar
- **Em particular, s3 e c4 só estabilizam depois que c3, c2 e c1 tiverem estabilizados…**

Analisando a Propagação do *Carry*

Os caminhos críticos em somadores *Ripple Carry* passam pela cadeia de propagação do *carry*



Analisando a Propagação do *Carry*



Pergunta: será que não é possível alterar a arquitetura do somador, de modo a "quebrar" ou reduzir tal interdependência?

A resposta é ... **SIM!!!**

Analisando a Propagação do *Carry*

Há três diferentes casos na propagação do *carry* que são mutuamente exclusivos:

ai	bi	ci	ki	ci+1	si
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	1	1

Caso 1 (sinal ***Kill***)

Quando as entradas A e B do somador local são iguais a zero, **independentemente do *carry* de entrada**, o carry de saída será igual a **zero** e, portanto, o estágio "**matará**" a propagação do *carry* do estágio anterior.

Analisando a Propagação do *Carry*

ai	bi	ci	gi	ci+1	si
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Caso 2 (sinal ***Generate***)

Quando as entradas A e B do somador local **são iguais a um**, independentemente do *carry* de entrada, **o *carry* de saída será igual a um** e, portanto, **haverá geração de *carry*** neste estágio.

Analisando a Propagação do *Carry*

ai	bi	ci	pi	ci+1	si
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	0	1	1

Caso 3 (sinal ***Propagate***)

Quando uma das entradas, A ou B, do somador local for igual a um e a outra for igual a zero, **o *carry* de saída será igual ao *carry* de entrada** e, portanto, o *carry* será **propagado**.

Analisando a Propagação do *Carry*

Resumindo os três casos do *carry*, temos:

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Agora, a idéia é:

- Encontrar uma equação para cada um dos três sinais (k_i, p_i e g_i)
- Achar as equações de s_i e c_{i+1} em função de k_i, p_i e g_i

Mas qual é a vantagem desta "reestruturação lógica"?

Analisando a Propagação do *Carry*

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Equações das funções k_i , p_i e g_i são:

$$\mathbf{k_i} = \overline{a_i} \cdot \overline{b_i} = \overline{a_i + b_i}$$

$$\mathbf{p_i} = a_i \oplus b_i$$

$$\mathbf{g_i} = a_i \cdot b_i$$

Analisando a Propagação do *Carry*

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Então, usando soma de produtos, o *carry out* do estágio i pode ser definido como:

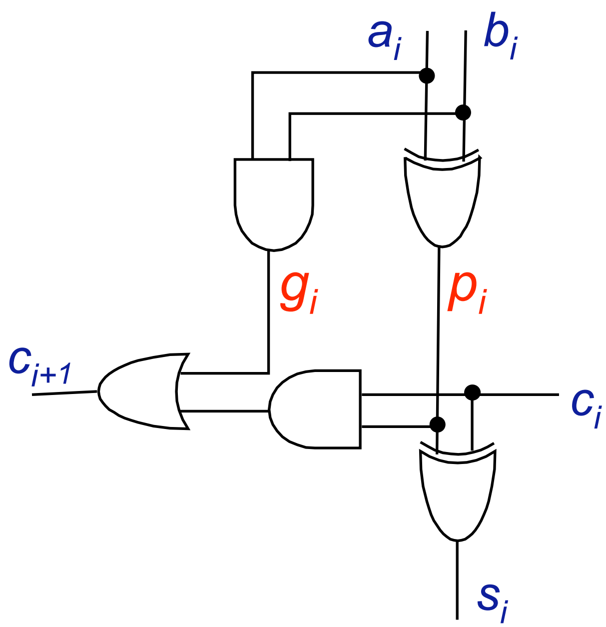
$$\mathbf{c_{i+1}} = g_i + p_i \cdot c_i$$

E a saída "soma" do estágio i pode ser expressa (em função de p_i e c_i) por:

$$\mathbf{s_i} = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$$

Analisando a Propagação do *Carry*

As fórmulas para s_i e c_{i+1} geram o mesmo circuito já estudado, com base em dois meio-somadores.



$\mathbf{p_i} = a_i \oplus b_i$

$\mathbf{g_i} = a_i \cdot b_i$

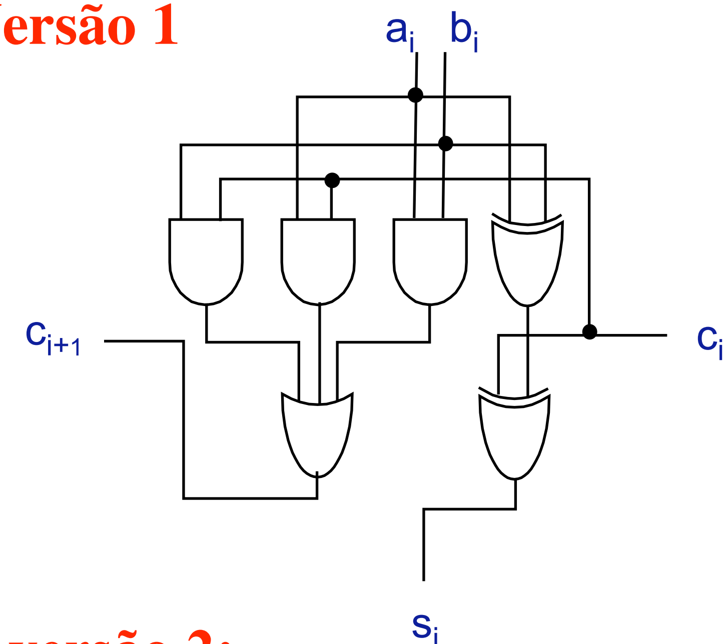
$\mathbf{c_{i+1}} = g_i + p_i \cdot c_i$

$\mathbf{s_i} = p_i \oplus c_i$

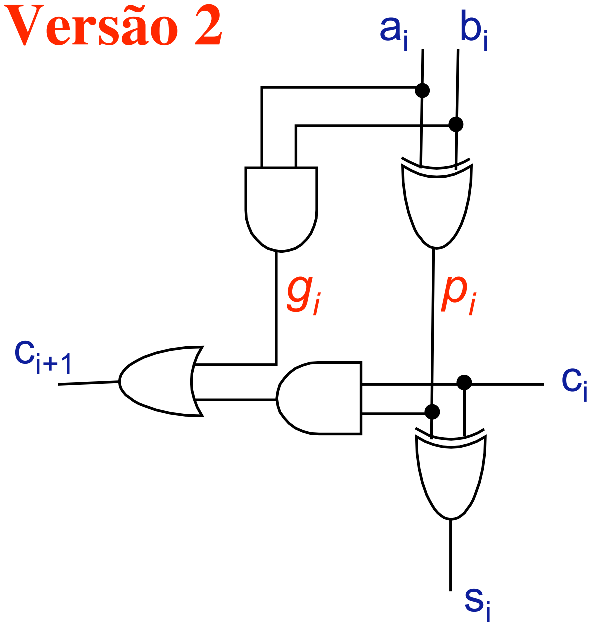
2. Circuitos Aritméticos

Comparando o Desempenho das Duas Versões

Versão 1



Versão 2



Na versão 2:

- Se g_i=1 não é preciso esperar pelo valor de c_i para determinar o valor de c_{i+1}
- Se p_i=1 , então já se sabe *a priori* que $c_{i+1} = c_i$

2. Circuitos Aritméticos

Somadores Rápidos

Para reduzir-se o atraso na propagação do *carry* as seguintes abordagens podem ser usadas:

1. **Reduzir o atraso na geração do *carry***
   (aplicada nos somadores Manchester);
2. **Diminuir o atraso da cadeia de propagação do *carry***
   (aplicada nos somadores *Carry-Lookahead*, *Carry-Select*, *Carry-Skip*, etc.);
3. **Mudar o sistema de representação numérica**
   (não será tratado na disciplina).

Estas soluções investem em desempenho, mas resultam em acréscimo de recursos (número de transistores utilizados).

Somadores Manchester *Chain*

Princípio: usar um circuito mais rápido para propagar a cadeia de *carry*;

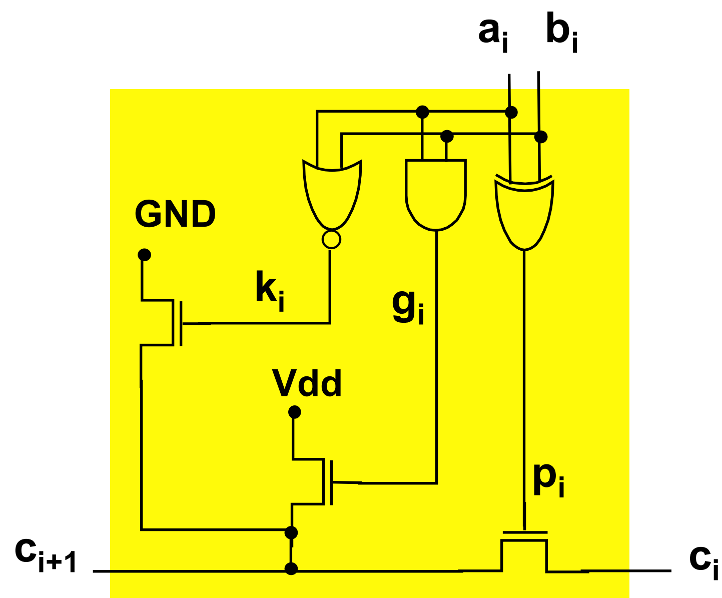
- **Este circuito mais rápido pode ser constituído de *Transmission Gates* ou transistores de passagem;**
- **Como as situações de *kill*, *propagate* e *generate* são mutuamente exclusivas, pode-se construir um conjunto de chaves com a seguinte lógica:**

  - Se $k_i = 1$, então $c_{i+1} = 0$
  - Se $g_i = 1$, então $c_{i+1} = 1$
  - Se $p_i = 1$, então $c_{i+1} = c_i$

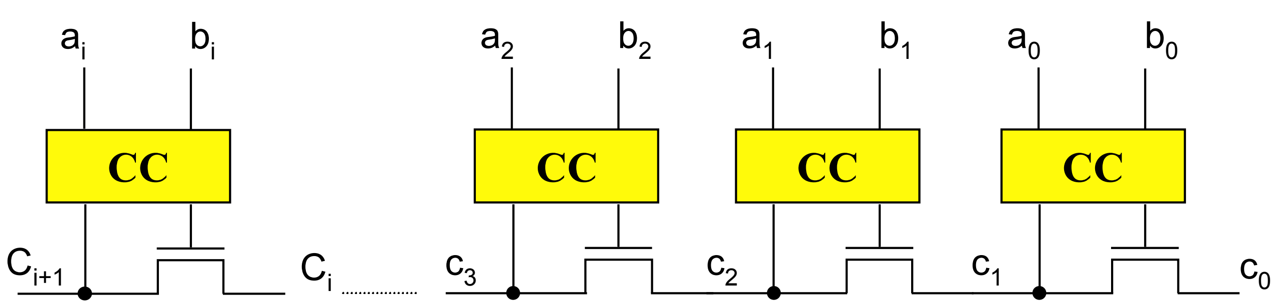
2. Circuitos Aritméticos

Somadores Manchester *Chain*

Uma Implementação com transistores de passagem



Controle da Cadeia de Propagação do *Carry* (CC)

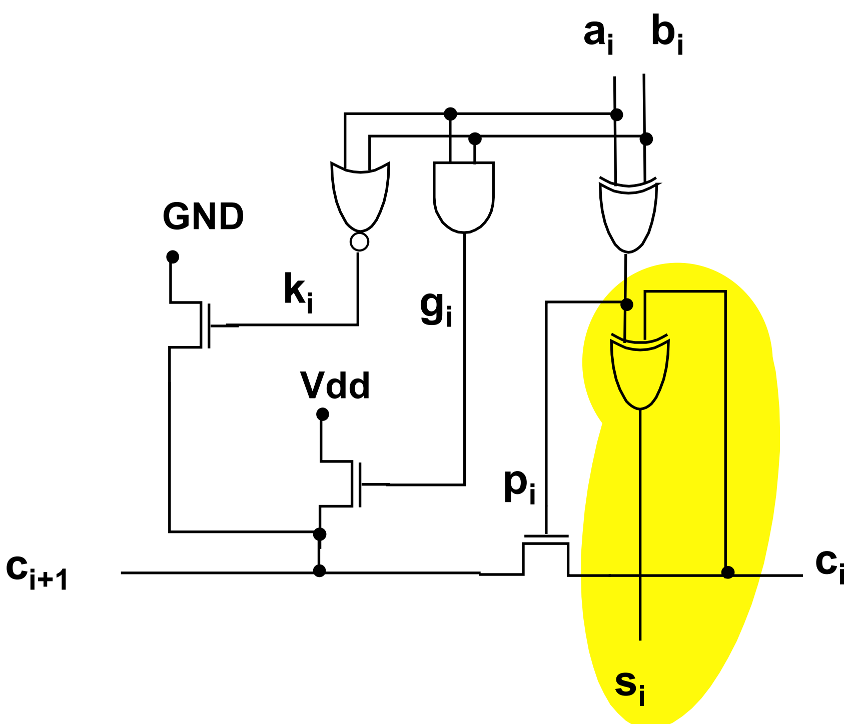


Cadeia de propagação do *Carry*

2. Circuitos Aritméticos

Somadores Manchester *Chain*

E como seria o somador *Manchester Chain* completo?



Basta modificar o controle da cadeia de propagação do *carry* para que ele calcule também a saída s_i

Somadores *Carry Lookahead*

Princípio: paralelizar o cálculo dos *carries*

- **No extremo, todos os *carries* podem ser computados ao mesmo tempo.**
- **Para o somador *Carry-Lookahead*, pode-se considerar que não existe mais a exclusividade mútua entre os sinais *generate* e *propagate*.**
- **Então, a função *propagate* pode ser simplificada para um simples "OU" entre as duas entradas, pois se o nível de soma gera *carry out* ($g_i = 1$), não importa o valor de *propagate*.**

Somadores *Carry Lookahead*

Assim, para o *carry-lookahead* temos que:

$$p_i = a_i + b_i$$

$$c_{i+1} = g_i + p_i \cdot c_i$$

Então:

$$\mathbf{c_1} = g_0 + p_0 \cdot c_0$$

$$\mathbf{c_2} = g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0)$$

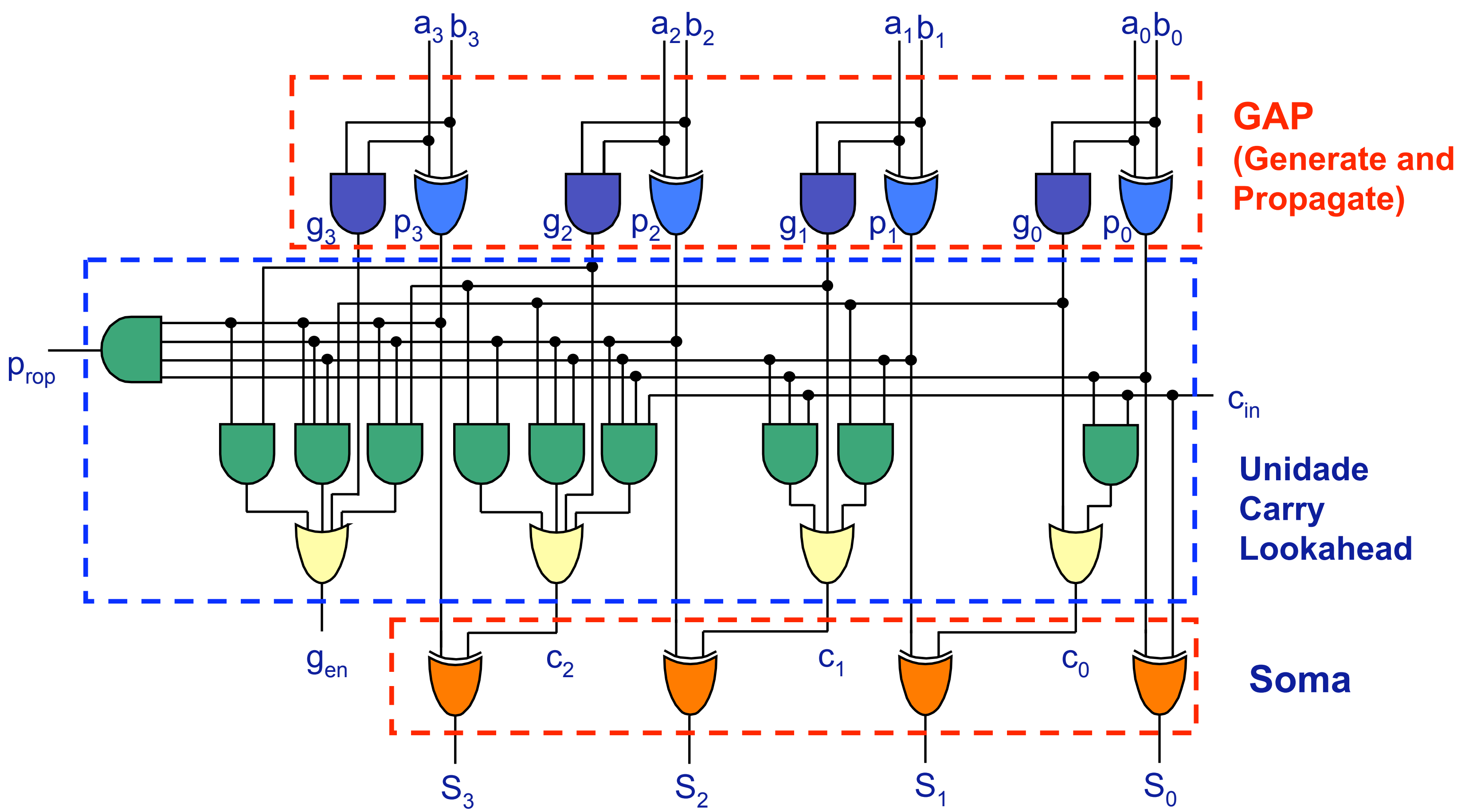
Expandindo, segue:

$$\mathbf{c_2} = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$$

$$\mathbf{c_3} = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

$$\mathbf{c_4} = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

Somadores *Carry Lookahead*



2. Circuitos Aritméticos

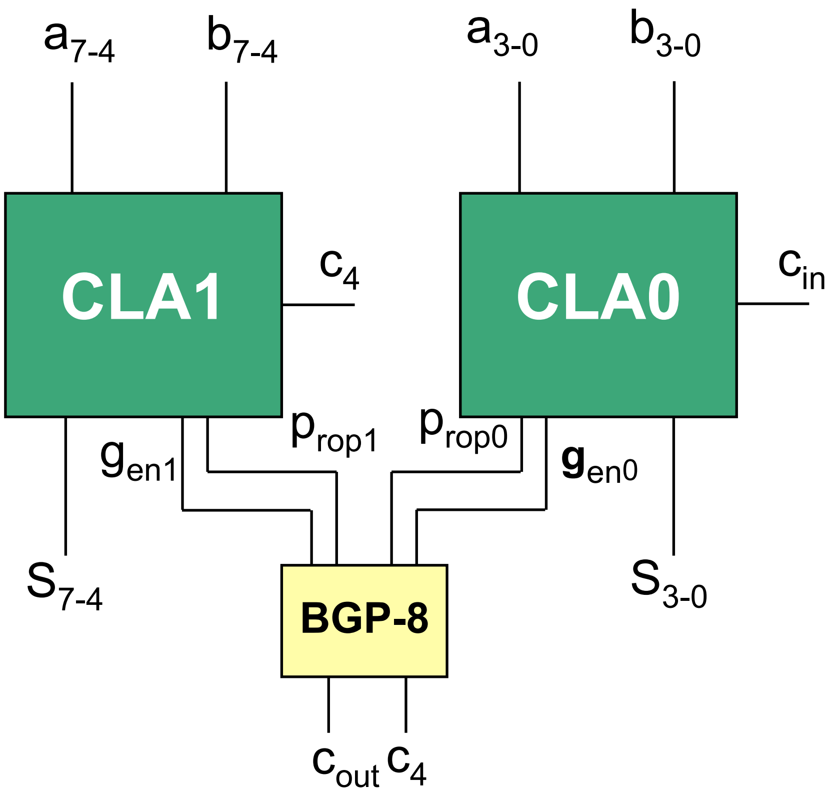
Somadores *Carry Lookahead*

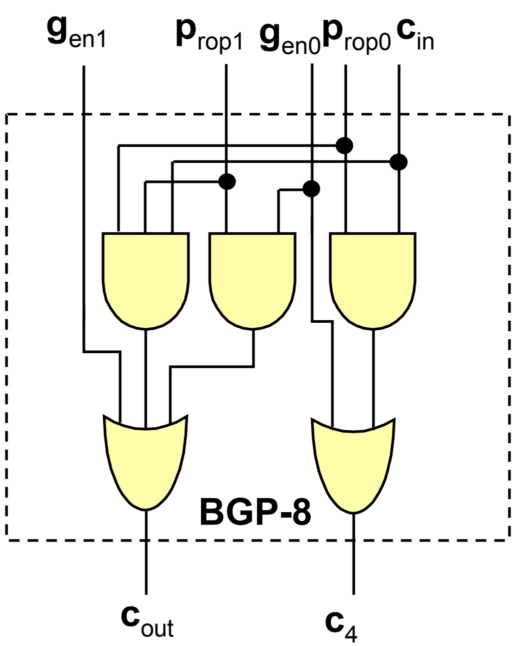
- **Problema com os CLGs: a complexidade da equação do *carry* cresce muito rapidamente!!!**
- **Para somadores com entradas usando muitos bits, a propagação do *carry* com cadeia *carry-lookahead* é mais lenta que um *ripple carry*.**
- **Tipicamente, o *carry-lookahead* não é usado para somadores com entradas maiores que 4 bits.**

Somadores *Carry Lookahead*

- **Solução:** aplicar o *lookahead* em grupos de somadores com, no máximo, **4 bits**.
- Funções auxiliares *P* e *G* são necessárias e indicam, respectivamente:
  - Se **P = 1**, então o *carry* é propagado pelo grupo.
  - Se **G = 1**, então o grupo gera *carry out*.

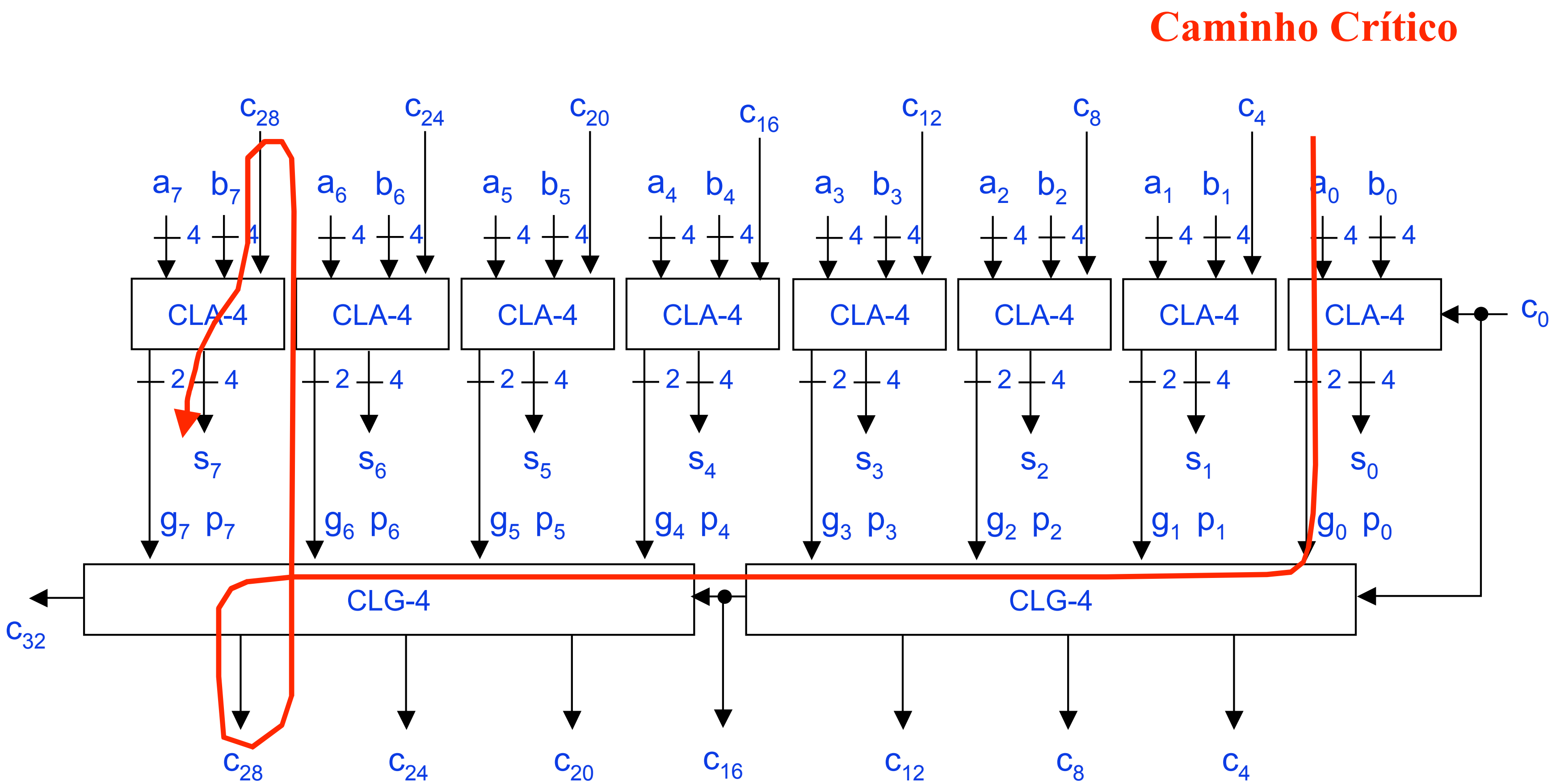
Somadores *Carry Lookahead*





2. Circuitos Aritméticos

Somadores *Carry Lookahead*



2. Circuitos Aritméticos

Somadores *Carry Select*

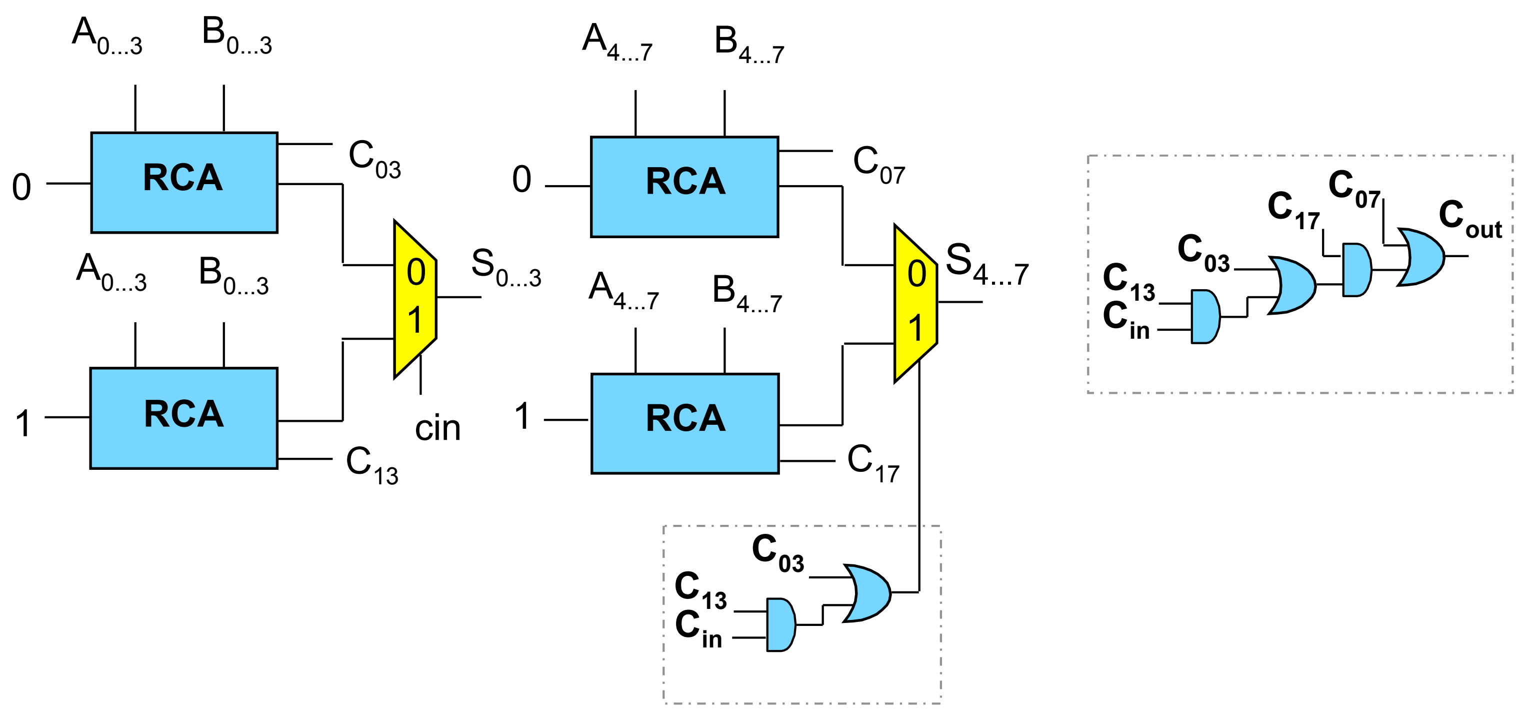
Princípios:

- **dividir a adição em seções de 4 ou 8 bits**
- **realizar a adição de cada seção simultaneamente para os dois casos possíveis (*carry in*=0 e *carry in*=1)**

- **Em cada seção de adição são usados dois somadores (*ripple carry ou carry lookahead*) idênticos e um multiplexador**
- **O multiplexador seleciona um dos dois resultados, utilizando como controle o *carry out* da seção anterior**

2. Circuitos Aritméticos

Somadores *Carry Select* com 8 bits
(usando somadores RCA...)

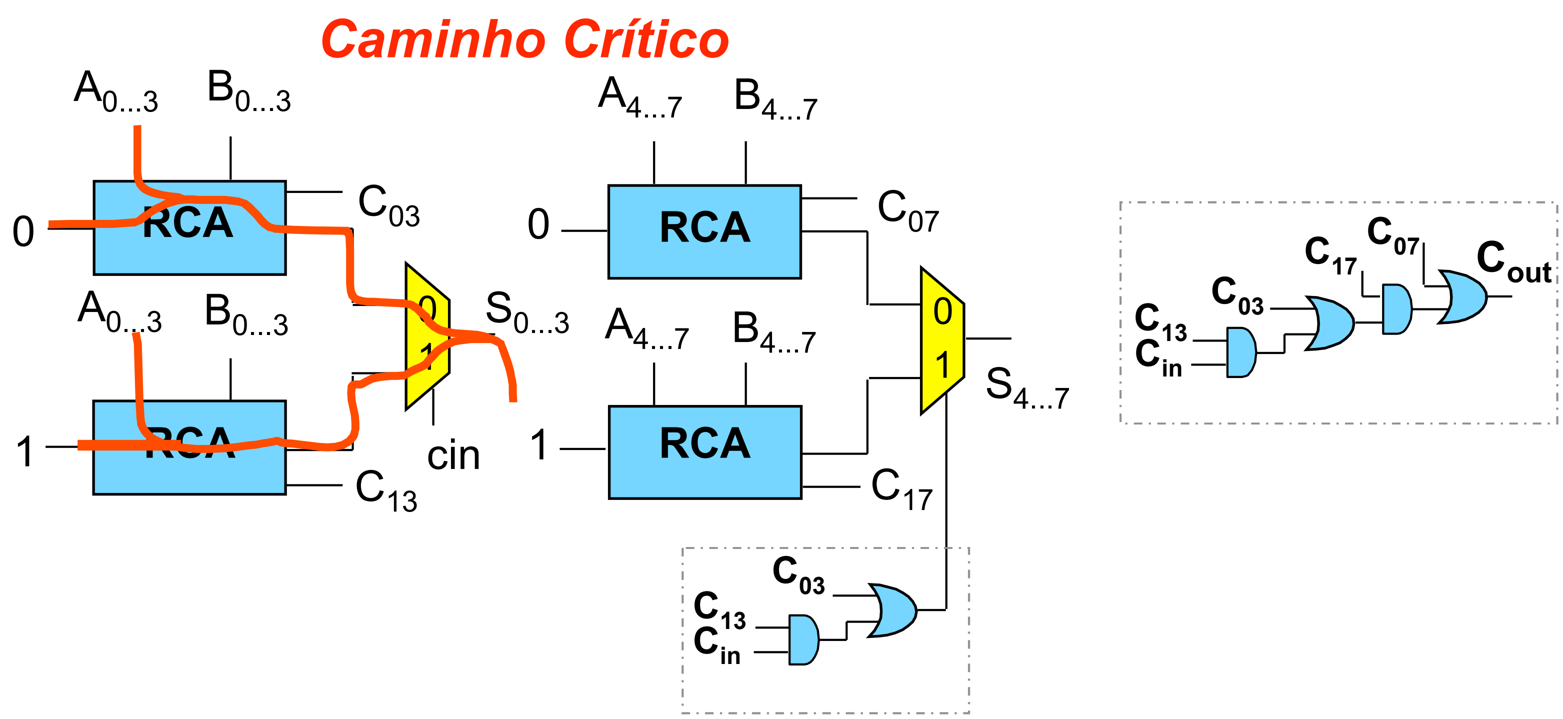


Somadores *Carry Select*

Equações utilizadas pelos MUX de cada seção de soma

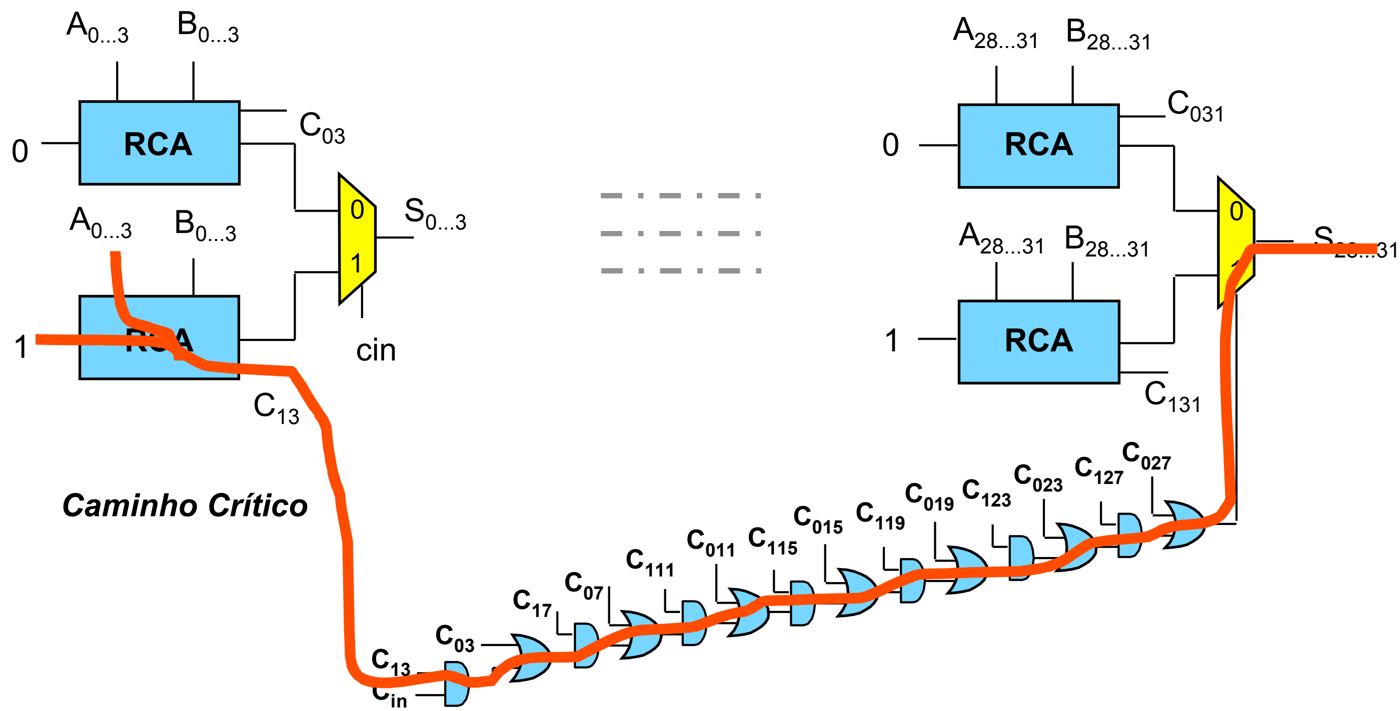
B1 = Cin
B2 = C03 or (C13 and CIN)
B3 = C07 or (C17 and (C03 or (C13 and CIN)))
B4 = C011 or (C111 and (C07 or (C17 and (C03 or (C13 and CIN)))))
B4 =C015 or (C115 and(C011 or (C111 and (C07 or (C17 and (C03 or (C13 and CIN))))))

Somadores *Carry Select* com 8 bits



Atraso crítico = Atraso RCA + Atraso MUX

Somadores *Carry Select* com 32 bits



Atraso crítico = Atraso RCA + Atraso Equação + Atraso MUX