

Capítulo 2–Projeto Lógico Combinacional G

Profa. Eliete Caldeira

Módulos padrão aritméticos

- ▶ Módulos–padrão aritméticos: somadores, subtratores, Unidades Lógicas e Aritméticas (ALUs), comparadores, multiplicadores

Somadores

- ▶ Um somador de N bits é um componente de bloco operacional que adiciona dois números A e B de N bits gerando:
 - Uma soma S de N bits e
 - Um transporte (“vai um” ou “carry out”) C de 1 bit

Ex: somador
de 2 bits

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Figure 4.24 Truth table for a 2-bit adder.

Somadores

- ▶ Para somadores com larguras maiores, a tabela verdade passa a ser de difícil construção (somador de 16 bits tem mais de 4 bilhões de linhas)
- ▶ Por isto, somadores não são construídos usando a lógica de 2 níveis

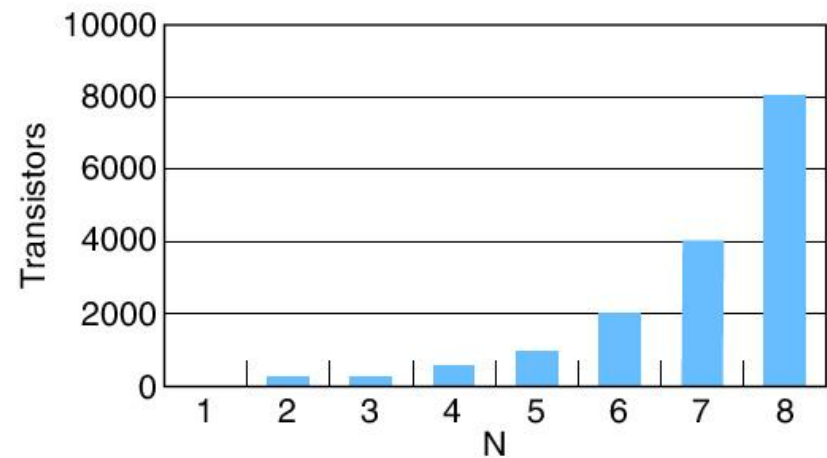


Figure 4.25 Why large adders aren't built using standard two-level combinational logic—notice the exponential growth. How many transistors would a 32-bit adder require?

Somadores

- ▶ Como fazer um circuito somador que não gaste tanta porta com tantas entradas?

Somadores

- ▶ Como fazer um circuito somador que não gaste tanta porta com tantas entradas?
- ▶ Seguindo o raciocínio de contas a mão!

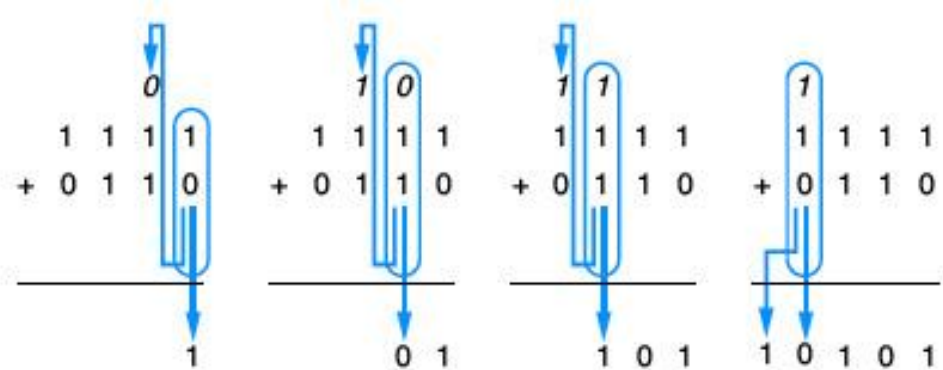


Figure 4.26 Adding two binary numbers by hand, column by column.

Somadores

- ▶ Um Meio-Somador (*half-adder*): é um componente combinacional que adiciona dois bits (a e b) e gera uma soma (s) e um bit de transporte de “vai um” (c_o)
- ▶ Pela tabela-verdade:

$$s = a \text{ xor } b$$

$$c_o = a \text{ and } b$$

Inputs		Outputs	
a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 4.28 Truth table for a half-adder.

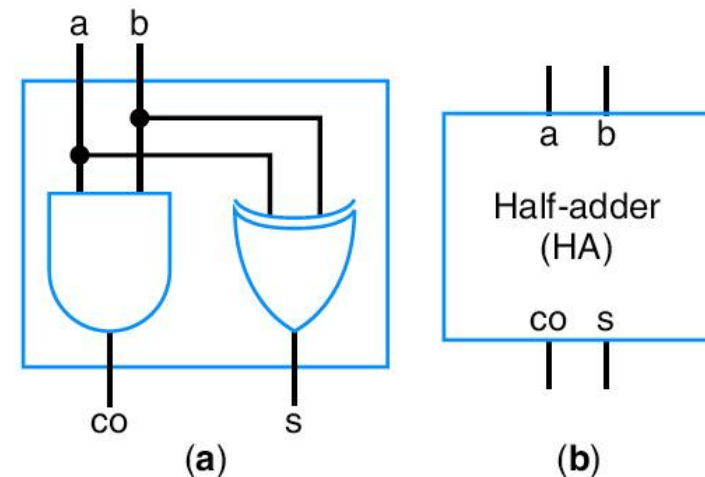


Figure 4.29 Half-adder: (a) circuit, and (b) block symbol.

Somadores

- Um Somador Completo (*full-adder*) é um componente combinacional que adiciona três bits (a , b e c_i) e gera uma soma (s) e um bit de transporte de “vai um” (c_o)

$$c_o = a'bc_i + ab'c_i + abc_i' + abc_i$$

$$c_o = a'bc_i + abc_i + ab'c_i + abc_i + abc_i' + abc_i$$

$$c_o = (a' + a)bc_i + (b' + b)ac_i + (c' + c)ab$$

$$c_o = bc_i + ac_i + ab$$

$$s = a'b'c_i + a'bc_i' + ab'c_i' + abc_i$$

$$s = a'(b'c_i + bc_i') + a(b'c_i' + bc_i)$$

$$s = a'(b \text{ xor } c_i) + a(b \text{ xor } c_i)'$$

$$s = a \text{ xor } b \text{ xor } c_i$$

Inputs			Outputs	
a	b	ci	co	s
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

Figure 4.30 Truth table for a full-adder.

Somadores

- ▶ Um Somador Completo (*full-adder*) é um componente combinacional que adiciona três bits (a , b e c_i) e gera uma soma (s) e um bit de transporte de “vai um” (c_o)
- ▶ $s = a \text{ xor } b \text{ xor } c_i$
- ▶ $c_o = b.c_i + a.c_i + a.b$

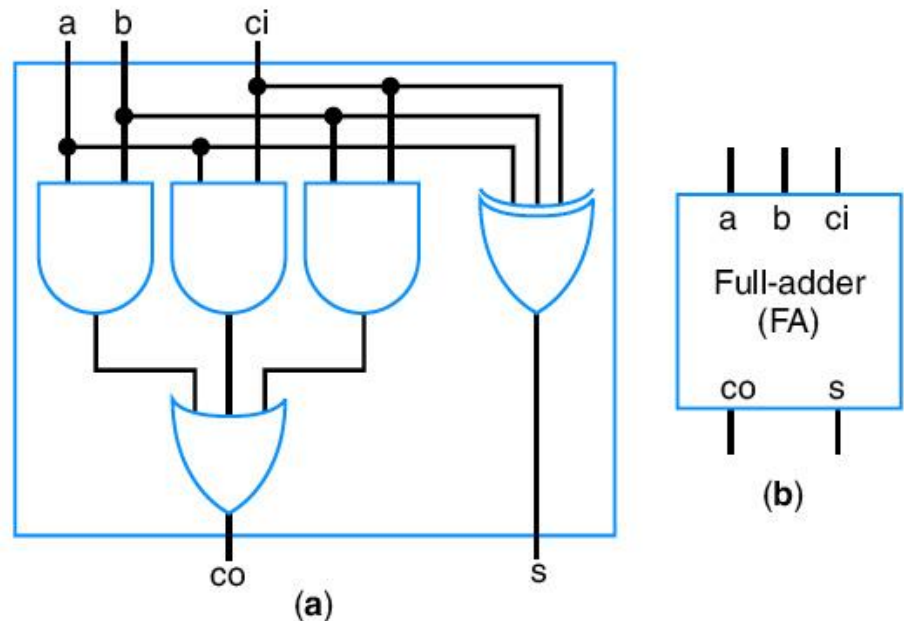


Figure 4.31 Full-adder: (a) circuit, and (b) block symbol.

Somadores

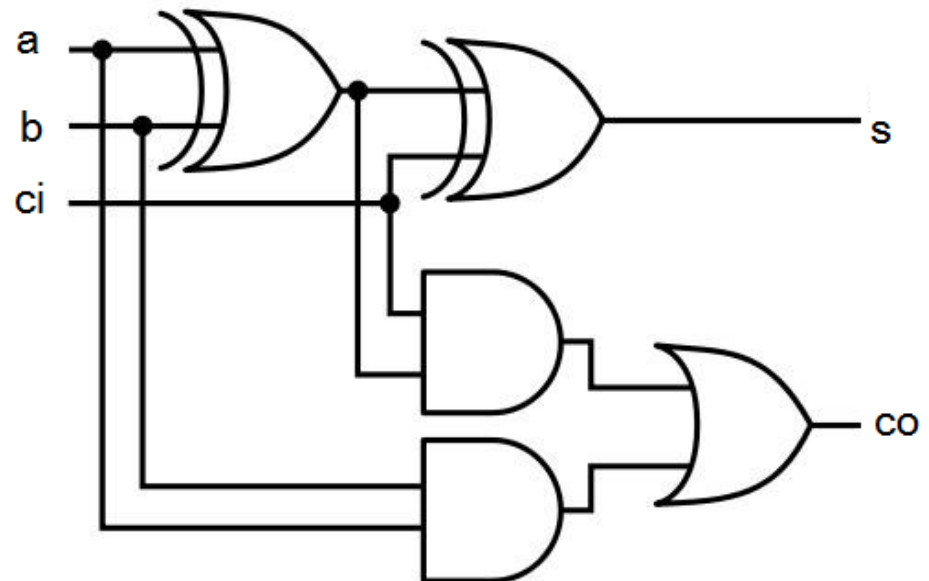
- ▶ Outra forma de implementar o Somador Completo (*full-adder*) pode ser obtida manipulando as expressões da função

$$c_o = a'.b.c_i + ab'.c_i + abc_i' + abc_i$$

$$c_o = c_i(a'b + ab') + ab(c_i' + c_i)$$

$$c_o = c_i(a \text{ xor } b) + ab$$

$$s = a \text{ xor } b \text{ xor } c_i$$



Somadores

- Somador de números de 4 bits

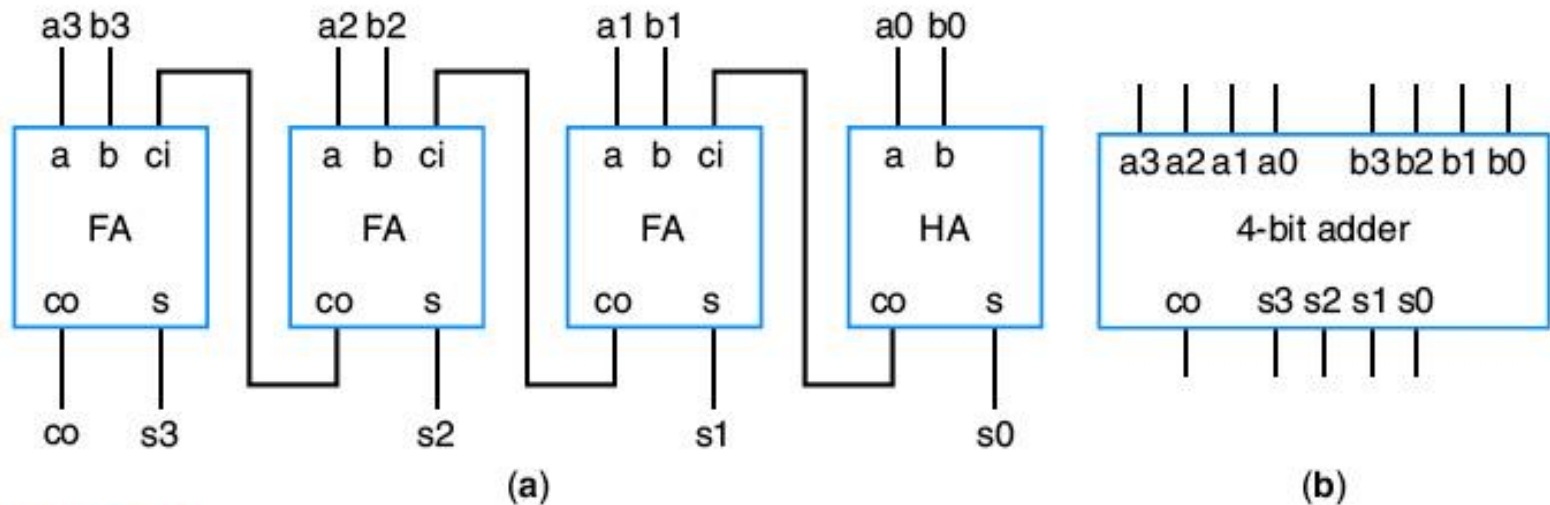


Figure 4.32 4-bit adder: (a) carry-ripple implementation with 3 full-adders and 1 half-adder, and (b) block symbol.

Somadores

- ▶ Somador de números de 4 bits com *carry* de entrada (“vem-um”)

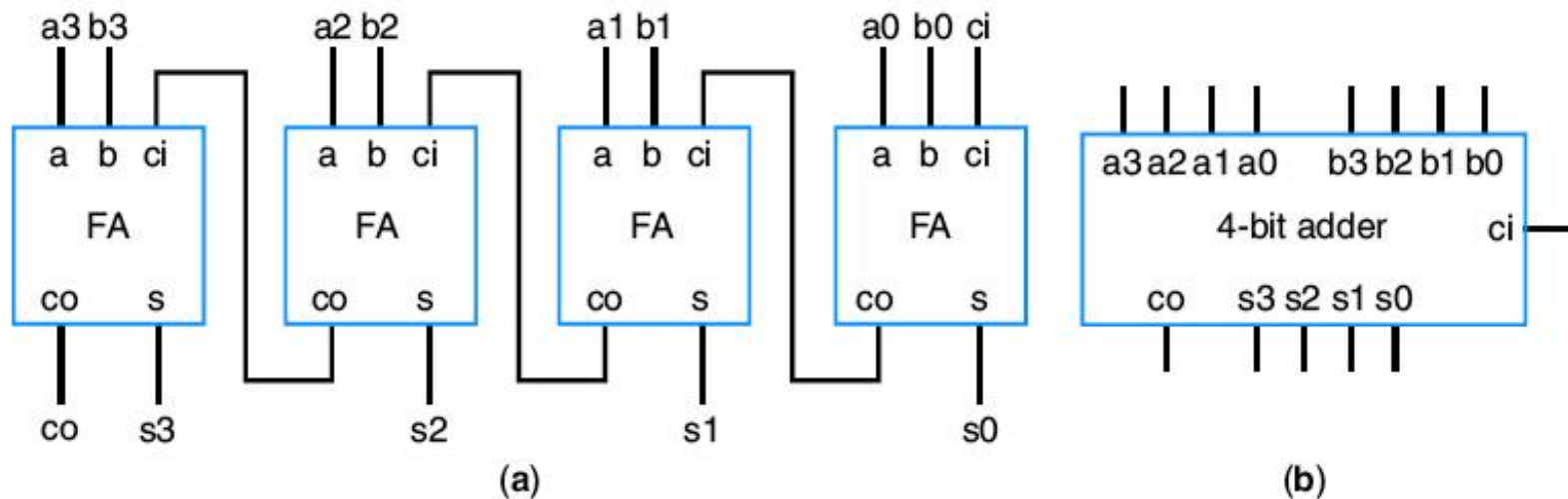


Figure 4.33 4-bit adder: (a) carry-ripple implementation with 4 full-adders, with a carry-in input, and (b) block symbol.

- ▶ Este é um somador de transporte propagado. Como fica o atraso?

Somadores

- Exemplo de uso:
Soma de 0111 e
0001

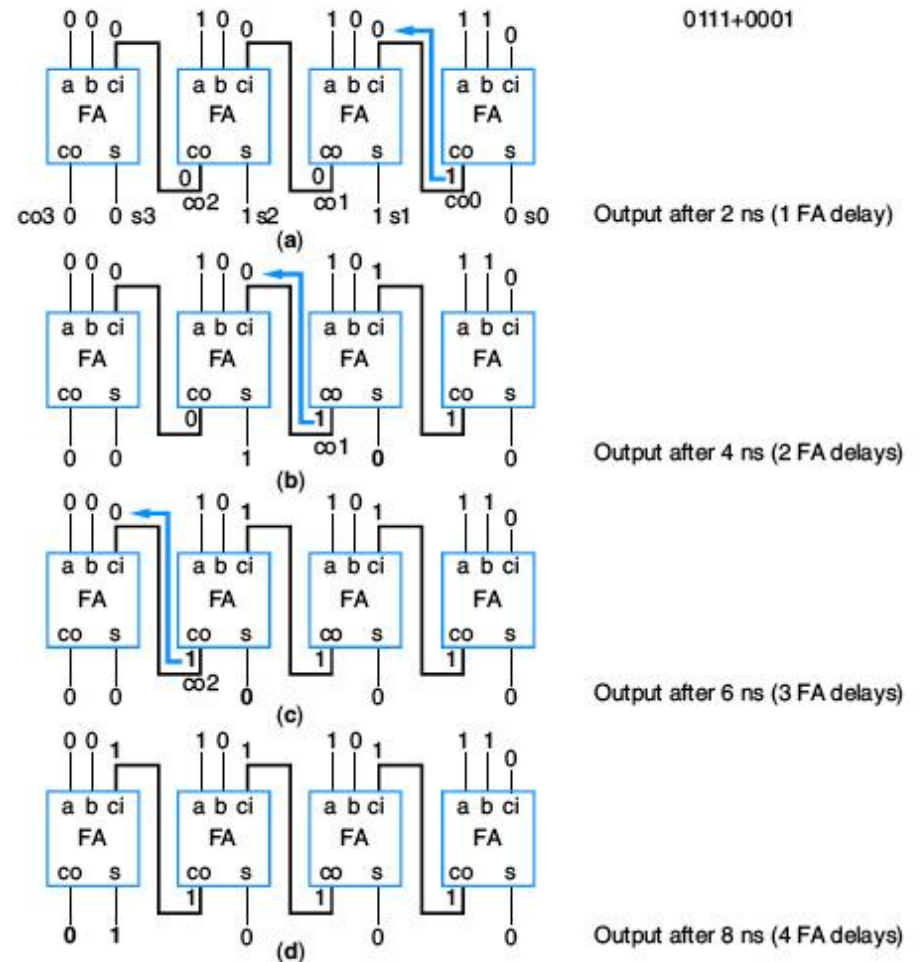


Figure 4.34 Example of adding 0111+0001 using a 4-bit carry-ripple adder. The output will exhibit temporarily incorrect (spurious) results until the carry bit from the rightmost bit has had a chance to propagate (ripple) all the way through to the leftmost bit.

Somadores

- Somador de $2N$ bits usando dois somadores de N bits

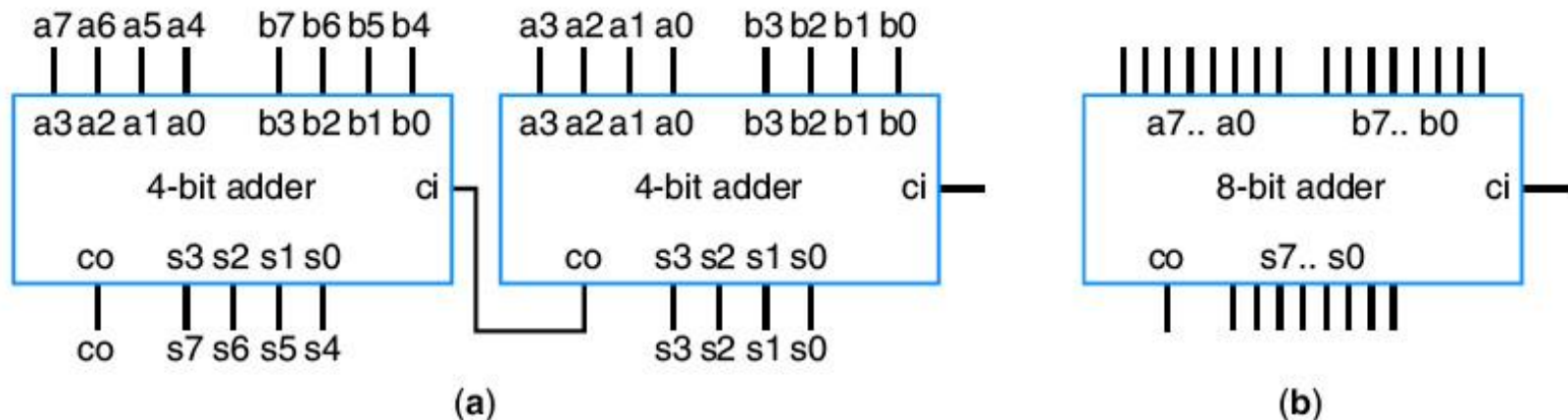


Figure 4.35 8-bit adder: (a) carry-ripple implementation built from two 4-bit carry-ripple adders, and (b) block symbol.

Subtratores

- ▶ Um subtrator de N bits é um componente de bloco operacional que toma duas entradas binárias A e B e produz um resultado S na saída igual a $A-B$.
- ▶ Exemplos:
 - $6-2 = 4$
 - $5-7 = -2$
- ▶ Mas como representar números negativos???

Representação de números negativos – Sinal e Magnitude

- ▶ Sinal e Magnitude
- ▶ Bit de ordem mais elevada:
 - Sinal do número: 0 → positivo e 1 → negativo.
- ▶ Bits de ordem inferior:
 - Representam a magnitude ou módulo do número.
- ▶ Ex.: $7 = 0111$ e $-7 = 1111$
- ▶ Representa números entre
 - $-(2^{n-1}-1) \leq x \leq (2^{n-1}-1)$
- ▶ Problemas:
 - Pouco eficiente
 - Representação dupla para o 0 (0000 e 1000)

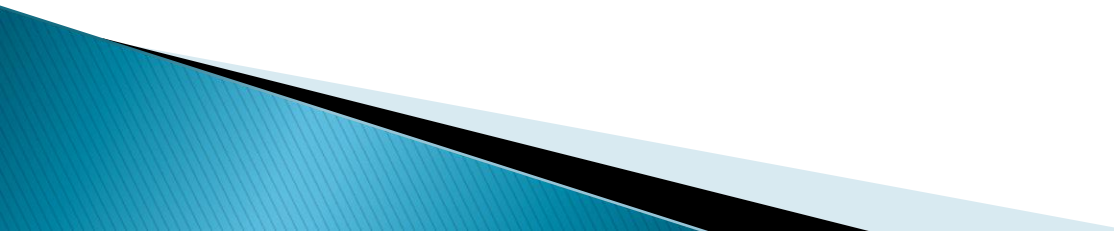
Representação de números negativos – Complemento

- ▶ Pode-se usar a adição para se fazer a subtração usando complementos
- ▶ Na base 10:
 $7 - 4 = ?$
 $= 7 + \text{Complemento}(4)$
 $= 7 + 6 = 13$
 $= 3$ (descartando o “vai um”)
- ▶ Assim, somar o complemento produz uma resposta com exatamente 10 a mais
- ▶ Descartando a coluna das dezenas, obtemos o resultado

Complemento:
o que falta para
se chegar a 10

1	→	9
2	→	8
3	→	7
4	→	6
5	→	5
6	→	4
7	→	3
8	→	2
9	→	1

Representação de números negativos – Complemento

- ▶ Para determinar o complemento na base 10, é preciso fazer uma subtração. Então não há grande vantagem no método
 - ▶ Como obter complemento na base 2?
- 

Representação de números negativos – Complemento de dois

- ▶ Para determinar o complemento de dois de um número representado na base 2:
- ▶ **Basta inverter todos os bits e somar 1 !!!**
- ▶ Esta é uma forma de representar números negativos!

EX:

0101	\rightarrow	Complemento 2 (0101) =	1010	+ 1	=	1011
5						-5
			↖ Inversão	↖ Soma 1		

Representação de números negativos – Complemento de dois

- ▶ Representação de números negativos
- ▶ Apenas 1 representação para o 0
- ▶ Isto implica em uma representação assimétrica. Representa números de :
 - $-(2^{n-1}) \leq x \leq (2^{n-1}-1)$
- ▶ Ex: Um número de 4 bits vai de -8 a +7
- ▶ Bit mais significativo: SINAL

Bits	Valor
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Representação de números negativos – Complemento de um

- ▶ É obtida invertendo todos os bits do número
- ▶ Representa números de :
 - $-(2^{n-1}-1) \leq x \leq (2^{n-1}-1)$
- ▶ Ex: $+2 = 0010$ e $-2 = 1101$
- ▶ Tem duas representações para o 0 (0000 e 1111)
- ▶ Bit mais significativo: SINAL

Subtração

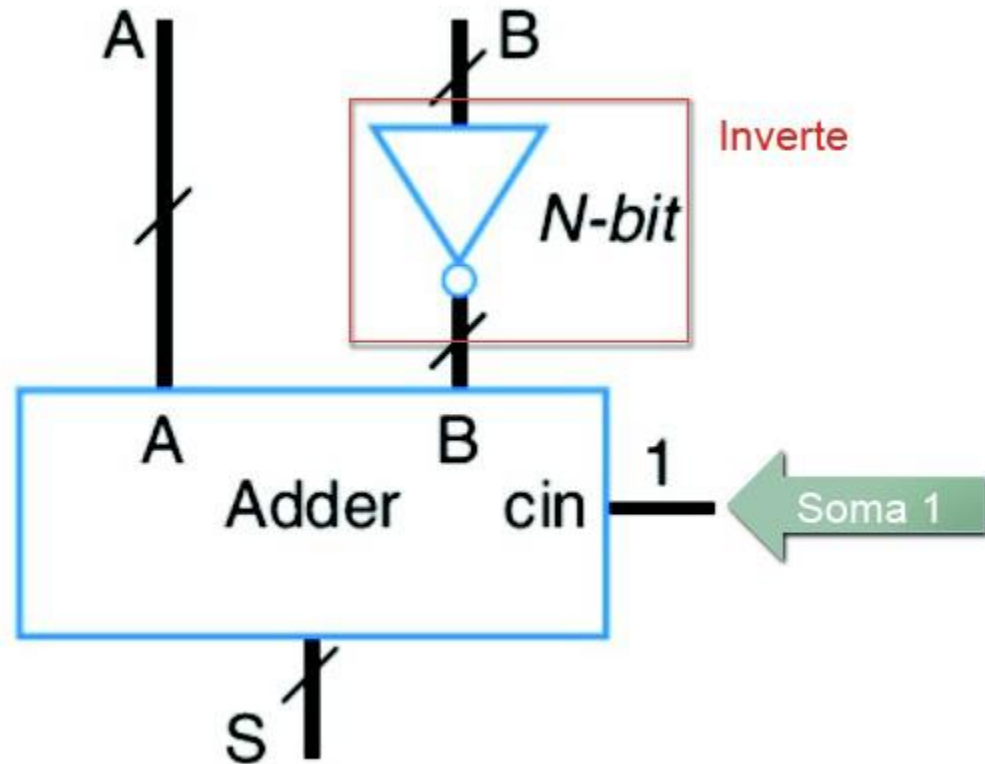
- ▶ Usando complemento de dois:
- ▶ Efetue:
 - $10 - 12 = ?$
 - $130 - 53 = ?$

Subtração

- ▶ Como fazer um subtrator usando um somador?

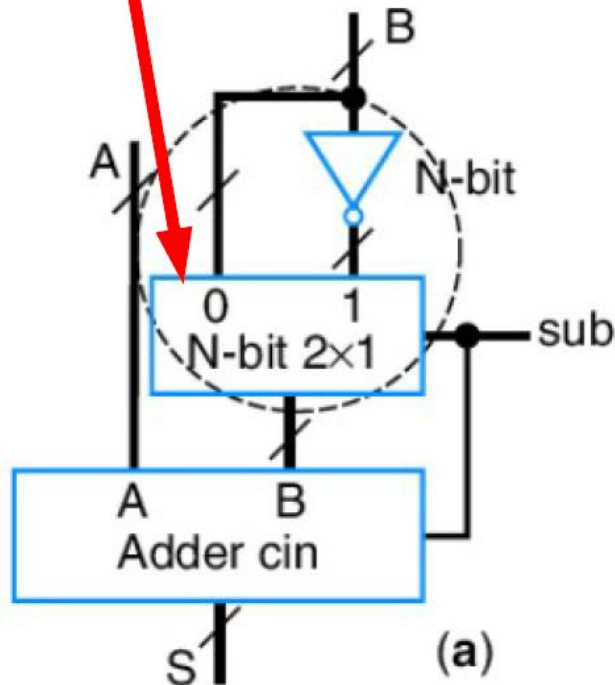
Subtração

- ▶ Como fazer um subtrator usando um somador?



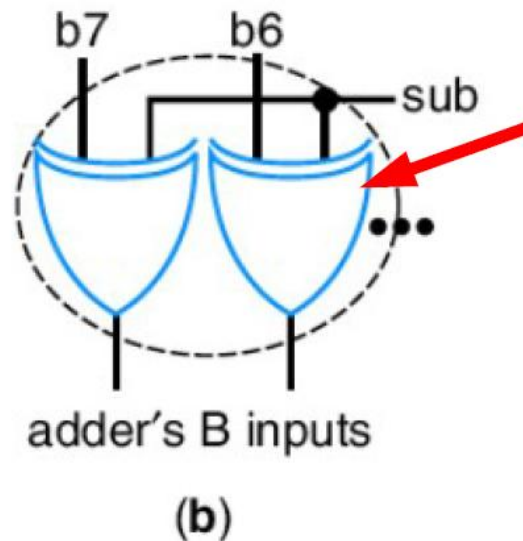
Somador/subtrator

MUX para selecionar entre B e B' de acordo com o sinal sub



Para:

sub = 0, soma
sub = 1, subtrai



Ou um conjunto de portas XOR para gerar B' quando sub = 1

Detecção de Estouro

- ▶ Estouro ou transbordamento (*overflow*): quando realizamos aritmética usando números binários de largura fixa de bits, algumas vezes o resultado tem largura maior do que essa largura fixa.
- ▶ Ex. 1: Número positivos sem complemento de 2
 $1111 + 0001 = 10000$ (de cinco bits)
Fácil, basta olhar o vai um do somador!
- ▶ Ex. 2: Positivos em complementos de 2
 $0111 + 0001 = 1000$
 $7 + 1 = -8$????????
- ▶ Quando somamos 2 número positivos, basta olhar se o resultado é negativo!

Detecção de Estouro

- ▶ Ex. 3: Negativos em complementos de 2
 $1111 + 1000 = 1\ 0111$
- ▶ Quando somamos 2 número negativos, basta olhar se a resultado é positivo!

- ▶ Ex. 4: E se somamos um positivo e um negativo???

NUNCA PODE OCORRER ESTOURO!

- ▶ O resultado nunca será mais positivo que o número positivo ou mais negativo que o número negativo

Detecção de estouro

- ▶ Em resumo...
- ▶ Para detecção do estouro sem complemento de 2 basta olhar o vai-um
 - **estouro = vai-um**
- ▶ A detecção do estouro em complemento de 2 envolve a detecção de:
 - Se ambos os números de entrada são positivos e produzem um resultado negativo, ou
 - Se ambos os números de entrada são negativos e produzem um resultado positivo
- ▶ Considerando $a(a_{N-1} \dots a_0)$, $b(b_{N-1} \dots b_0)$ e $r(r_{N-1} \dots r_0)$ em complemento de 2
estouro = $a_{N-1}b_{N-1}r_{N-1}' + a_{N-1}'b_{N-1}r_{N-1}$

Detecção de estouro

- ▶ Ou ainda:
- ▶ Comparar o bit de transporte que entra na coluna do bit de sinal com o bit de transporte que sai dessa mesma coluna, o “vai um”, quando o bit de transporte que entra e diferente do que sai, houve estouro.

sign bits

$$\begin{array}{rcccc} \textcircled{0} & 1 & 1 & 1 \\ + & \textcircled{0} & 0 & 0 & 1 \\ \hline \textcircled{1} & 0 & 0 & 0 \\ \text{overflow} \end{array}$$

$$\begin{array}{rcccc} \textcircled{1} & 1 & 1 & 1 \\ + & \textcircled{1} & 0 & 0 & 0 \\ \hline \textcircled{0} & 1 & 1 & 1 \\ \text{overflow} \end{array}$$

$$\begin{array}{rcccc} \textcircled{1} & 0 & 0 & 0 \\ + & \textcircled{0} & 1 & 1 & 1 \\ \hline \textcircled{1} & 1 & 1 & 1 \\ \text{no overflow} \end{array}$$

Para ser continuado...