

# Arquitetura e Organização de Computadores

## Aula-09: Aritmética do Computador

Eliseu César Miguel

Livro: Arquitetura e Organização de Computadores  
William Stallings 8<sup>a</sup> Edição  
Universidade Federal de Alfenas

December 22, 2020



# Organização da Aula

- 1 Introdução
- 2 Operações sobre Complemento de 2
- 3 Operações sobre Inteiros Sem Sinal
- 4 Operações sobre Inteiros Com Sinal
- 5 Operações sobre Ponto Flutuante

# Organização da Aula

- 1 Introdução
- 2 Operações sobre Complemento de 2
- 3 Operações sobre Inteiros Sem Sinal
- 4 Operações sobre Inteiros Com Sinal
- 5 Operações sobre Ponto Flutuante

# Organização da Aula

- 1 Introdução
- 2 Operações sobre Complemento de 2
- 3 Operações sobre Inteiros Sem Sinal
- 4 Operações sobre Inteiros Com Sinal
- 5 Operações sobre Ponto Flutuante

# Organização da Aula

- 1 Introdução
- 2 Operações sobre Complemento de 2
- 3 Operações sobre Inteiros Sem Sinal
- 4 Operações sobre Inteiros Com Sinal
- 5 Operações sobre Ponto Flutuante

# Organização da Aula

- 1 Introdução
- 2 Operações sobre Complemento de 2
- 3 Operações sobre Inteiros Sem Sinal
- 4 Operações sobre Inteiros Com Sinal
- 5 Operações sobre Ponto Flutuante

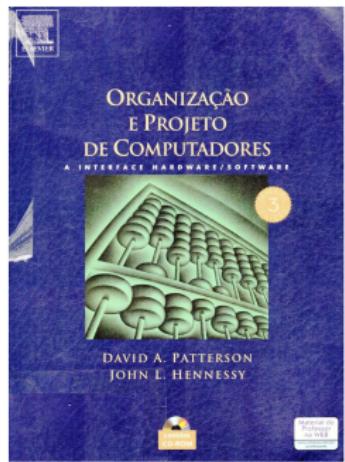
# Bibliografia básica



Livro Texto



Complementar



Complementar

Além da bibliografia básica, visite o programa de ensino para ver outras bibliografias. Também, durante o curso, várias bibliografias em sítios da Internet serão apresentadas.

# Introdução: Elementos necessários para este capítulo

O Conteúdo visto na disciplina de Lógica Digital é a base para a arquitetura e, em particular, para esse capítulo. Dentre os conceitos mais importantes, sugiro que seja feita uma revisão sobre:

- Sistemas numéricicos
- Representação de números em diferentes bases
- Representação de números com sinal
- Representação de números em ponto flutuante
- Operações sobre complemento de 2 e ponto flutuante
- Circuitos lógicos

# Operações em Complemento de 2: Adição e subtração

## Adição e subtração em Complemento de 2 e ocorrência de *overflow*

**Figura 9.3** Adição de números na representação do complemento de dois

$\begin{array}{r} 1001 = -7 \\ +0101 = 5 \\ \hline 1110 = -2 \end{array}$ <p>(a) <math>(-7) + (+5)</math></p>	$\begin{array}{r} 1100 = -4 \\ +0100 = 4 \\ \hline 10000 = 0 \end{array}$ <p>(b) <math>(-4) + (+4)</math></p>
$\begin{array}{r} 0011 = 3 \\ +0100 = 4 \\ \hline 0111 = 7 \end{array}$ <p>(c) <math>(+3) + (+4)</math></p>	$\begin{array}{r} 1100 = -4 \\ +1111 = -1 \\ \hline 11011 = -5 \end{array}$ <p>(d) <math>(-4) + (-1)</math></p>
$\begin{array}{r} 0101 = 5 \\ +0100 = 4 \\ \hline 1001 = \text{Overflow} \end{array}$ <p>(e) <math>(+5) + (+4)</math></p>	$\begin{array}{r} 1001 = -7 \\ +1010 = -6 \\ \hline 10011 = \text{Overflow} \end{array}$ <p>(f) <math>(-7) + (-6)</math></p>

**Regra do *overflow*:** se dois números são somados e ambos são positivos ou ambos negativos, então o *overflow* ocorre se, e somente se, o resultado tiver o sinal oposto.

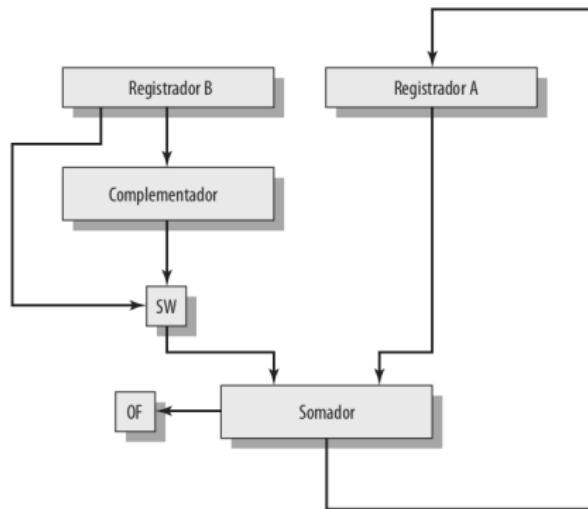
As Figuras 9.3e e f mostram exemplos de *overflow*. Observe que o *overflow* pode ocorrer havendo ou não um *carry*. A subtração é facilmente tratada com a seguinte regra:

**Regra da subtração:** para subtrair um número (subtraendo) de outro (minuendo), apanhe o complemento de dois (negação) do subtraendo e some-o ao minuendo.

# Operações em Complemento de 2: Adição e subtração

Esse diagrama é simplificado. O retorno do resultado e um dos operandos depende do tipo de arquitetura: Pilha, Acumulador, Registrador/Memória ou Registrador/Registrador

**Figura 9.6** Diagrama em blocos do hardware para adição e subtração

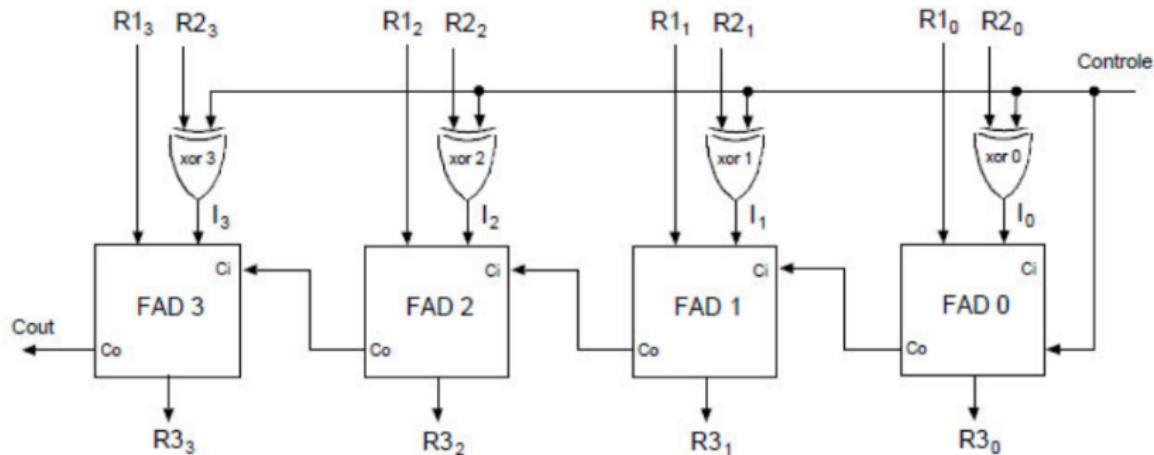


OF = bit de *overflow* (do inglês *overflow bit*)

SW = seletor – multiplexador (seleciona adição ou subtração)

# Operações em Complemento de 2: Adição e subtração

Esse diagrama oferece a soma ou a subtração em um ciclo de clock. Veja que o complemento de um dos números ( $R_2$ ) é feito a partir da linha *controle*. com ela, o complemento de 2 do número é realizado imediatamente antes da operação



Agradeço a fonte<sup>1</sup> pela figura

<sup>1</sup> <http://www.fabiones.com.br/Blog/introducao-a-elettronica-digital.php>

# Multiplicação: Inteiros sem sinal

Inicialmente, analisaremos uma solução para multiplicação de inteiros no formato BCD sem sinal. Com essa base, a aplicaremos aos números em Complemento de 2

**Figura 9.7** Multiplicação de inteiros binários sem sinal

$$\begin{array}{r} 1011 \\ \times 1101 \\ \hline 1011 \\ 0000 \\ 1011 \\ 1011 \\ \hline 10001111 \end{array}$$

Multiplicando (11)  
Multiplicador (13)

Produtos parciais

Produto (143)

Observações importantes:

- A multiplicação gera resultados intermediários
- É composta de várias somas
- Exige um circuito para deslocamento

Como melhorar o desempenho?

- Fazer as somas intermediárias e armazenar apenas o resultado temporário
- Quando o fator a ser multiplicado for 0 (zero), fazer apenas o deslocamento

# Multiplicação: Inteiros sem sinal

Inicialmente, analisaremos uma solução para multiplicação de inteiros no formato BCD sem sinal. Com essa base, a aplicaremos aos números em Complemento de 2

**Figura 9.7** Multiplicação de inteiros binários sem sinal

$$\begin{array}{r} 1011 \\ \times 1101 \\ \hline 1011 \\ 0000 \\ 1011 \\ 1011 \\ \hline 10001111 \end{array}$$

Multiplicando (11)  
Multiplicador (13)

Produtos parciais

Produto (143)

Observações importantes:

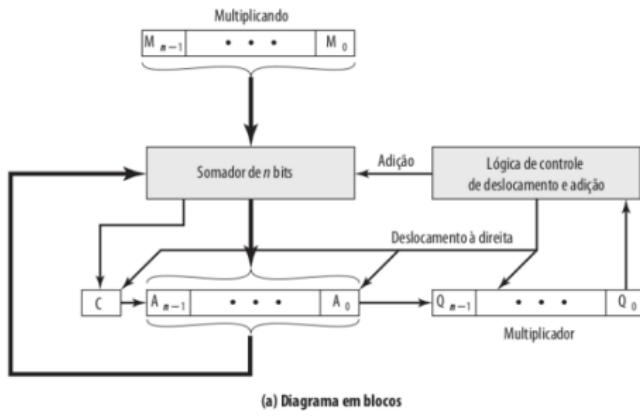
- A multiplicação gera resultados intermediários
- É composta de várias somas
- Exige um circuito para deslocamento

Como melhorar o desempenho?

- Fazer as somas intermediárias e armazenar apenas o resultado temporário
- Quando o fator a ser multiplicado for 0 (zero), fazer apenas o deslocamento

# Multiplicação: Inteiros sem sinal

**Figura 9.8** Implementação de hardware da multiplicação binária sem sinal

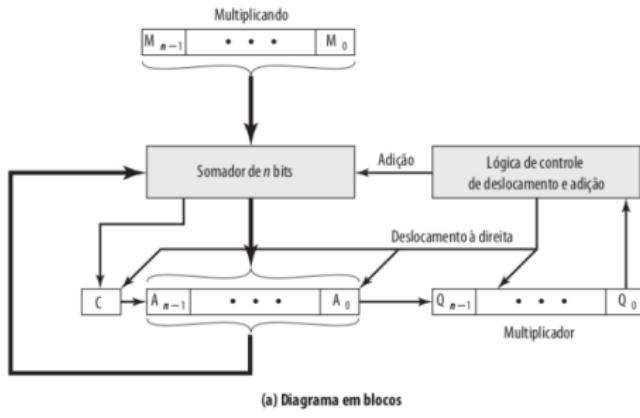


C	A	Q	M	Valores iniciais
0	0000	1101	1011	Valores iniciais

(b) Exemplo da Figura 9.7 (produto em  $A, Q$ )

# Multiplicação: Inteiros sem sinal

**Figura 9.8** Implementação de hardware da multiplicação binária sem sinal

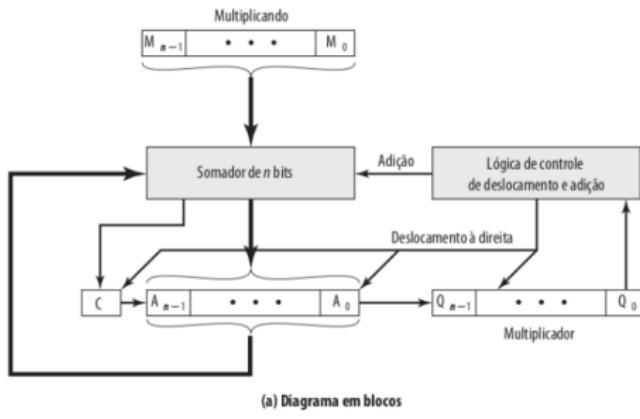


C	A	Q	M		
0	0000	1101	1011	Valores	iniciais
0	1011	1101	1011	Adição	Primeiro
0	0101	1110	1011	Desl.	ciclo

(b) Exemplo da Figura 9.7 (produto em A, Q)

# Multiplicação: Inteiros sem sinal

**Figura 9.8** Implementação de hardware da multiplicação binária sem sinal



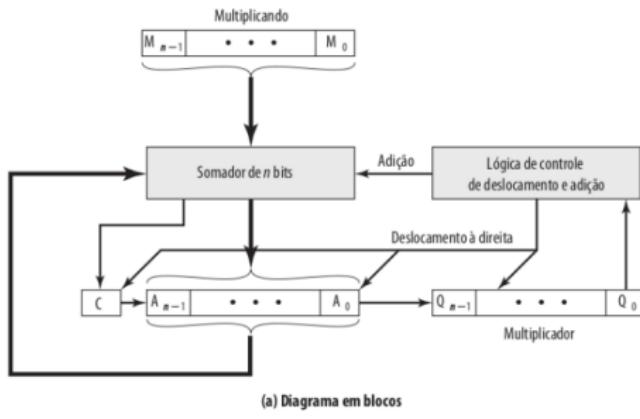
(a) Diagrama em blocos

C	A	Q	M	
0	0000	1101	1011	Valores iniciais
0	1011	1101	1011	Adição } Primeiro
0	0101	1110	1011	Desl. } ciclo
0	0010	1111	1011	Desl. } Segundo
				ciclo

(b) Exemplo da Figura 9.7 (produto em A, Q)

# Multiplicação: Inteiros sem sinal

**Figura 9.8** Implementação de hardware da multiplicação binária sem sinal

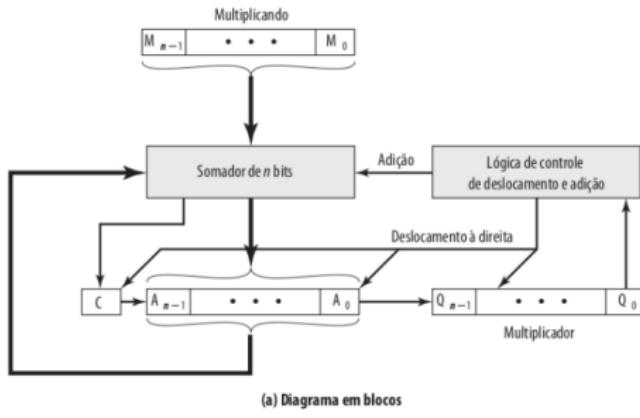


C	A	Q	M	
0	0000	1101	1011	Valores iniciais
0	1011	1101	1011	Adição } Primeiro
0	0101	1110	1011	Desl. } ciclo
0	0010	1111	1011	Desl. } Segundo
0	1101	1111	1011	Adição } Terceiro
0	0110	1111	1011	Desl. } ciclo

(b) Exemplo da Figura 9.7 (produto em  $A, Q$ )

# Multiplicação: Inteiros sem sinal

**Figura 9.8** Implementação de hardware da multiplicação binária sem sinal

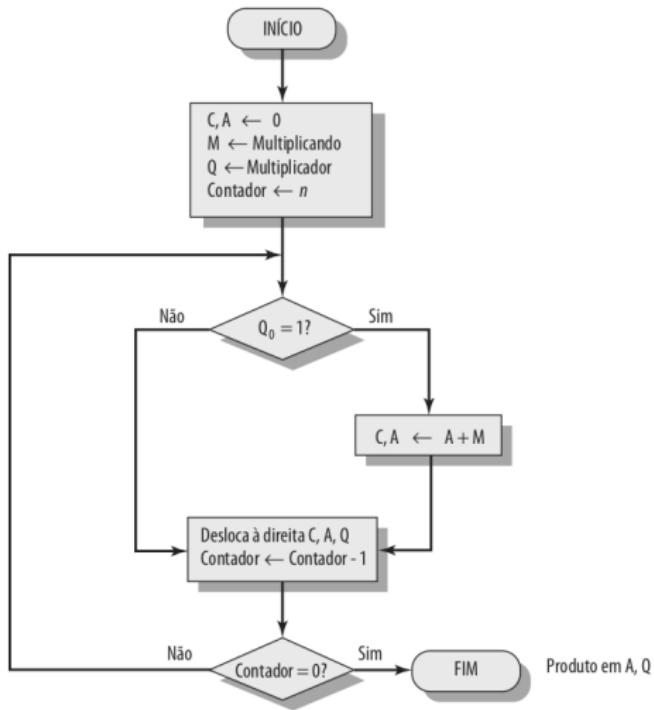


C	A	Q	M		
0	0000	1101	1011	Valores	iniciais
0	1011	1101	1011	Adição	Primeiro
0	0101	1110	1011	Desl.	ciclo
0	0010	1111	1011	Desl.	Segundo
0	1101	1111	1011	Adição	Terceiro
0	0110	1111	1011	Desl.	ciclo
1	0001	1111	1011	Adição	Quarto
0	1000	1111	1011	Desl.	ciclo

(b) Exemplo da Figura 9.7 (produto em A, Q)

# Multiplicação: Inteiros sem sinal

**Figura 9.9** Fluxograma para a multiplicação binária sem sinal



# Multiplicação: Inteiros sem sinal

## Multiplicação em cascata

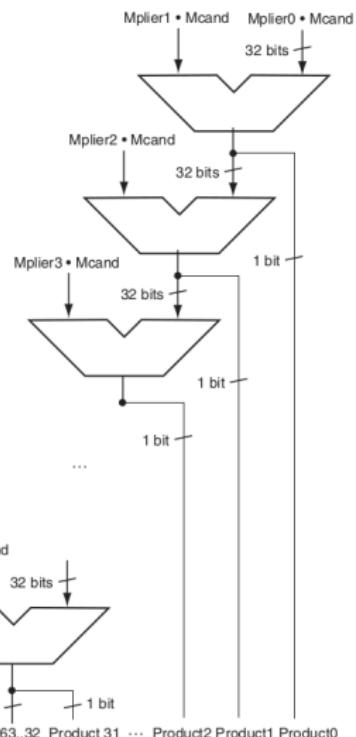
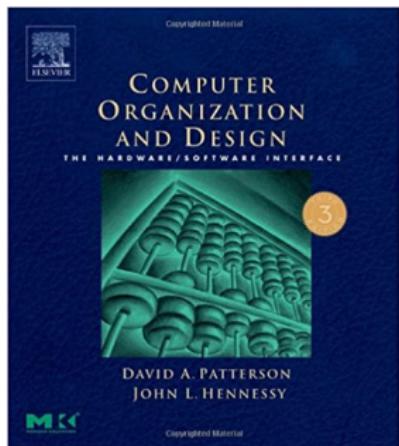
Não entraremos em detalhes

Caso o leitor queira se inteirar:

Material Complementar:

Capítulo 3, Página 176

3<sup>a</sup> Ed. (Inglês)



**FIGURE 3.9 Fast multiplication hardware.** Rather than use a single 32-bit adder 32 times, this hardware "unrolls the loop" to use 32 adders. Each adder produces a 32-bit sum and a carry out. The least significant bit is a bit of the product, and the carry out and the upper 31 bits of the sum are passed along to the next adder.

# Multiplicação: Inteiros **com** sinal

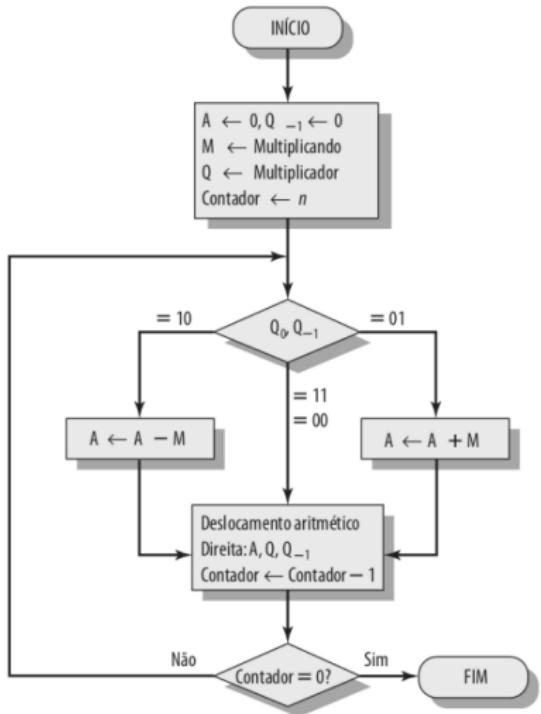
Surge, para os números com sinal, uma estratégia trivial:

- Converter os números em seus respectivos módulos
- Executar a multiplicação sem sinal
- Ajustar os sinais ao fim da multiplicação

Contudo, existe uma opção para a multiplicação de inteiros com sinal:

- O *Algoritmo de Booth*

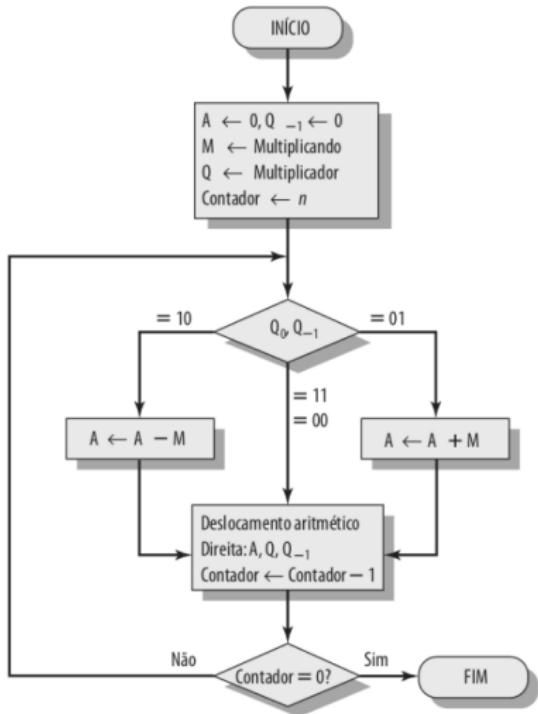
# Multiplicação: Inteiros com sinal - Algoritmo de Booth



A	Q	Q <sub>-1</sub>	M	Valores iniciais
0000	0011	0	0111	

Multiplicação de  $7 \times 3$

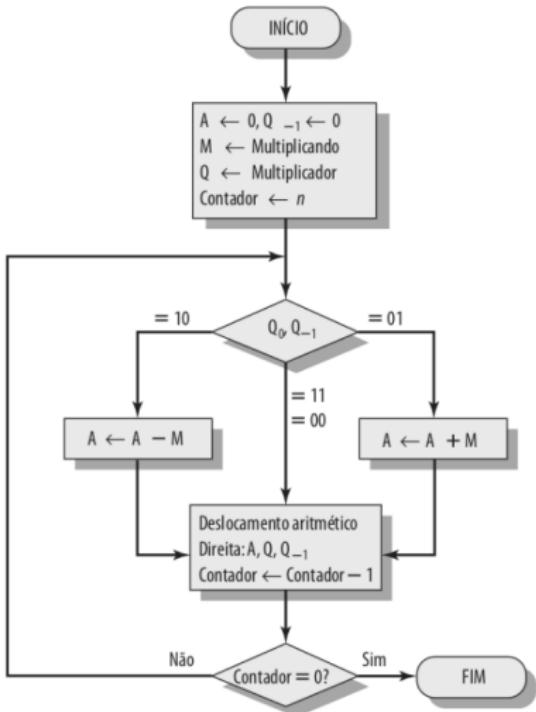
# Multiplicação: Inteiros com sinal - Algoritmo de Booth



A	Q	$Q_{-1}$	M	Valores iniciais
0000	0011	0	0111	
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo

Multiplicação de  $7 \times 3$

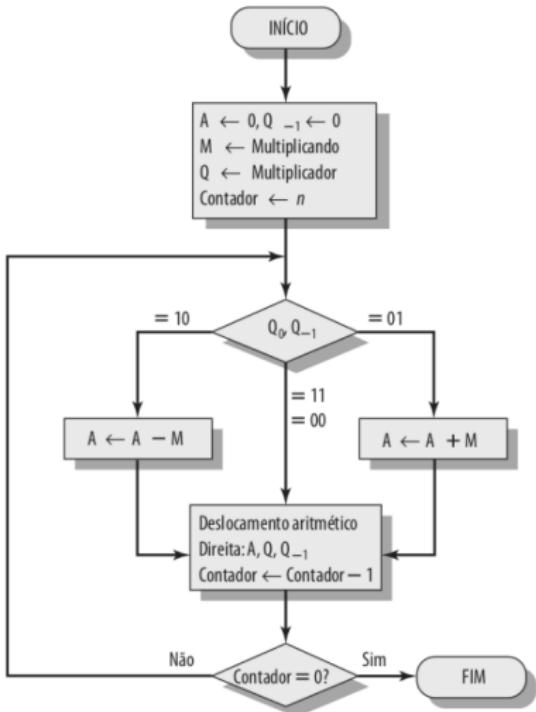
# Multiplicação: Inteiros com sinal - Algoritmo de Booth



A	Q	$Q_{-1}$	M	Valores iniciais
0000	0011	0	0111	
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo
1110	0100	1	0111	Deslocamento } Segundo
				ciclo

Multiplicação de  $7 \times 3$

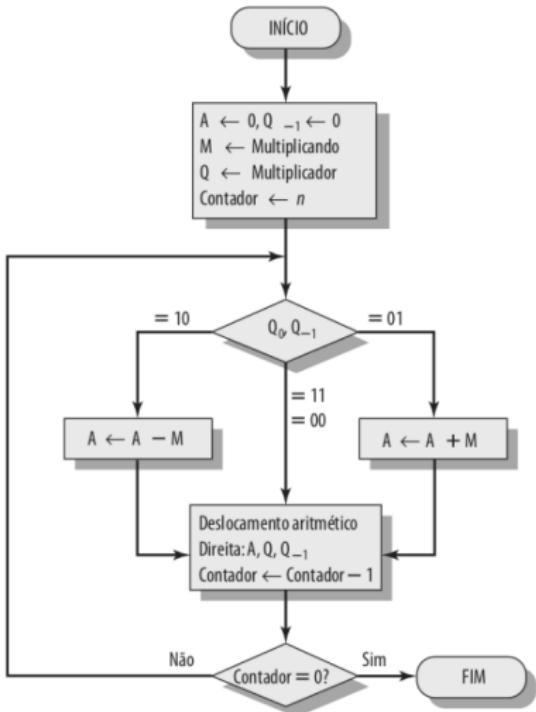
# Multiplicação: Inteiros com sinal - Algoritmo de Booth



A	Q	$Q_{-1}$	M	Valores iniciais
0000	0011	0	0111	
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo
1110	0100	1	0111	Deslocamento } Segundo
0101	0100	1	0111	Deslocamento } Terceiro
0010	1010	0	0111	Deslocamento } ciclo

Multiplicação de  $7 \times 3$

# Multiplicação: Inteiros com sinal - Algoritmo de Booth



A	Q	$Q_{-1}$	M	Valores iniciais
0000	0011	0	0111	
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo
1110	0100	1	0111	Deslocamento } Segundo ciclo
0101	0100	1	0111	$A \leftarrow A + M$ } Terceiro
0010	1010	0	0111	Deslocamento } ciclo
0001	0101	0	0111	Deslocamento } Quarto ciclo

Multiplicação de  $7 \times 3$

# Multiplicação: Inteiros com sinal - Algoritmo de Booth

## Exemplos de execução

**Figura 9.14** Exemplos usando o algoritmo de Booth

$\begin{array}{r} 0111 \\ \times 0011 \\ \hline 11111001 \end{array}$	$\begin{array}{r} 0111 \\ \times 1101 \\ \hline 11111001 \end{array}$
(0)	(0)
1-0	1-0
1-1	0-1
0-1	1-0
$\underline{\underline{000111}}$	$\underline{\underline{111001}}$
(21)	(-21)

$$(a) (7) \times (3) = (21)$$

$$(b) (7) \times (-3) = (-21)$$

$\begin{array}{r} 1001 \\ \times 0011 \\ \hline 00000111 \end{array}$	$\begin{array}{r} 1001 \\ \times 1101 \\ \hline 00000111 \end{array}$
(0)	(0)
1-0	1-0
1-1	0-1
0-1	1-0
$\underline{\underline{111001}}$	$\underline{\underline{000111}}$
(-21)	(21)

$$(c) (-7) \times (3) = (-21)$$

$$(d) (-7) \times (-3) = (21)$$

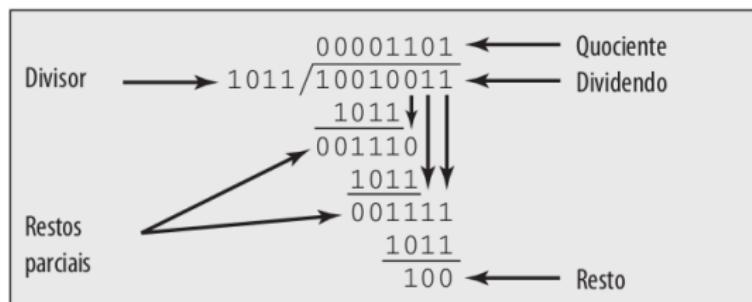
- Observe que estes exemplos são de demonstração do algoritmo.
- Em um computador, a quantidade de *bits* para representar os números é definida.
- Neste caso, seriam necessários números de pelo menos 6 *bits*.

# Divisão: Inteiros sem sinal

A divisão é mais complicada que a multiplicação.

Ela exige cálculos intermediários que são desfeitos durante a execução

**Figura 9.15** Exemplo de divisão de inteiros binários sem sinal

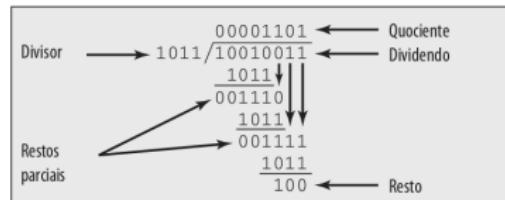
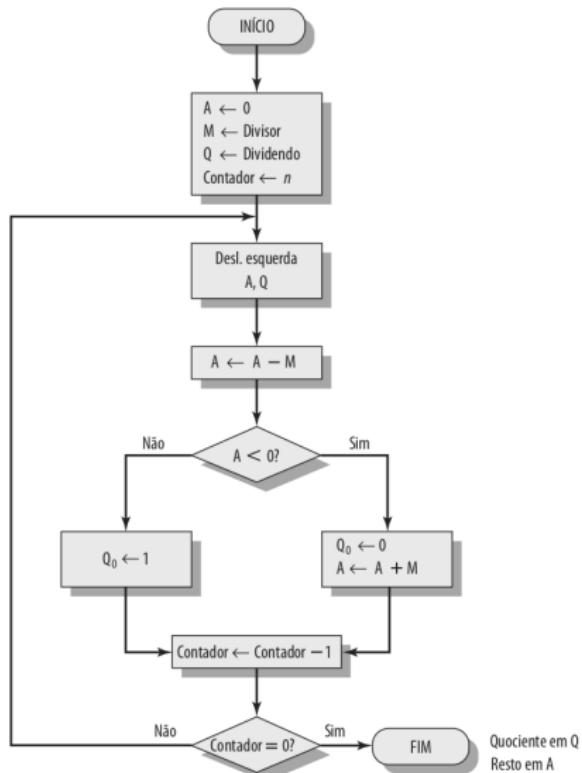


Contudo, a divisão em números binários é mais simples que a divisão em decimal.

Você sabe explicar o por quê?

# Divisão: Inteiros sem sinal

Fluxograma para divisão binária sem sinal



# Divisão: Inteiros com sinal

Vejamos uma alternativa para a divisão inteira com sinais

Para tratar com números negativos, sabemos que o resto é definido por

$$D = Q \times V + R$$

Considere os seguintes exemplos de divisão de inteiros com todas as combinações possíveis de sinais de  $D$  e  $V$ :

$$D = 7 \quad V = 3 \quad \Rightarrow \quad Q = 2 \quad R = 1$$

$$D = 7 \quad V = -3 \quad \Rightarrow \quad Q = -2 \quad R = 1$$

$$D = -7 \quad V = 3 \quad \Rightarrow \quad Q = -2 \quad R = -1$$

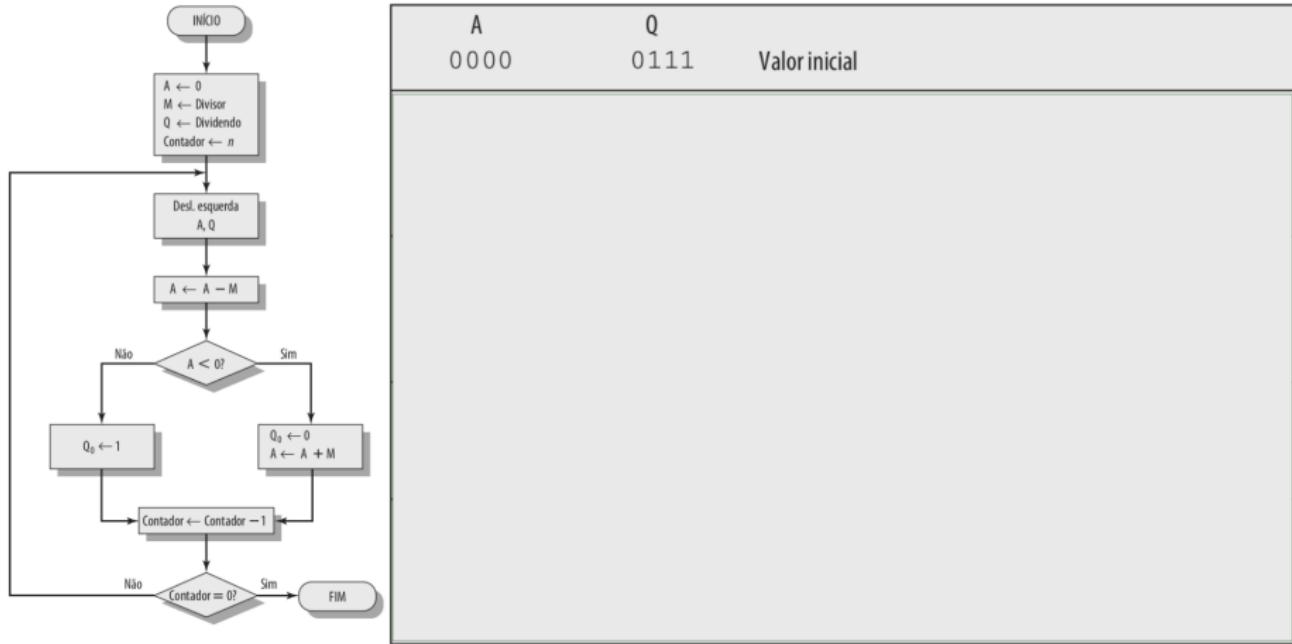
$$D = -7 \quad V = -3 \quad \Rightarrow \quad Q = 2 \quad R = -1$$

Considerando a análise acima, podemos usar um algoritmo de divisão de inteiros sem sinal para números com sinal

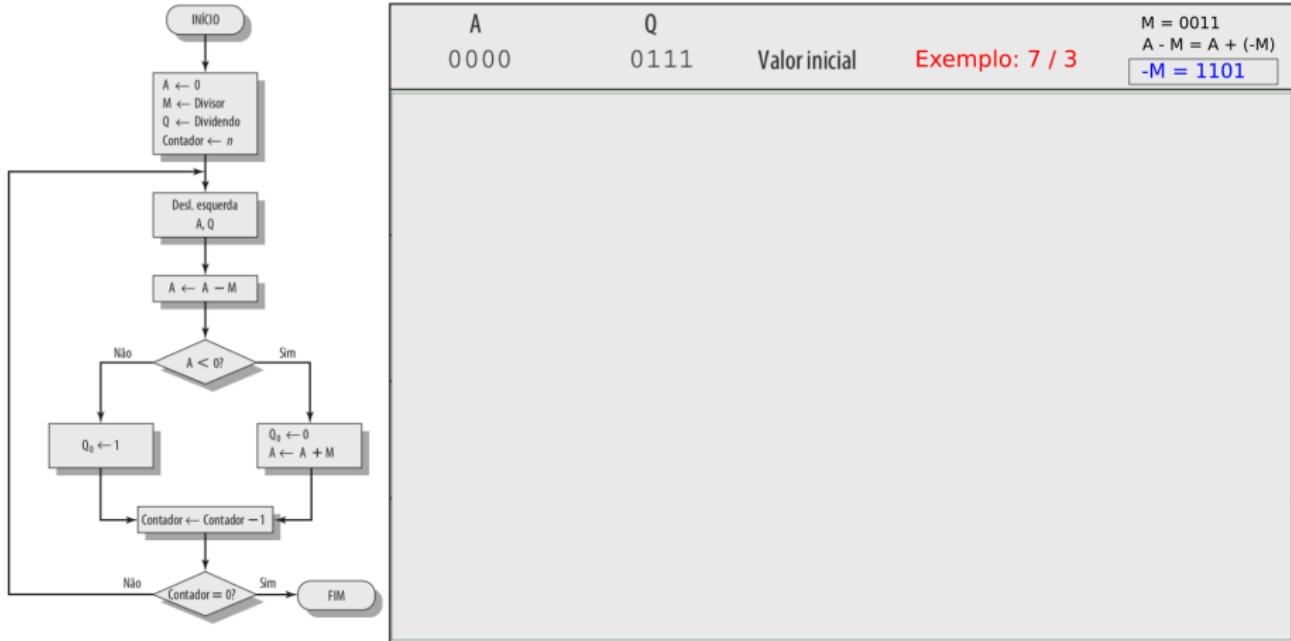
Para isso, devemos (por exemplo em Complemento de 2)

- Avaliar os sinais dos números *divisor* e *dividendo*
- Converter ambos para sua forma positiva
- Aplicar o algoritmo de divisão
- Ao quociente e ao resto, converter os sinais

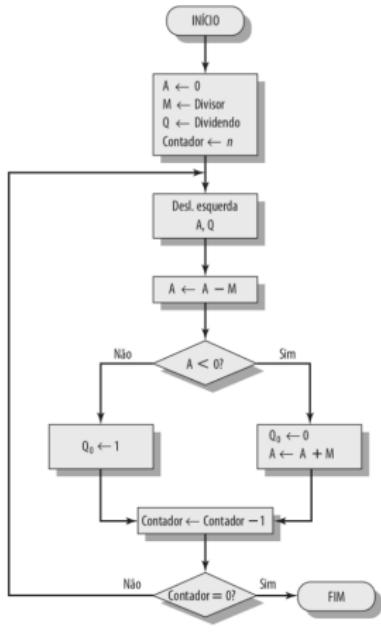
# Divisão: Inteiros - Exemplo



# Divisão: Inteiros - Exemplo



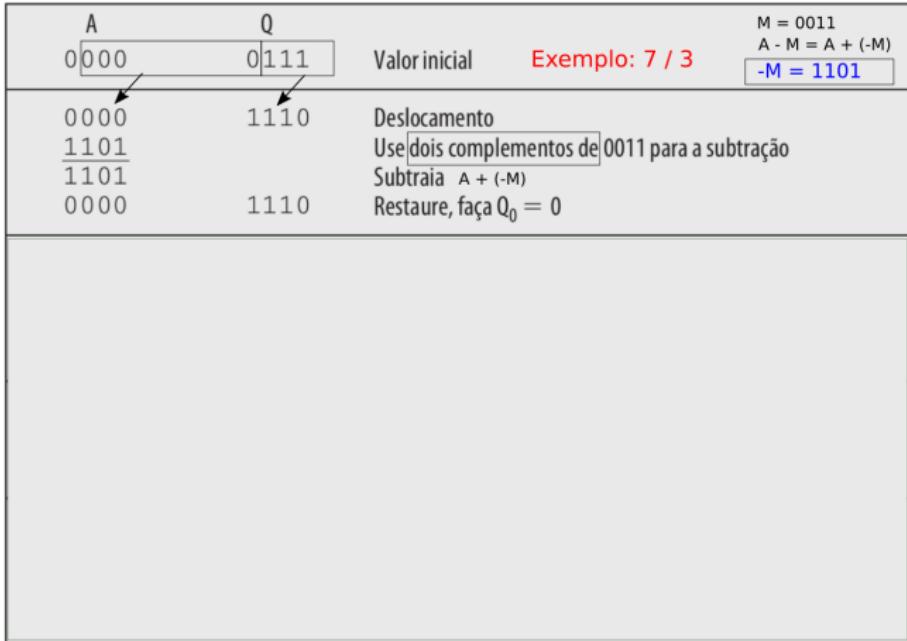
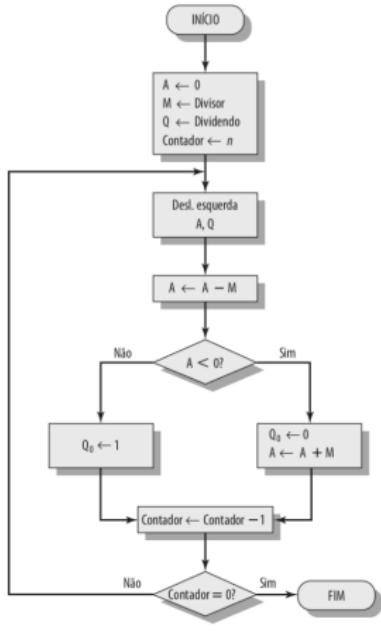
# Divisão: Inteiros - Exemplo



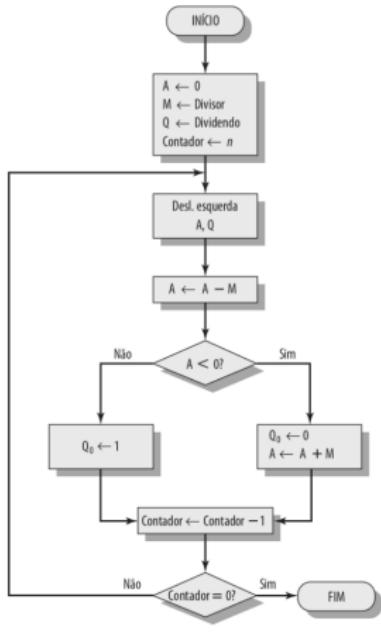
A	Q	Valor inicial	Exemplo: 7 / 3	M = 0011 A - M = A + (-M) -M = 1101
0000	0111			

0000      1110      Deslocamento  
1101      Use dois complementos de 0011 para a subtração  
1101      Subtraia  
0000      1110      Restaure, faça Q<sub>0</sub> = 0

# Divisão: Inteiros - Exemplo

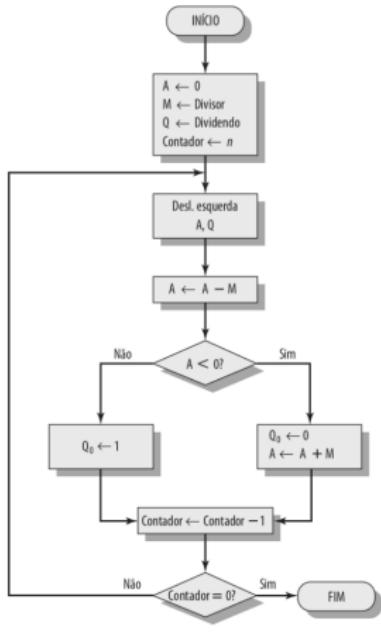


# Divisão: Inteiros - Exemplo



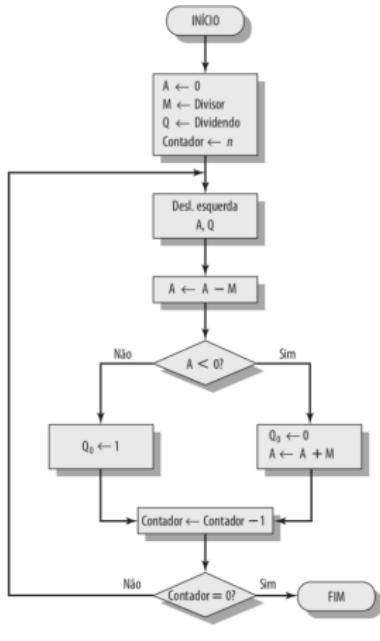
A	Q	Valor inicial	Exemplo: 7 / 3	M = 0011 A - M = A + (-M) -M = 1101
0000	0 111			
0000 1101 1101 0000	1110	Deslocamento Use dois complementos de 0011 para a subtração Subtraia A + (-M) Restaure, faça Q <sub>0</sub> = 0		
0001 1101 1110 0001	1100	Deslocamento Subtraia Restaure, faça Q <sub>0</sub> = 0		

# Divisão: Inteiros - Exemplo



A	Q	Valor inicial	Exemplo: 7 / 3	
0000	0111			M = 0011 A - M = A + (-M) -M = 1101
0000	1110	Deslocamento		
1101		Use dois complementos de 0011 para a subtração		
1101		Subtraia A + (-M)		
0000	1110	Restaure, faça Q <sub>0</sub> = 0		
0001	1100	Deslocamento		
1101				
1110		Subtraia		
0001	1100	Restaure, faça Q <sub>0</sub> = 0		
0011	1000	Deslocamento		
1101				
0000	1001	Subtraia faça Q <sub>0</sub> = 1		

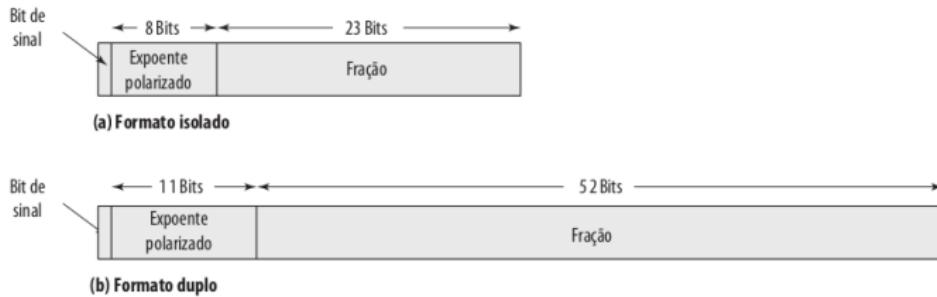
# Divisão: Inteiros - Exemplo



A	Q	Valor inicial	Exemplo: 7 / 3	
0000	0 111			M = 0011 A - M = A + (-M) -M = 1101
0000	1110	Deslocamento		
1101		Use dois complementos de 0011 para a subtração		
1101		Subtraia A + (-M)		
0000	1110	Restaure, faça Q <sub>0</sub> = 0		
0001	1100	Deslocamento		
1101				
1110		Subtraia		
0001	1100	Restaure, faça Q <sub>0</sub> = 0		
0011	1000	Deslocamento		
1101				
0000	1001	Subtraia faça Q <sub>0</sub> = 1		
0001	0010	Deslocamento		
1101				
1110		Subtraia		
0001	0010	Restaure, faça Q <sub>0</sub> = 0		

# Ponto Flutuante: Representação

Figura 9.21 Formatos IEEE 754



Observe que para um número na base decimal, por exemplo, temos:

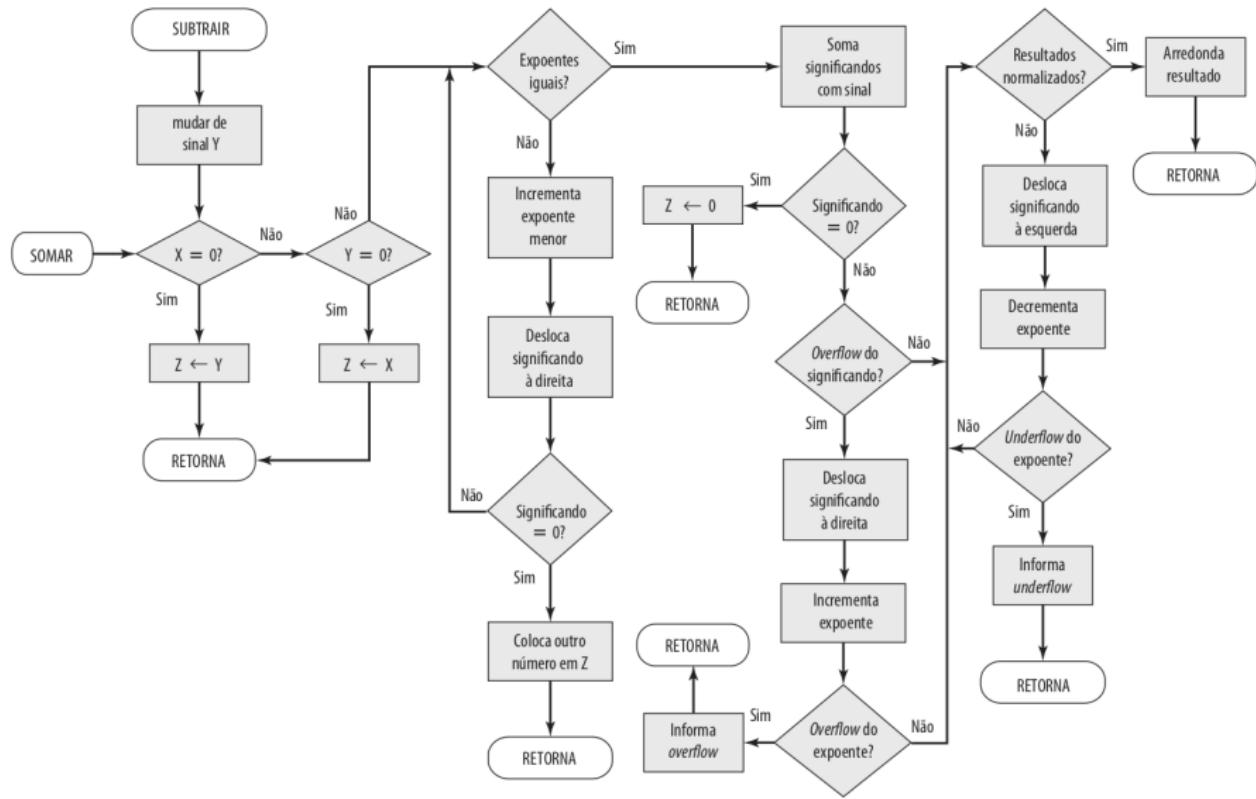
- Multiplicação:  $(A \times 10^y) \times (B \times 10^z) = (A \times B) \times 10^{(y+z)}$
- Divisão:  $(A \times 10^y) / (B \times 10^z) = (A/B) \times 10^{(y-z)}$
- Neste caso,  $A$  e  $B$  são números inteiros, assim como  $y$  e  $z$
- Assim, essas operações de ponto flutuante são reduzidas às operações de inteiros
- E a soma?

# Ponto Flutuante: Adição e Subtração

Aqui, sintetizamos os passos da adição ou subtração para ponto flutuantes

- Verifique zero (a resposta é trivial)
- Alinhe as mantissas (ajustando expoentes)
- Some ou subtraia mantissas
- Normalize resultado

# Ponto Flutuante: Adição e Subtração



# Ponto Flutuante: Multiplicação ou divisão

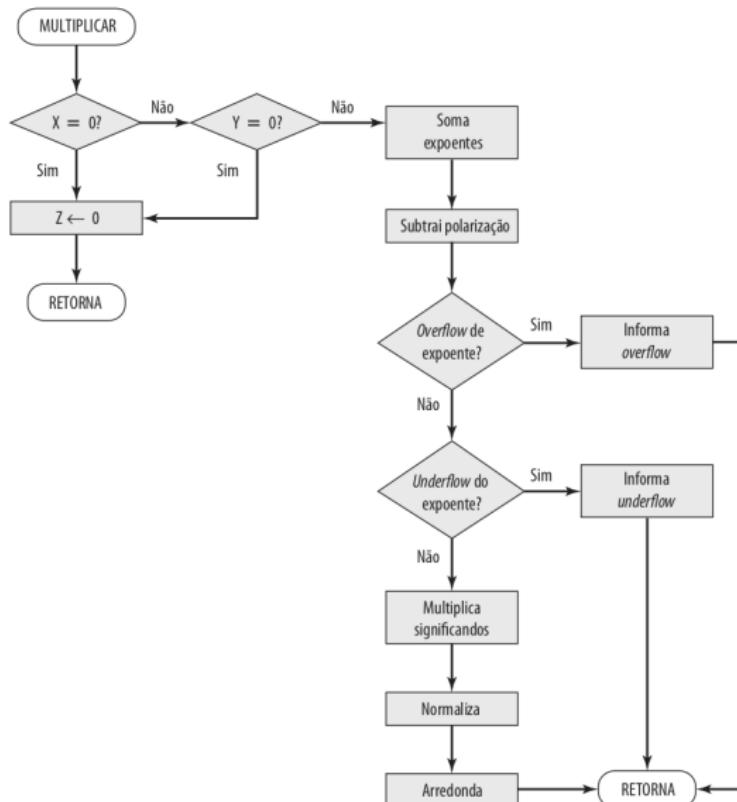
Aqui sintetizamos os passos para a multiplicação ou divisão para ponto flutuantes

- Verifique zero (a resposta é trivial)
- Some ou subtraia expoentes
- Multiplique ou divida as mantissas (observe sinal).
- Normalize resultado
- Arredonde

Observação: os resultados intermediários devem ser em armazenamento de tamanho duplo

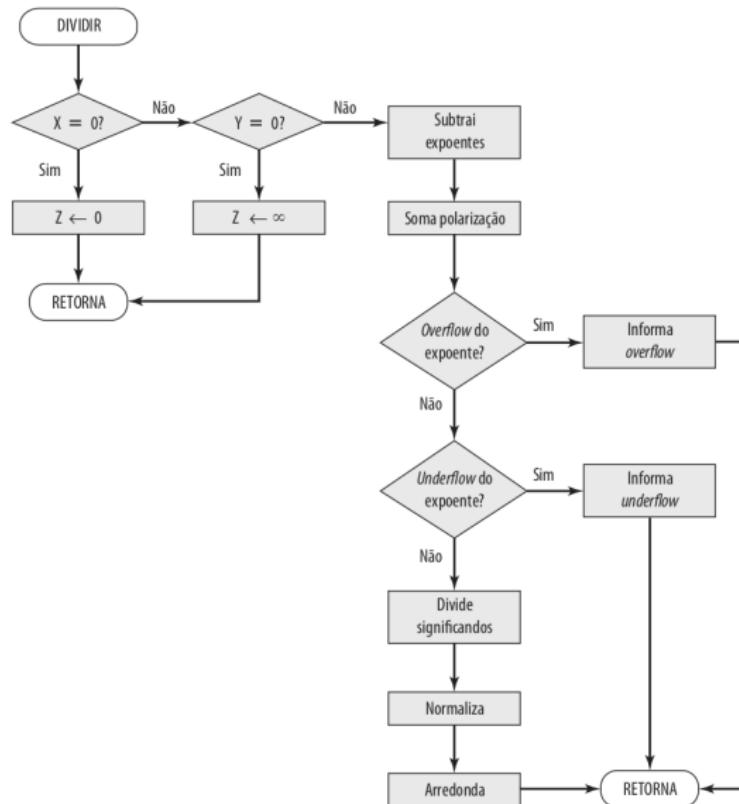
# Ponto Flutuante: Multiplicação

**Figura 9.23** Multiplicação de ponto flutuante ( $Z \leftarrow X \times Y$ )



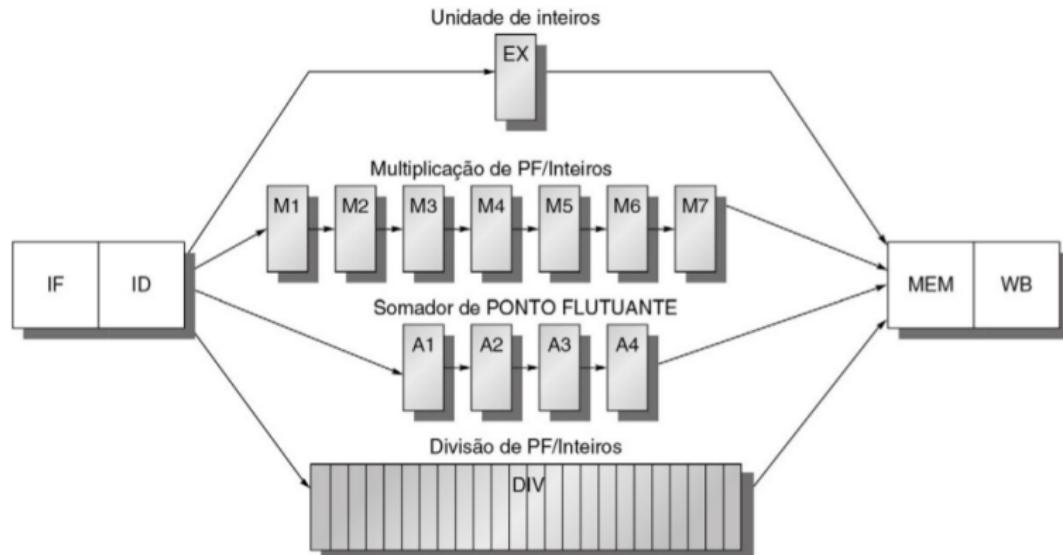
# Ponto Flutuante: Divisão

**Figura 9.24** Divisão de ponto flutuante ( $Z \leftarrow X/Y$ )



# Exemplo de ULA

Mostramos um exemplo de unidades funcionais e a quantidade de ciclos para a execução:



**FIGURA C.35** Um pipeline que admite múltiplas operações de PF pendentes.

O multiplicador e o somador de PF estão totalmente em pipeline e possuem uma profundidade de sete e quatro estágios, respectivamente. O divisor de PF não está em pipeline, mas exige 24 ciclos de clock para concluir. A latência nas instruções entre o despacho de uma operação de PF e o uso do resultado dessa operação sem incorrer em um stall RAW é determinada pelo número de ciclos gastos nos estágios de execução. Por exemplo, a quarta instrução após uma adição de PF pode usar o resultado da adição de PF. Para operações da ALU com inteiros, a profundidade do pipeline de execução é sempre um e a próxima instrução pode usar os resultados.



# Agradecimentos

## Agradecimentos Especiais:

Agradeço a toda a comunidade L<sup>A</sup>T<sub>E</sub>X.  
Em especial a *Till Tantau* pelo *Beamer*.

<https://www.tcs.uni-luebeck.de/mitarbeiter/tantau/>

Desta forma, tornou-se possível a escrita deste material didático.

## Exercícios:

Lista de exercícios divulgada no Moodle

