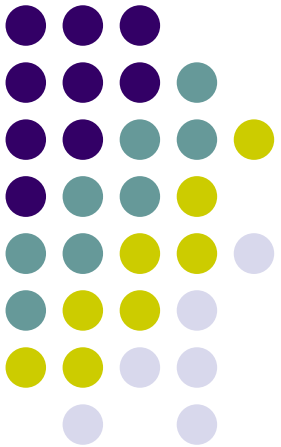
Organização de Computadores

ALU e aritmética computacional

Prof. José Paulo G. de Oliveira
Engenharia da Computação, UPE
jpgo@poli.upe.br

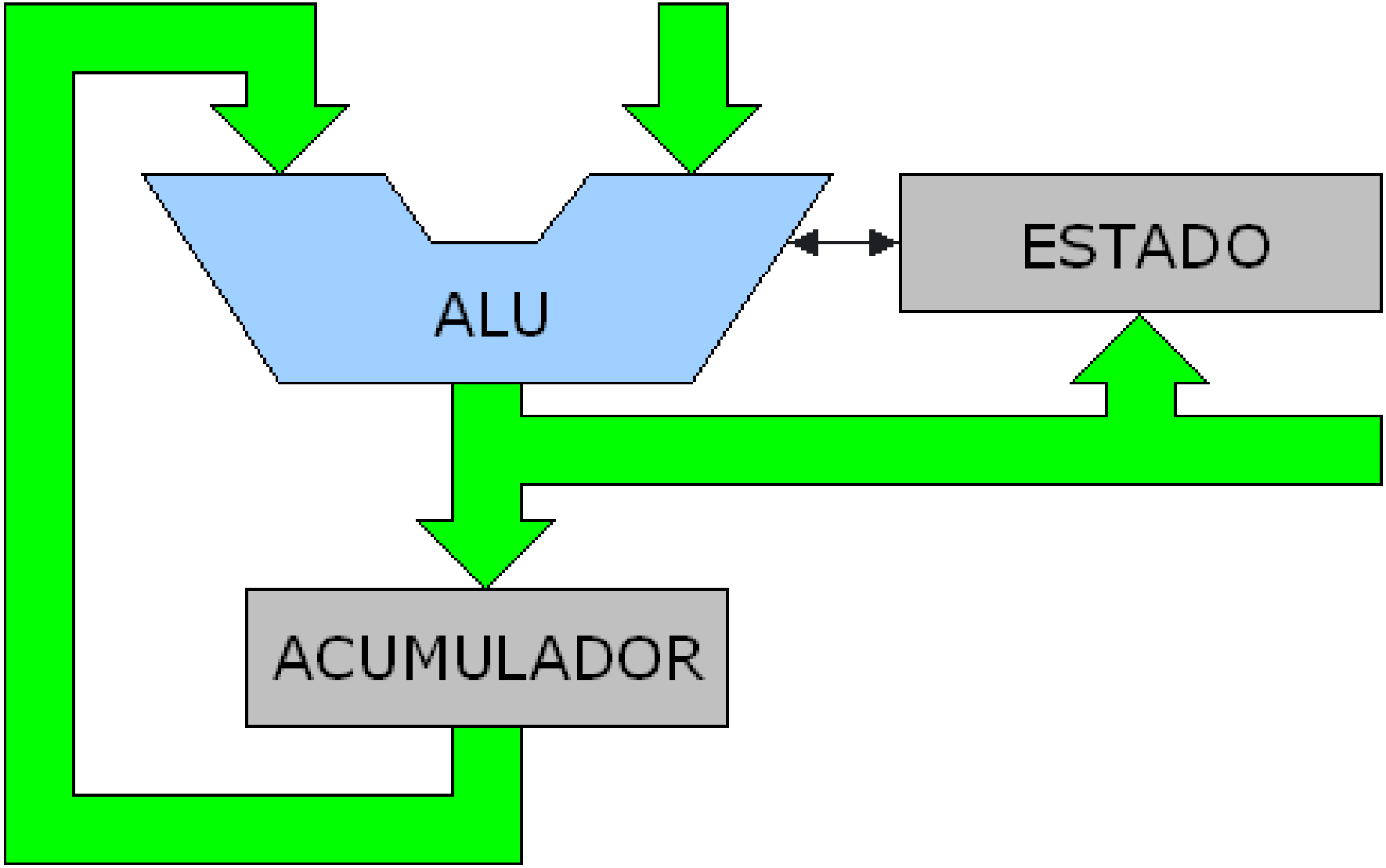


Conteúdo

- ULA (ALU)
- Representação numérica
- Operações básicas
- Algoritmo de Booth
- Números reais e ponto flutuante

Conteúdo

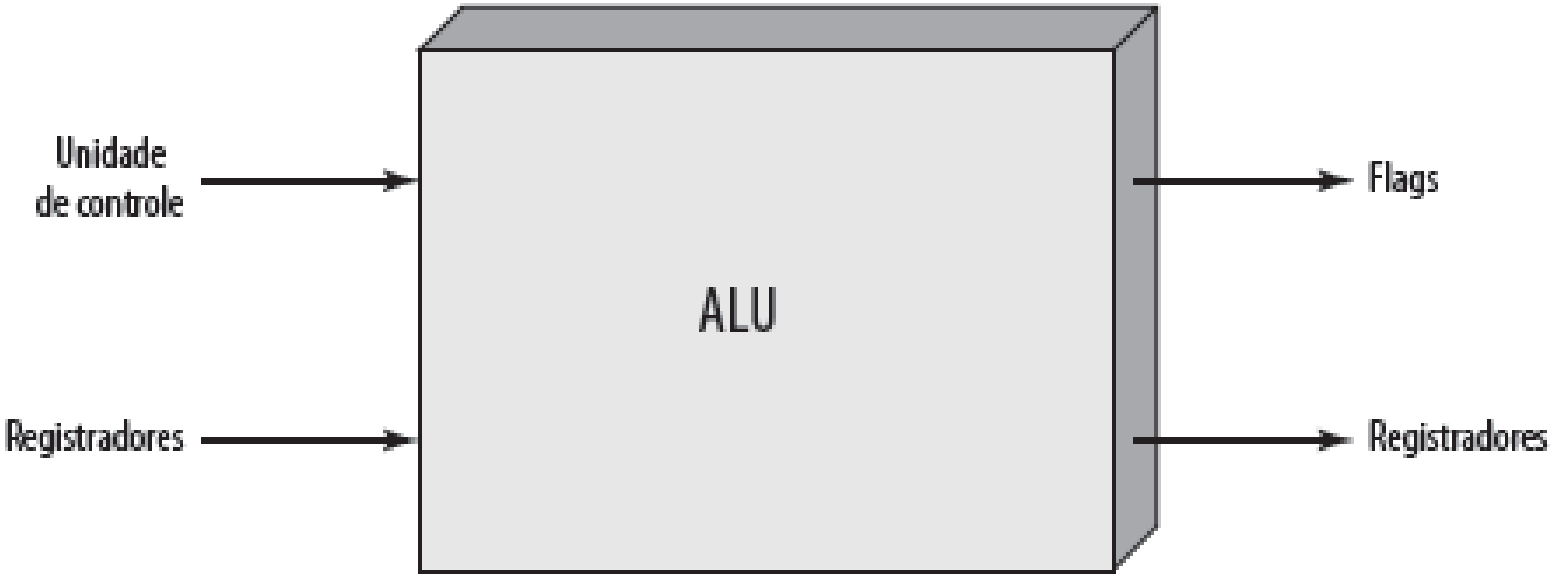
- **ULA (ALU)**
- Representação numérica
- Operações básicas
- Algoritmo de Booth
- Números reais e ponto flutuante


ALU
ESTADO
ACUMULADOR

Unidade de lógica de aritmética

- Faz os cálculos.
- Tudo o mais no computador existe para atender a essa unidade.
- Trata números binários
- Pode tratar de números de ponto flutuante (reais).
  - Pode ser uma FPU separada (coprocessador matemático).
  - Pode estar no chip da CPU separado (486DX +).

Entradas e saídas da ALU



Conteúdo

- ULA (ALU)
- **Representação numérica**
- Operações básicas
- Algoritmo de Booth
- Números reais e ponto flutuante

Representação de operandos numéricos

- Só há 0 & 1 para representar tudo
- Números armazenados em binário
- E.g., 41=00101001
- Sem sinal negativo!
- Sem ponto!

Representação de operandos numéricos

- Só há 0 & 1 para representar tudo
- Números armazenados em binário
- E.g., 41=00101001
- Sem sinal negativo!
- Sem ponto!

- Como solucionar?
  - Sinal-magnitude
  - Complemento a dois
  - ...

Sinal-magnitude

- Bit mais à esquerda é bit de sinal
- 0 significa positivo
- 1 significa negativo
- +18 = 00010010
- -18 = 10010010
- Problemas:
  - Precisa considerar sinal e magnitude na aritmética
  - Duas representações de zero (+0 e -0)

Complemento a dois

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

Complemento a dois – complemento a 2^N

Exemplo (Número de bits N = 3)

Complemento a 2^N de 011 é 101, pois:

1000 – 011 = 101

Benefícios

- Uma representação de zero.
- Aritmética funciona naturalmente (ver mais adiante).
- Negação é muito fácil:
  - 3 = 00000011
  - Complemento Booleano gera 11111100
  - Adicione 1 11111101

Negação especial – caso 1

- 0 = 00000000
- *NOT* bit a bit 11111111
- Some 1 ao LSB +1
- Resultado 1 00000000
- Como o bit de estouro (N+1) não é considerado:
  - 0 = -0

Negação especial – caso 2

- -128 = 10000000
- *NOT* bit a bit 01111111
- Some 1 ao LSB +1
- Resultado 10000000
- Portanto:
- -(-128) = -128
- Monitore **msb** (bit de sinal)
- Ele deve mudar durante a negação

Faixa de valores

- Complemento a 2 com 8 bits:
  - +127 = 01111111 = 2^7 -1
  - -128 = 10000000 = -2^7
- Complemento a 2 com 16 bits:
  - +32767 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

  - Faixa: -2^{n-1} até $+2^{n-1}$ - 1

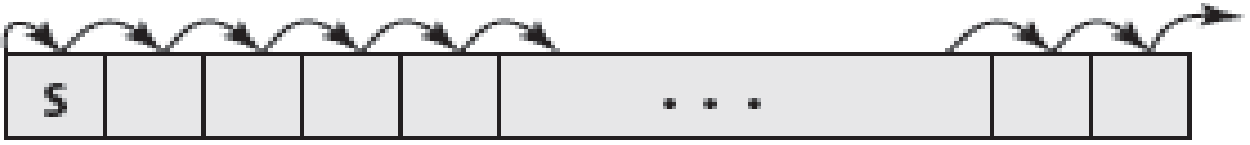
Conversão entre tamanhos

- Números positivos com 0's iniciais
- +18 = 00010010
- +18 = 00000000 00010010

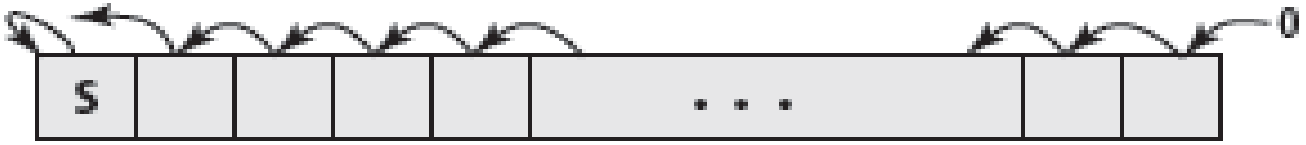
Conversão entre tamanhos

- Números positivos com 0's iniciais
- +18 = 00010010
- +18 = 00000000 00010010
- Números negativos com 1's iniciais
- -18 = 11101110
- -18 = 11111111 11101110
- Ou seja, sequência com **msb** (bit de sinal)

Lembrando...



(c) Deslocamento aritmético à direita



(d) Deslocamento aritmético à esquerda

Conteúdo

- ULA (ALU)
- Representação numérica
- **Operações básicas**
- Algoritmo de Booth
- Números reais e ponto flutuante

Adição e subtração

- Adição binária normal
- Monitore estouro no bit de sinal

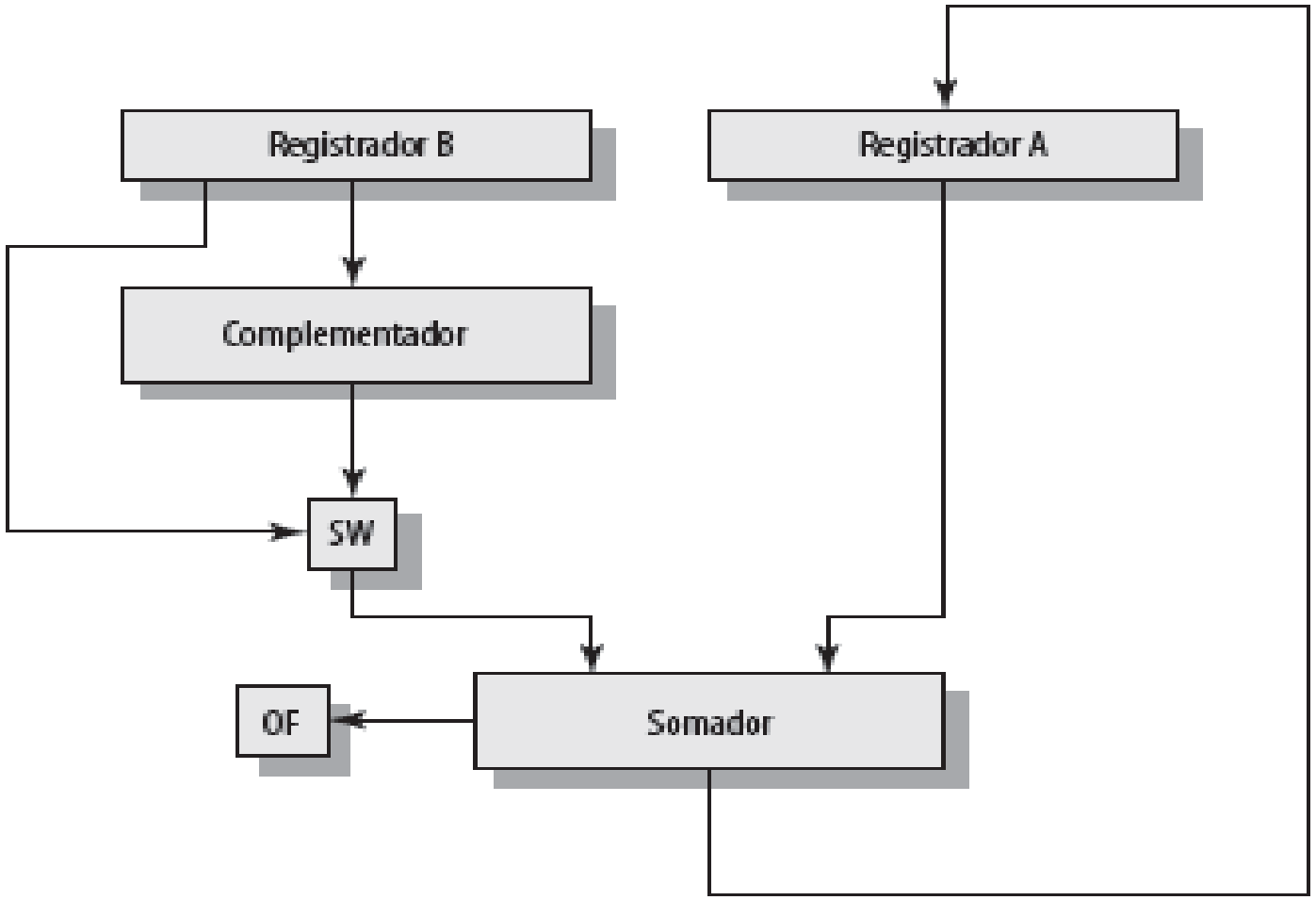
Adição e subtração

- Adição binária normal
- Monitore estouro no bit de sinal

- Subtração: Tome o complemento a dois do subtraendo e some ao minuendo
  - Ou seja, a - b = a + (-b)

- Assim, só precisamos de circuitos de adição e complemento

Hardware para adição e subtração



OF = bit de *overflow* (do inglês *overflow bit*)
SW = seletor – multiplexador (seleciona adição ou subtração)

Multiplicação (sem sinal)

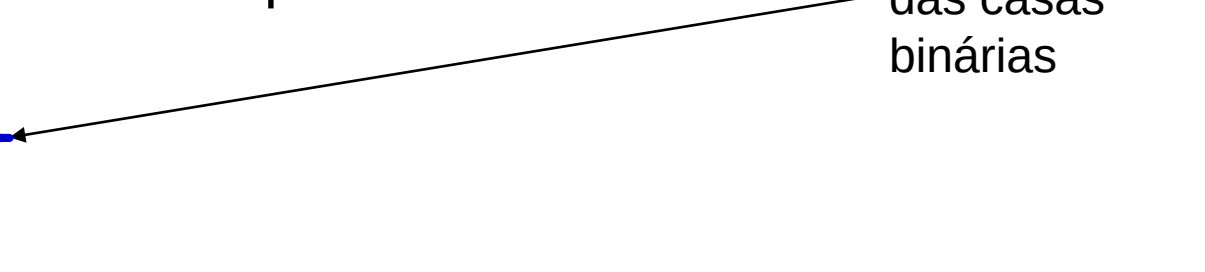
- Complexa
- Calcule produto parcial para cada dígito
- Cuidado com o valor da casa (coluna)
- Some produtos parciais

Exemplo de multiplicação (sem sinal)

- 1011 Multiplicando (11 dec)
- x 1101 Multiplicador (13 dec)
- 1011 Produtos parciais
- 0000
- 1011
- 1011
- 10001111 Produto (143 dec)

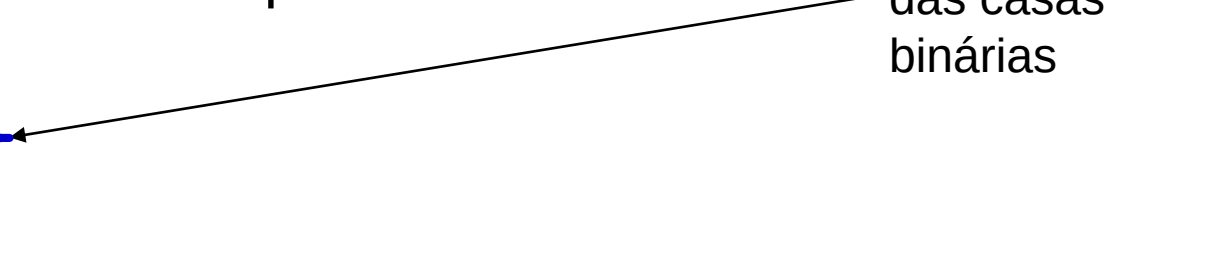
Exemplo de multiplicação (sem sinal)

- 1011 Multiplicando (11 dec)
- x 1101 Multiplicador (13 dec)
- 1011 Produtos parciais
- 0000
- 1011
- 1011
- 10001111 Produto (143 dec)



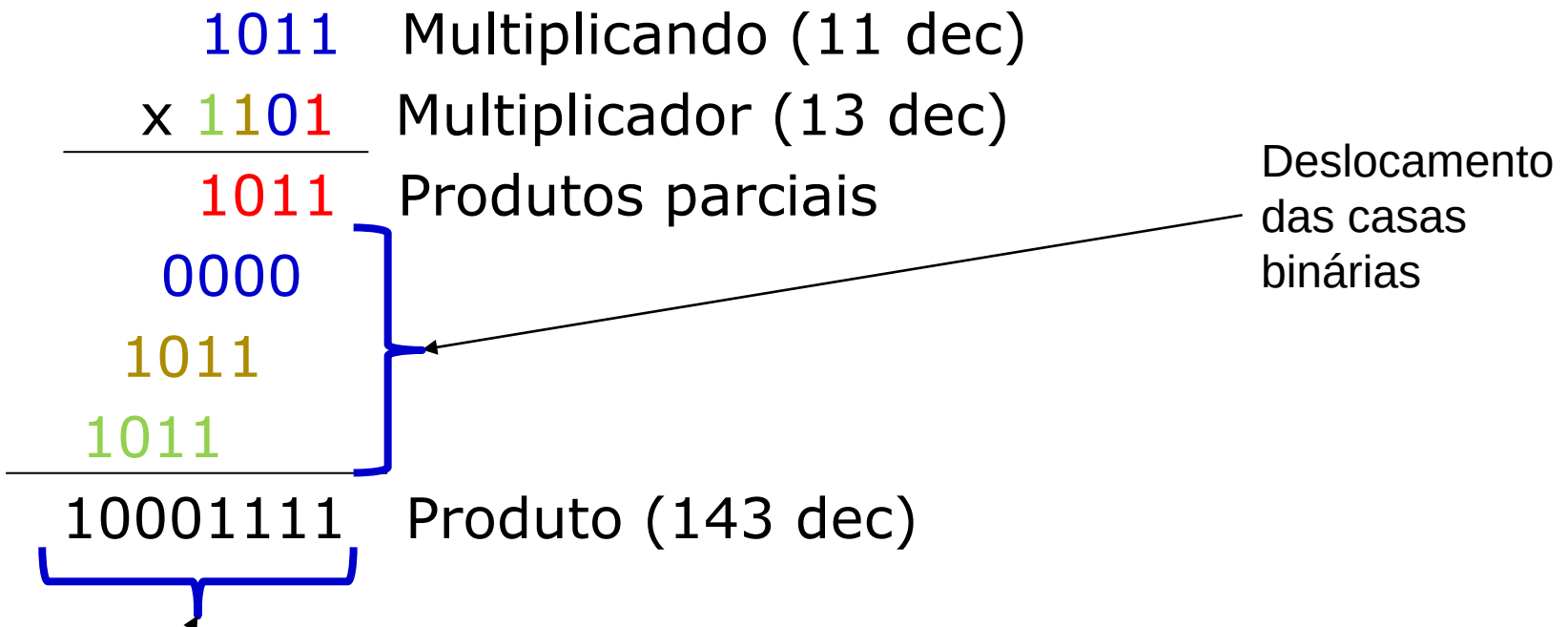
Exemplo de multiplicação (sem sinal)

- 1011 Multiplicando (11 dec)
- x 1101 Multiplicador (13 dec)
- 1011 Produtos parciais
- 0000
- 1011
- 1011
- 10001111 Produto (143 dec)



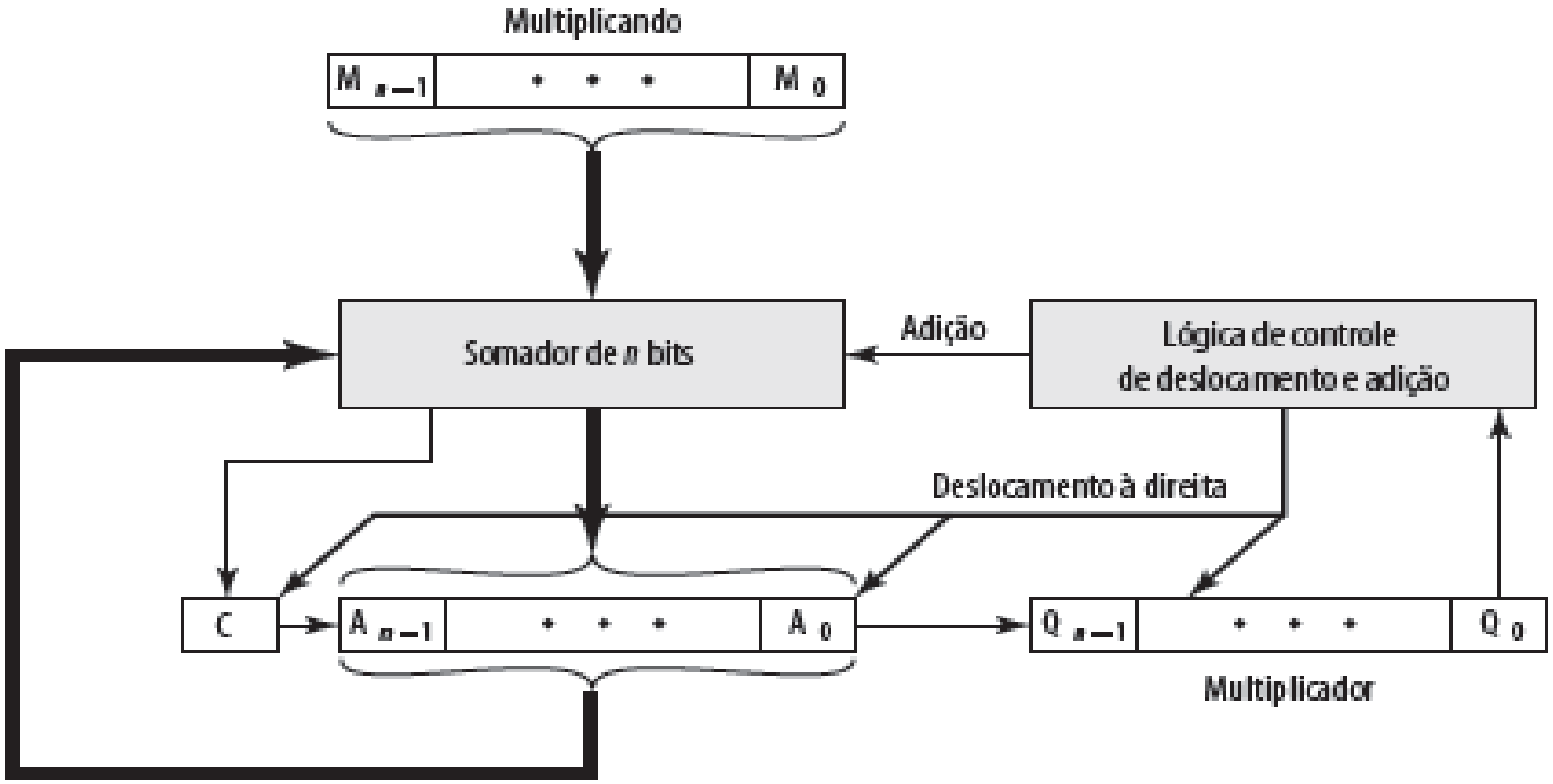
- Nota1: se bit multiplicador for 1, copia o Multiplicando (valor da casa). Caso contrário, copia '0000'

Exemplo de multiplicação (sem sinal)



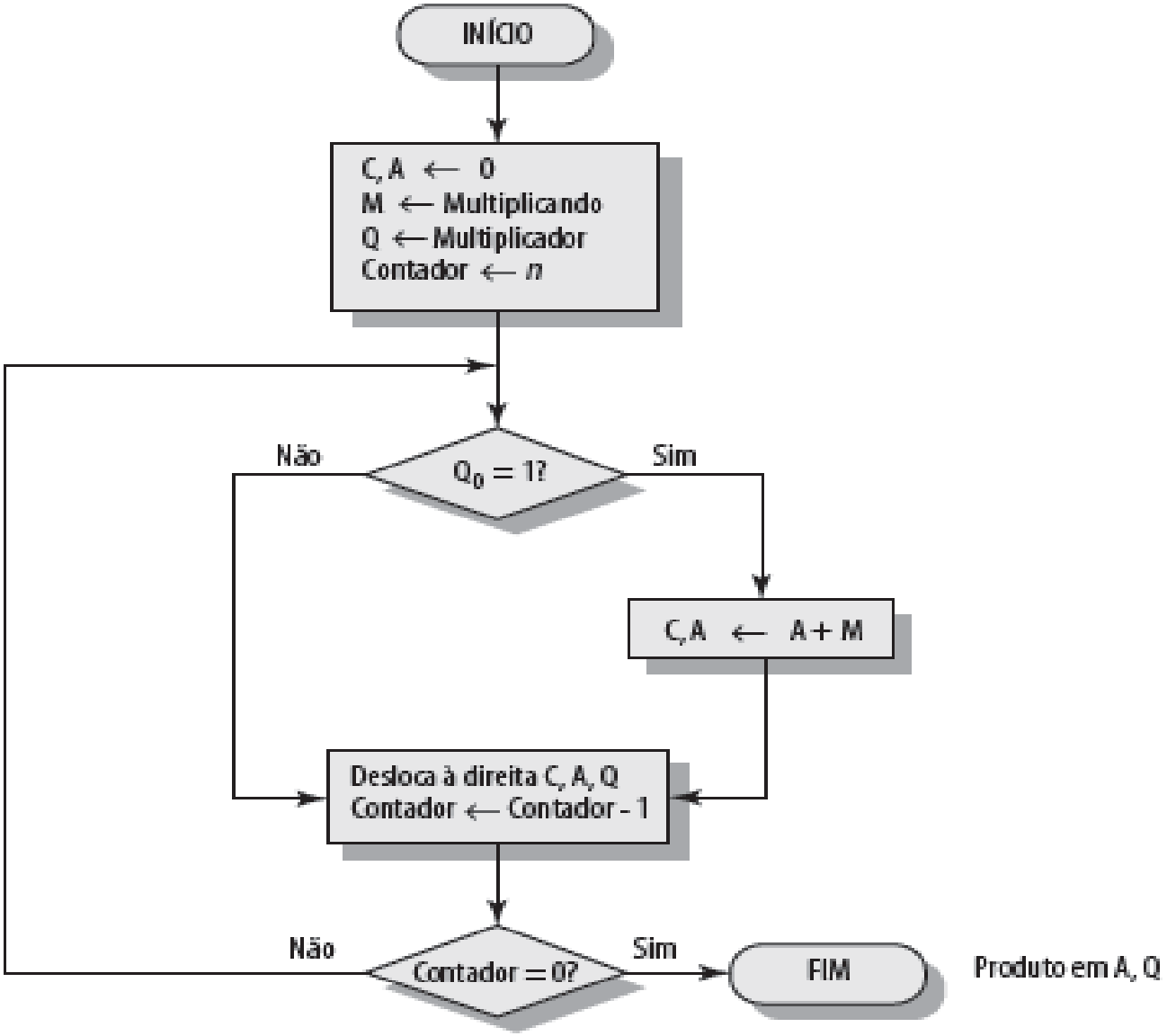
- Nota1: se bit multiplicador for 1, copia o Multiplicando (valor da casa). Caso contrário, copia '0000'
- Nota2: resultado deve ser representado com *tamanho duplo*

HW para multiplicador binário (sem sinal)



Resultado - AQ

Fluxograma para a multiplicação binária sem sinal

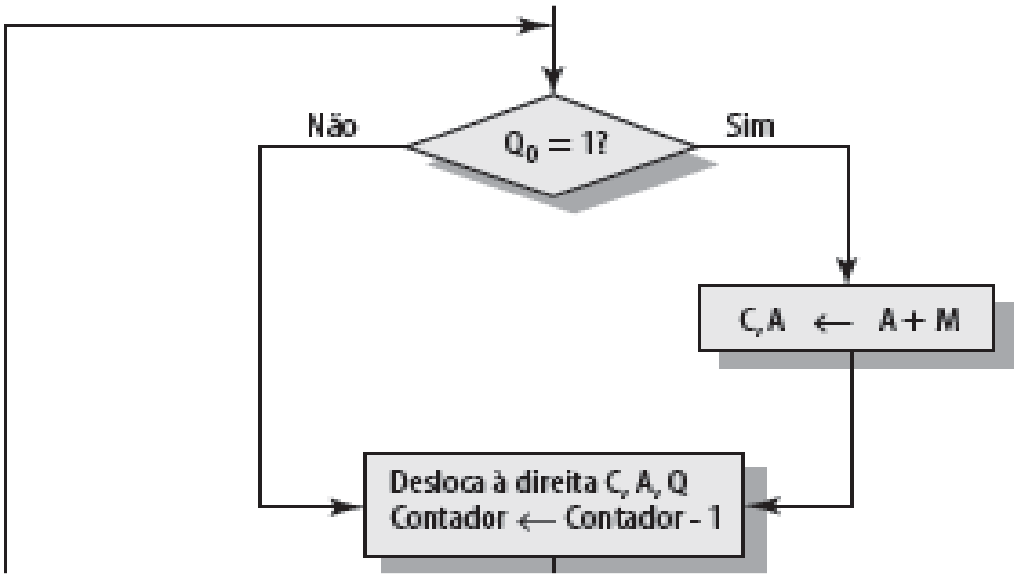


Execução do exemplo: 11 x 13 (n = 4)

C	A	Q	M	
0	0000	1101	1011	Valores iniciais

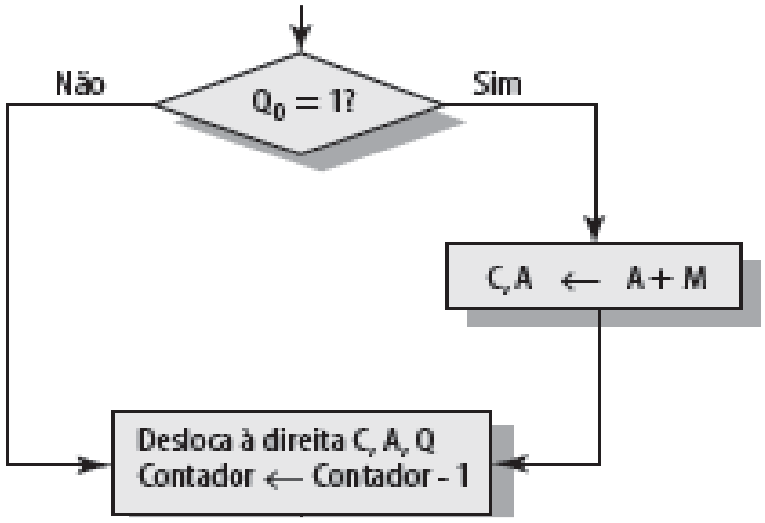
Execução do exemplo: 11 x 13

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



Execução do exemplo: 11 x 13

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



Execução do exemplo: 11 x 13

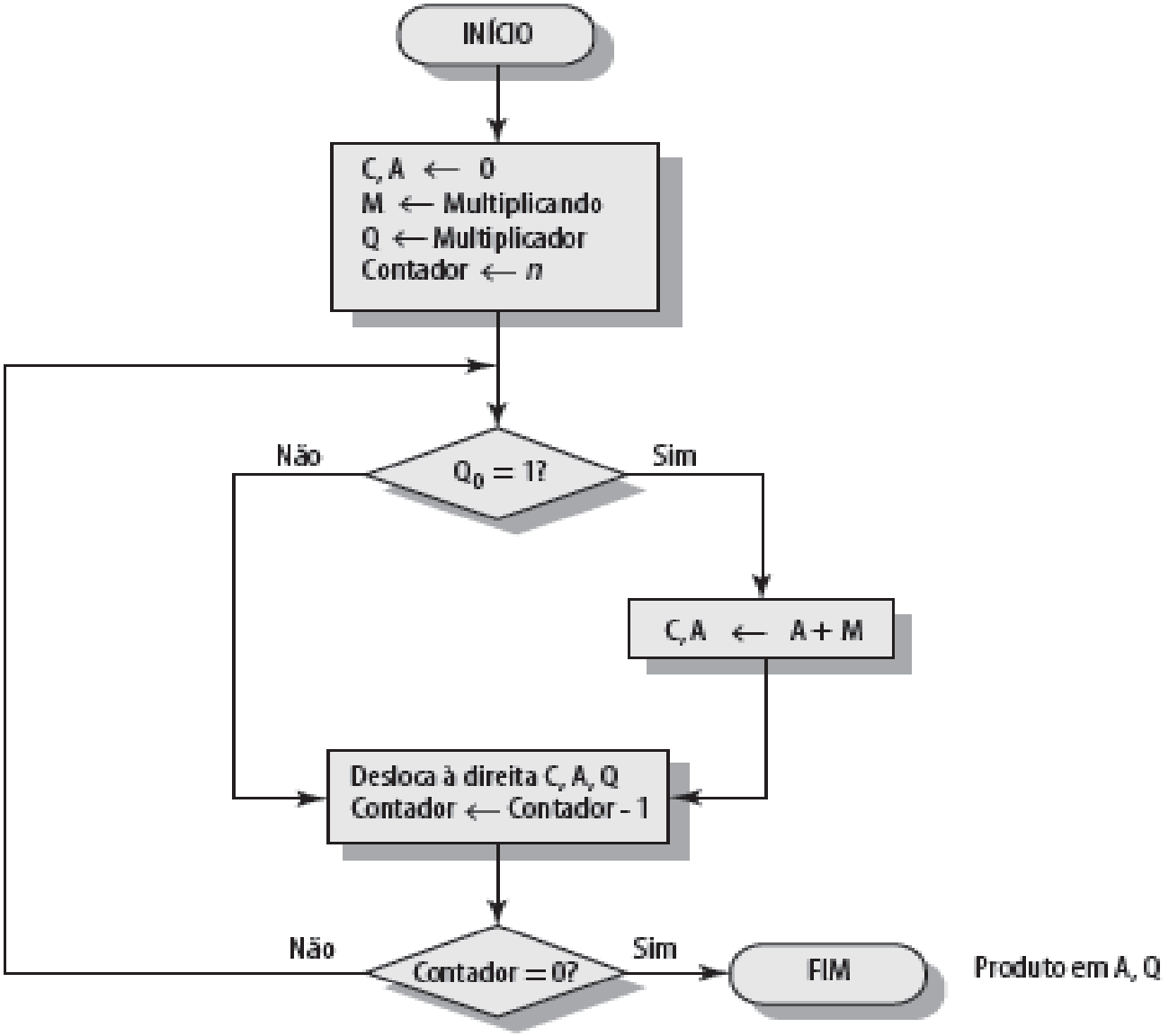
C	A	Q	M		
0	0000	1101	1011	Valores iniciais	
0	1011	1101	1011	Adição	Primeiro ciclo
0	0101	1110	1011	Desl.	
0	0010	1111	1011	Desl.	Segundo ciclo
0	1101	1111	1011	Adição	Terceiro ciclo
0	0110	1111	1011	Desl.	

Execução do exemplo: 11 x 13

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

Cont = 0

Execução do exemplo: 11 x 13



Execução do exemplo: 11 x 13

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

→ = 143

Conteúdo

- ULA (ALU)
- Representação numérica
- Operações básica
- **Algoritmo de Booth**
- Números reais e ponto flutuante

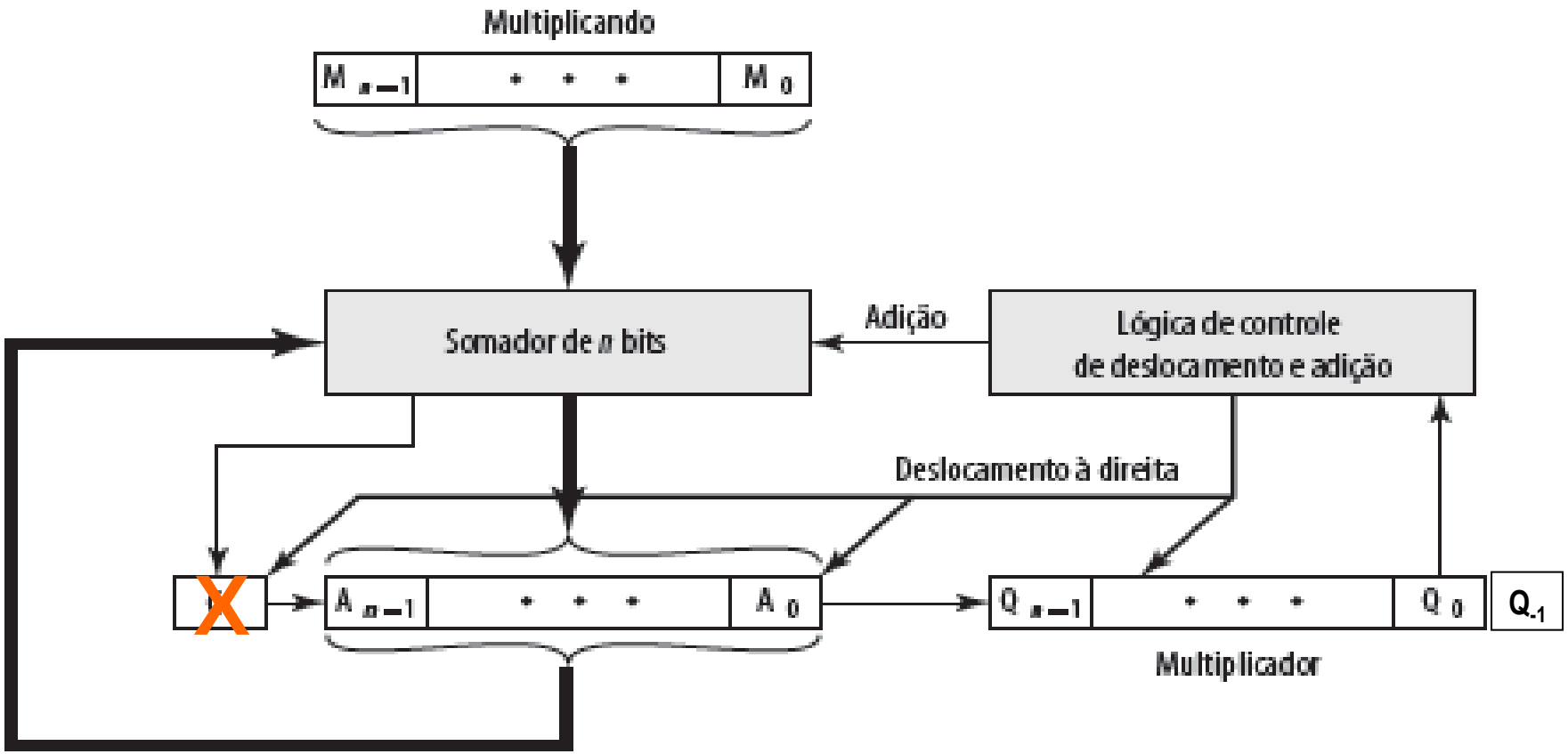
Multiplicando números negativos

- Complemento a dois direto não funciona!
- Solução 1:
  - Converta para positivo, se for preciso
  - Multiplique como antes
  - Se sinais diferentes, negue a resposta

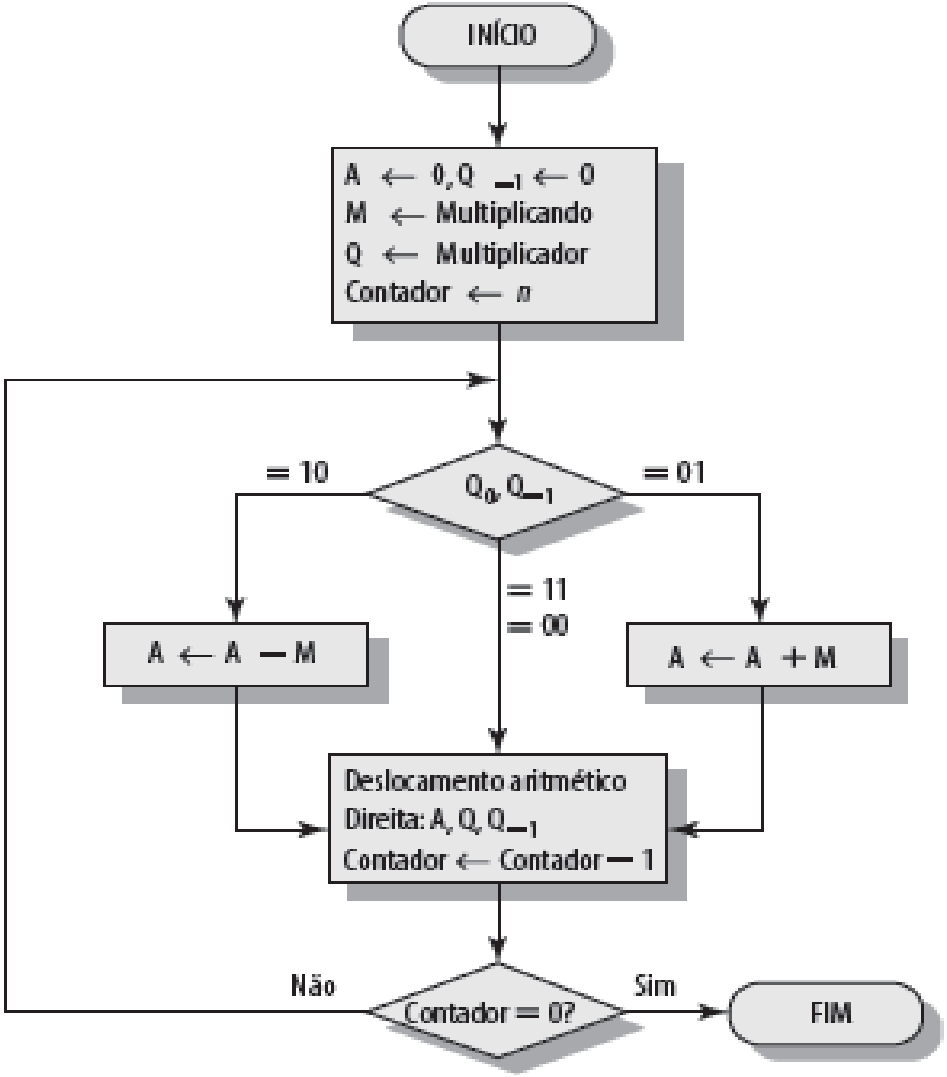
Multiplicando números negativos

- Complemento a dois direto não funciona!
- Solução 1:
  - Converta para positivo, se for preciso
  - Multiplique como antes
  - Se sinais diferentes, negue a resposta
- Solução 2:
  - Algoritmo de Booth

HW para algoritmo de Booth



Algoritmo de Booth



Resultado - AQ

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^5 - 2^1] = M \times 30$

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^5 - 2^1] = M \times 30$

Ou:

$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^6 + 2^5 + 2^4 + 2^3 + 2^1]$

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^5 - 2^1] = M \times 30$

Ou:

$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^6 + 2^5 + 2^4 + 2^3 + 2^1]$

$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^7 - 2^3 + 2^2 - 2^1]$

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^5 - 2^1] = M \times 30$

Ou:

$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^6 + 2^5 + 2^4 + 2^3 + 2^1]$

$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^7 - 2^3 + 2^2 - 2^1]$

O algoritmo de Booth usa este esquema:

- Efetua uma subtração quando encontra o primeiro 1 de um bloco (1 – 0) e uma adição ao encontrar o fim do bloco (0 – 1).

Algoritmo de Booth - fundamento

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^4 + 2^3 + 2^2 + 2^1] = M \times 30$

$M \times [0\ 0\ 0\ 1\ 1\ 1\ 1\ 0] = M \times [2^5 - 2^1] = M \times 30$

Ou:

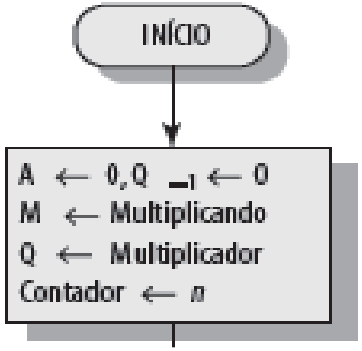
$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^6 + 2^5 + 2^4 + 2^3 + 2^1]$

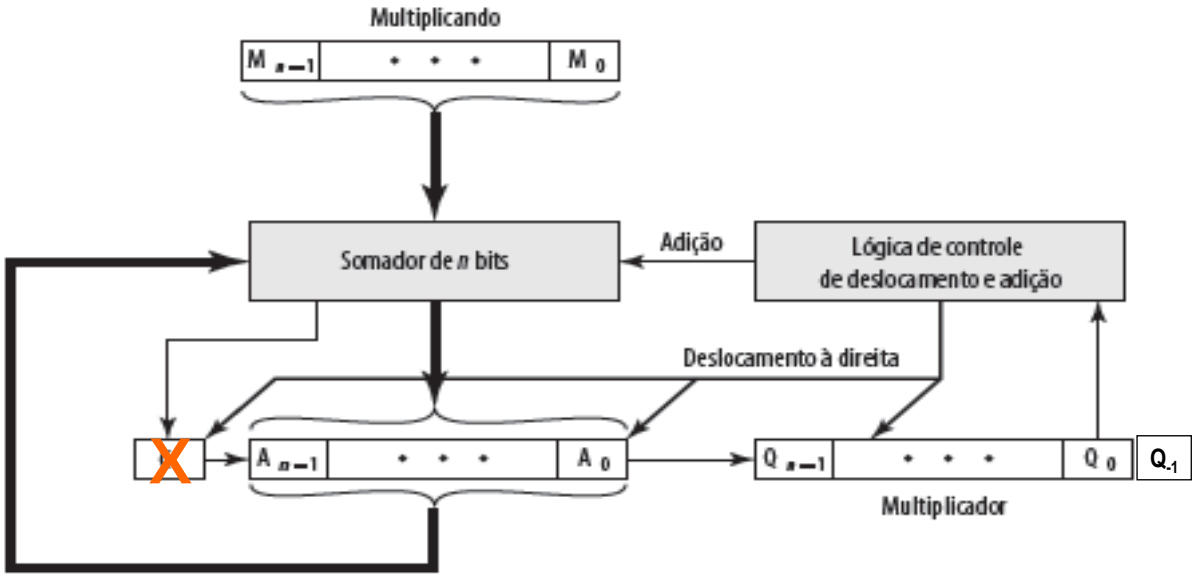
$M \times [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0] = M \times [2^7 - 2^3 + 2^2 - 2^1]$

O algoritmo de Booth usa este esquema:

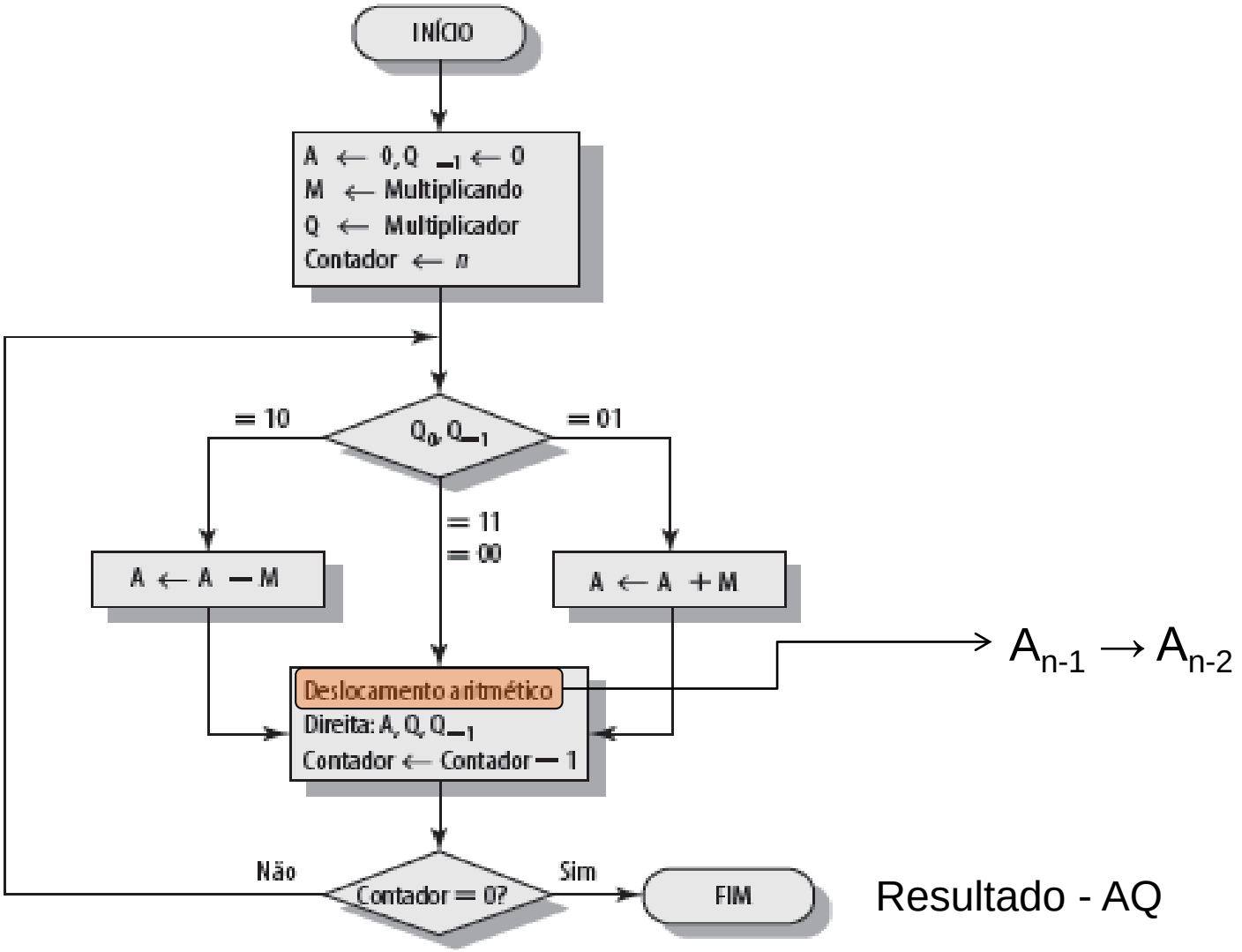
- Efetua uma subtração quando encontra o primeiro 1 de um bloco (1 – 0) e uma adição ao encontrar o fim do bloco (0 – 1).

Algoritmo de Booth



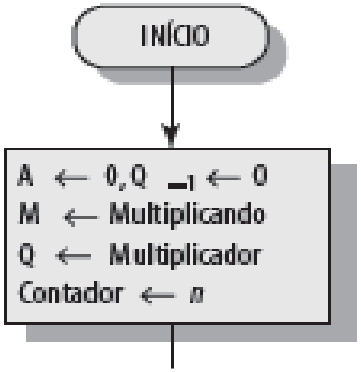


Algoritmo de Booth



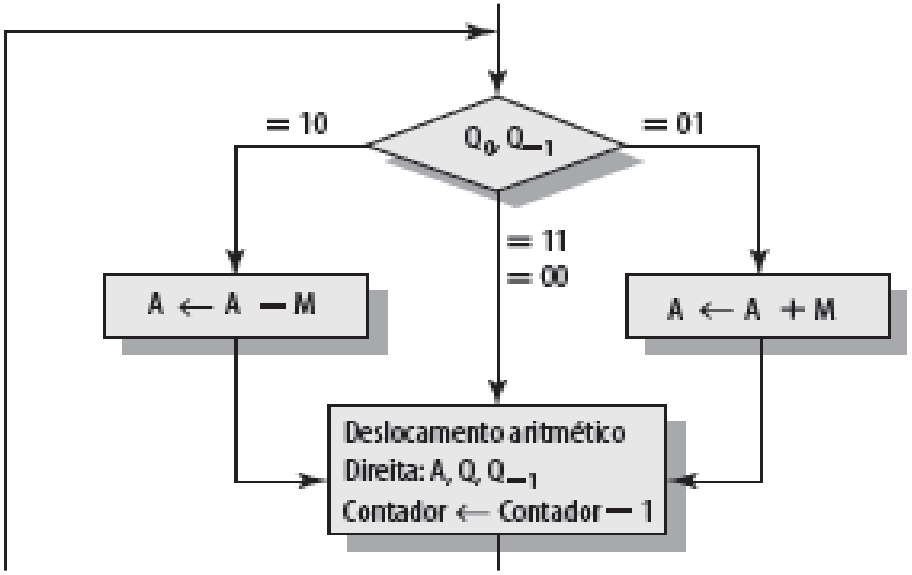
Exemplo do algoritmo de Booth: 7 x 3 (n = 4)

A	Q	Q_{-1}	M	
0000	0011	0	0111	Valores iniciais



Exemplo do algoritmo de Booth: 7 x 3

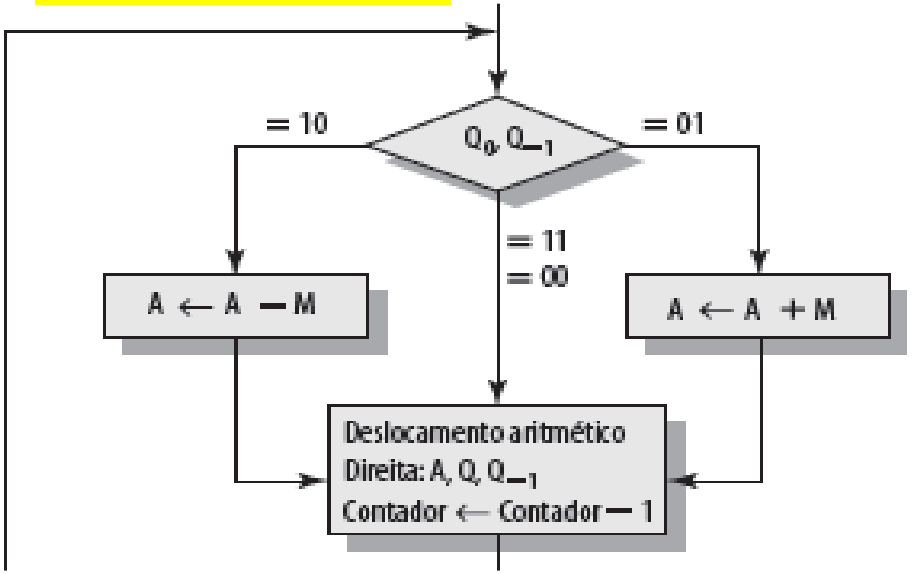
A	Q	Q_{-1}	M	
0000	0011	0	0111	Valores iniciais
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo



Exemplo do algoritmo de Booth: 7 x 3

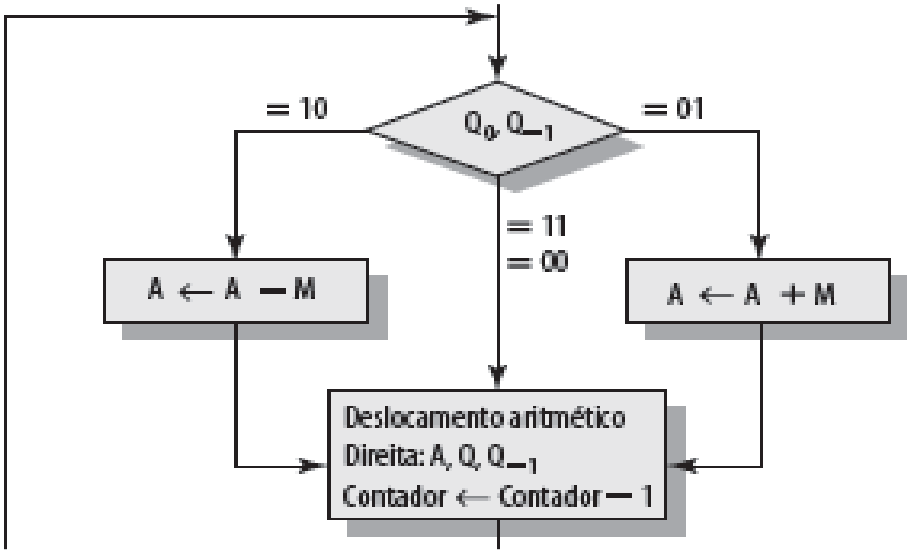
A	Q	Q_{-1}	M	
0000	0011	0	0111	Valores iniciais
1001	0011	0	0111	$A \leftarrow A - M$ } Primeiro
1100	1001	1	0111	Deslocamento } ciclo

ARITMÉTICO!!!



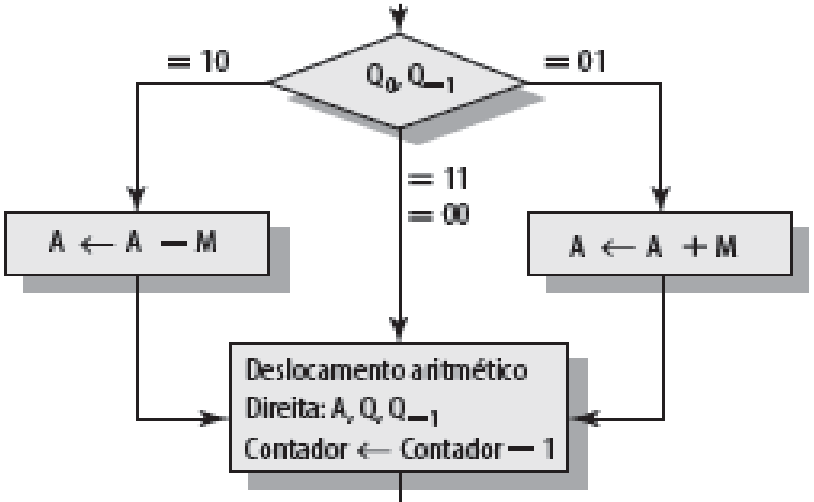
Exemplo do algoritmo de Booth: 7 x 3

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



Exemplo do algoritmo de Booth: 7 x 3

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



Exemplo do algoritmo de Booth: 7 x 3

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

→ = 21

Exrcício:

Use o algoritmo de Booth para calcular -7 x 3

Conteúdo

- ULA (ALU)
- Representação numérica
- Operações básica
- Algoritmo de Booth
- **Números reais e ponto flutuante**

Números reais

- Números com frações

Números reais

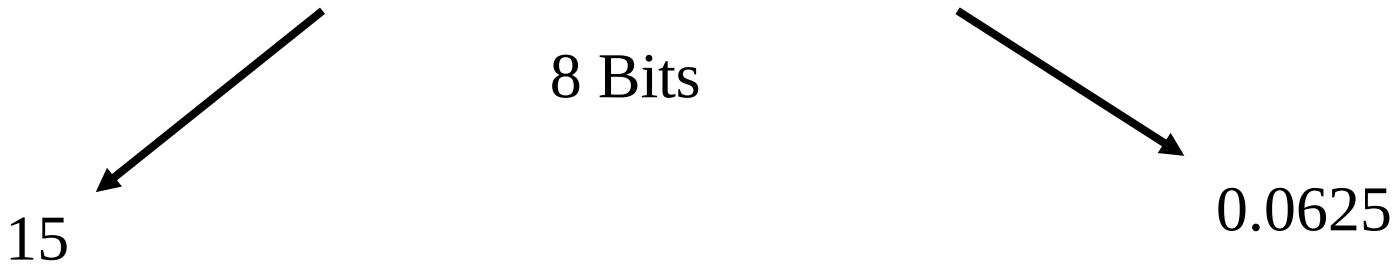
- Números com frações
- Poderia ser feito em binário puro
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625$

Números reais

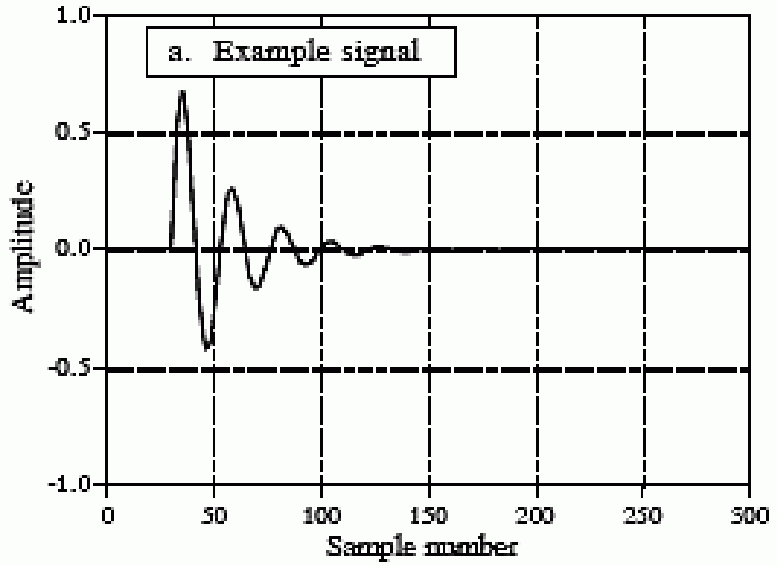
- Números com frações
- Poderia ser feito em binário puro
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625$
- Ponto binário fixo
  - Representação Muito limitada
    - Para **grandes números** e para **pequenas frações**

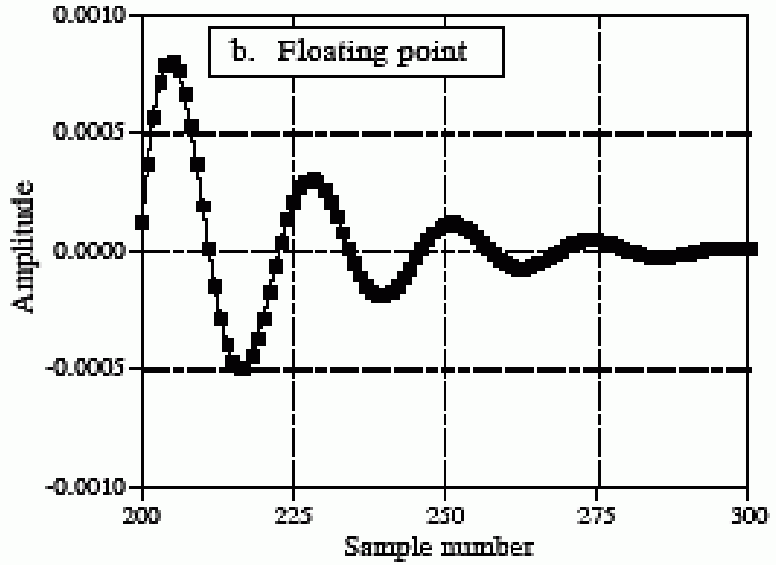
Números reais

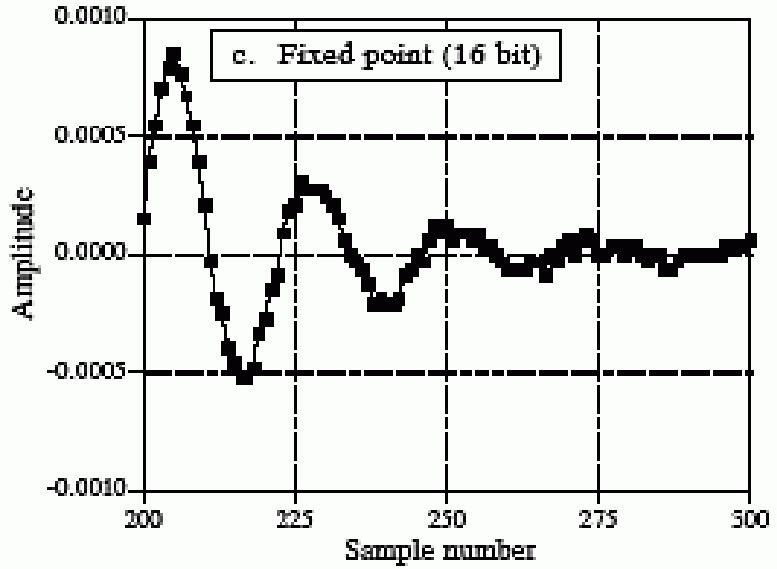
- Números com frações
- Poderia ser feito em binário puro
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625$
- Ponto binário fixo
  - Representação Muito limitada
    - Para **grandes números** e para **pequenas frações**

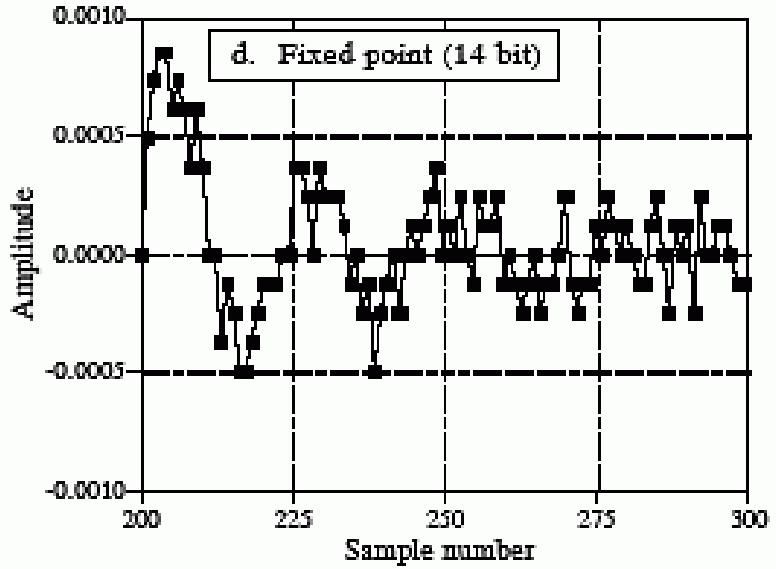


Ex. limitação ponto fixo





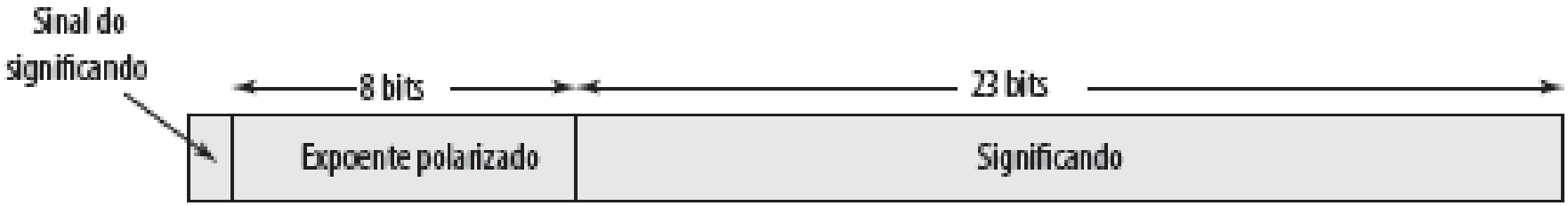




Números reais

- Números com frações
- Poderia ser feito em binário puro
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625$
- Ponto binário fixo
- **Ponto móvel**
  - **Como você mostra onde ele está?**
  - **E o sinal?**

Ponto flutuante - REPRESENTAÇÃO



- +/- significando x $2^{exponente}$
- Nome impróprio
- Ponto é realmente fixo entre bit de sinal e corpo da mantissa
- Expoente é que "se move" e indica valor da casa (posição do ponto)

Detalhes da implementação

1- Polarização do expoente

- Expoente está em notação polarizada
  - **Polarização = $2^{k-1}-1$**
  - Ex.: polarização 127 significa campo de expoente com 8 bits

1- Polarização do expoente

- Expoente está em notação polarizada
  - **Polarização = $2^{k-1}-1$**
  - Ex.: polarização 127 significa campo de expoente com 8 bits
  - Intervalo de valor puro 0 até 255
  - Subtraia 127 para obter valor correto:
    - Intervalo de -127 a +128

1- Polarização do expoente

- Expoente está em notação polarizada
  - **Polarização = $2^{k-1}-1$**
  - Ex.: polarização 127 significa campo de expoente com 8 bits
  - Intervalo de valor puro 0 até 255
  - Subtraia 127 para obter valor correto:
    - Intervalo de -127 a +128
- Assim, é possível representar números fracionários (com Expoente < 0)

2 - Normalização

- Várias possibilidades de representação:

$$0.110\text{x}2^{5} = 110\text{x}2^{2} = 0.0110\text{x}2^{6}$$

2 - Normalização

- Várias possibilidades de representação:

$$0.110\text{x}2^{5} = 110\text{x}2^{2} = 0.0110\text{x}2^{6}$$

- Números de PF geralmente são **normalizados**, ou seja, expoente é ajustado de modo que bit inicial (MSB) da mantissa (significando) **seja 1**
- Por ser sempre 1, não é preciso armazená-lo

2 - Normalização

- Várias possibilidades de representação:

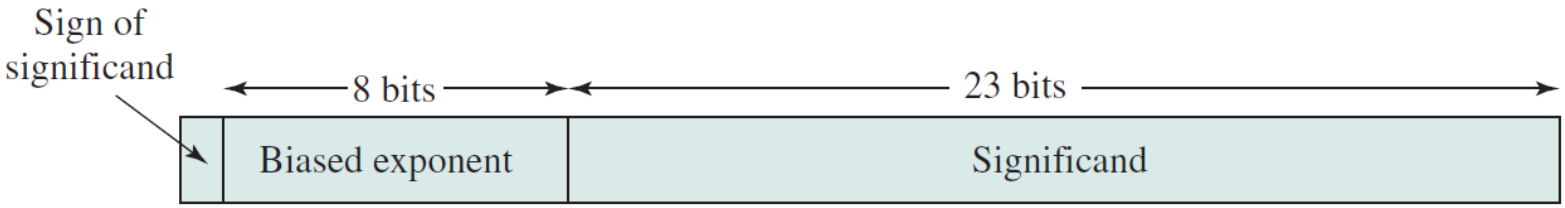
$$0.110\text{x}2^{5} = 110\text{x}2^{2} = 0.0110\text{x}2^{6}$$

- Números de PF geralmente são **normalizados**, ou seja, expoente é ajustado de modo que bit inicial (MSB) da mantissa (significando) **seja 1**
- Por ser sempre 1, não é preciso armazená-lo
- Como na notação científica, onde os números são normalizados para um único dígito antes do ponto decimal: 3,123 x 10^3

$$PF = \pm 1.bbbbbbb\ldots b\text{x}2^{E}, b = (0\,ou\,1)$$

Exemplos de ponto flutuante – *32 bits*

$$PF = \pm 1.bbbbbb\ldots b \times 2^{E}, b = (0\,ou\,1)$$



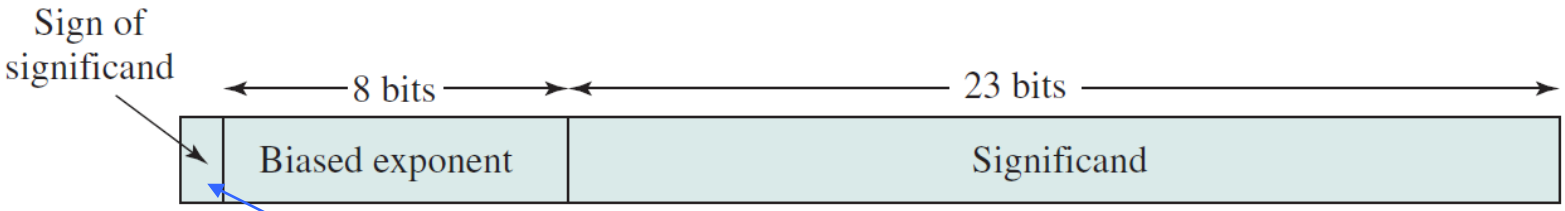
(a) Format

$1.1010001 \times 2^{10100}$
$-1.1010001 \times 2^{10100}$
$1.1010001 \times 2^{-10100}$
$-1.1010001 \times 2^{-10100}$

1's são desprezados (sem necessidade de armazenamento)

Exemplos de ponto flutuante – *32 bits*

$$PF = \pm 1.bbbbbb\ldots b \times 2^{E}, b = (0\,ou\,1)$$



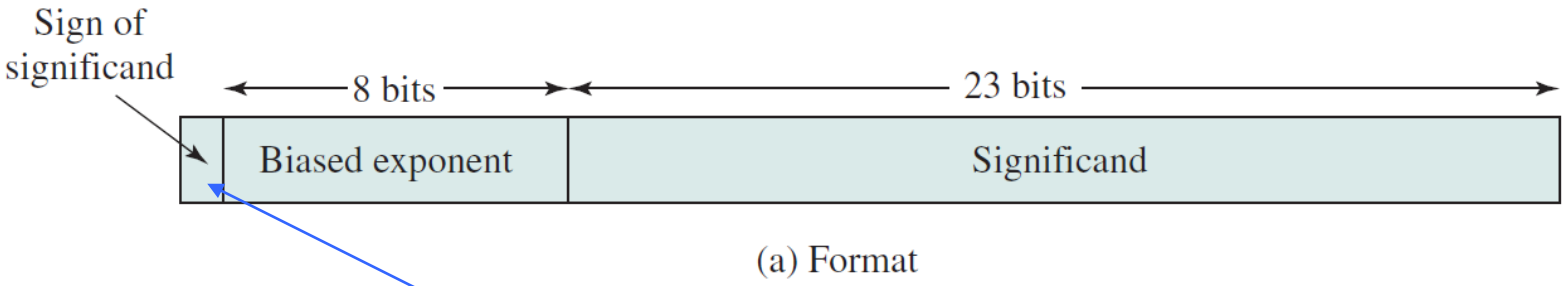
(a) Format

$1.1010001 \times 2^{10100} = 0$
$-1.1010001 \times 2^{10100} = 1$
$1.1010001 \times 2^{-10100} = 0$
$-1.1010001 \times 2^{-10100} = 1$

1's são desprezados (sem necessidade de armazenamento)

Exemplos de ponto flutuante – *32 bits*

$$PF = \pm 1.bbbbbb\ldots b \times 2^{E}, b = (0\,ou\,1)$$



(a) Format

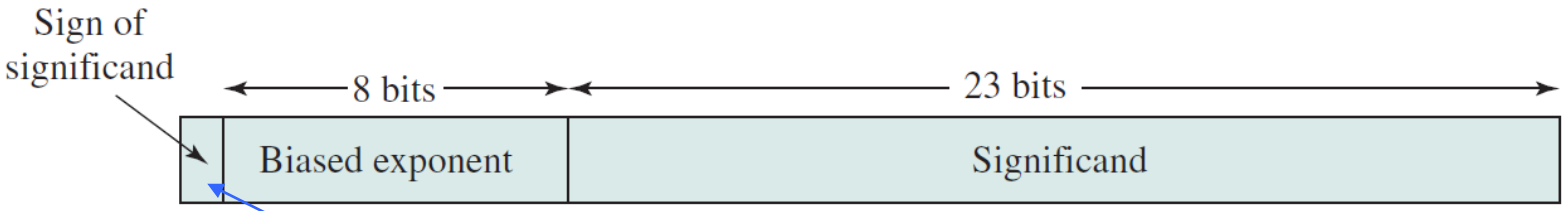
```
 1.1010001 × 2^10100  = 0 10010011 10100010000000000000000 =  1.6328125 × 2^20
-1.1010001 × 2^10100  = 1 10010011 10100010000000000000000 = -1.6328125 × 2^20
 1.1010001 × 2^-10100 = 0 01101011 10100010000000000000000 =  1.6328125 × 2^-20
-1.1010001 × 2^-10100 = 1 01101011 10100010000000000000000 = -1.6328125 × 2^-20
```

127 é adicionado ao valor do expoente antes de armazenar no campo correspondente

1's são desprezados (sem necessidade de armazenamento)

Exemplos de ponto flutuante – *32 bits*

$$PF = \pm 1.bbbbbb\ldots b \times 2^{E}, b = (0\,ou\,1)$$



(a) Format

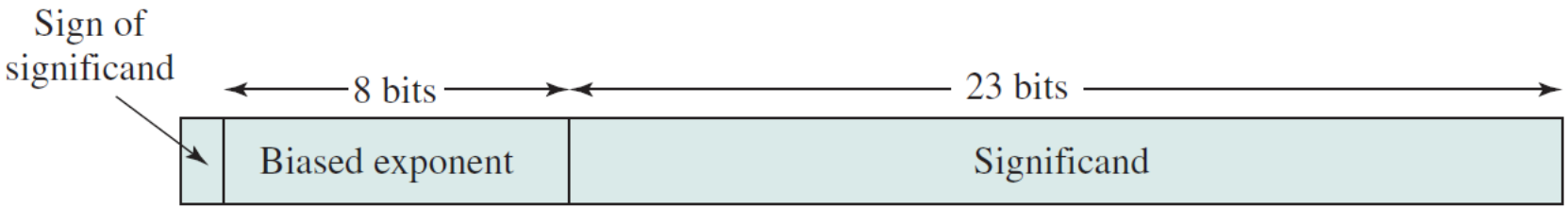
23 posições

$1.1010001 \times 2^{10100}$	=	0	10010011	10100010000000000000000	=	1.6328125×2^{20}
$-1.1010001 \times 2^{10100}$	=	1	10010011	10100010000000000000000	=	-1.6328125×2^{20}
$1.1010001 \times 2^{-10100}$	=	0	01101011	10100010000000000000000	=	1.6328125×2^{-20}
$-1.1010001 \times 2^{-10100}$	=	1	01101011	10100010000000000000000	=	$-1.6328125 \times 2^{-20}$

127 é adicionado ao valor do expoente antes de armazenar no campo correspondente

1's são desprezados (sem necessidade de armazenamento)

Exemplo: ***-10.5***


Sign of
significand
8 bits
23 bits
Biased exponent
Significand

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)
5. Ignora o primeiro 1:
6. $\Rightarrow$ Mantissa = 01010000...0

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)
5. Ignora o primeiro 1:
6. $\Rightarrow$ Mantissa = 01010000...0
7. Usar 23 bits: **Mantissa** = 01010000000000000000000

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)
5. Ignora o primeiro 1:
6. $\Rightarrow$ Mantissa = 01010000...0
7. Usar 23 bits: **Mantissa** = 01010000000000000000000
8. Expoente é +3.

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)
5. Ignora o primeiro 1:
6. ⇒ Mantissa = 01010000...0
7. Usar 23 bits: **Mantissa** = 01010000000000000000000
8. Expoente é +3.

9. Soma polarização 127:

127 + (+3) = 130 decimal ⇒ E = 10000010 binário.

Exemplo: ***-10.5***

1. Converter: -10.5 decimal --> -1010.1 binário
2. Normalizar: -1010.1 binário --> -1.0101 * 2^(+3)
3. **Sinal** = 1, pois o número é negativo
4. Mantissa (101010000...0)
5. Ignora o primeiro 1:
6. ⇒ Mantissa = 01010000...0
7. Usar 23 bits: **Mantissa** = 01010000000000000000000
8. Expoente é +3.

9. Soma polarização 127:

   127 + (+3) = 130 decimal ⇒ E = 10000010 binário.

10. Resultado:

    11000001010100000000000000000

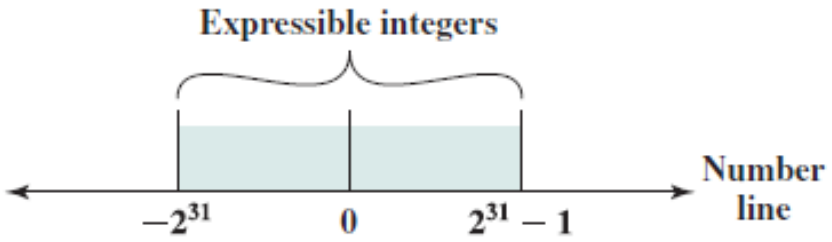
Intervalos dos valores em PF

Com **32 bits**, números nas seguintes faixas podem ser representados:

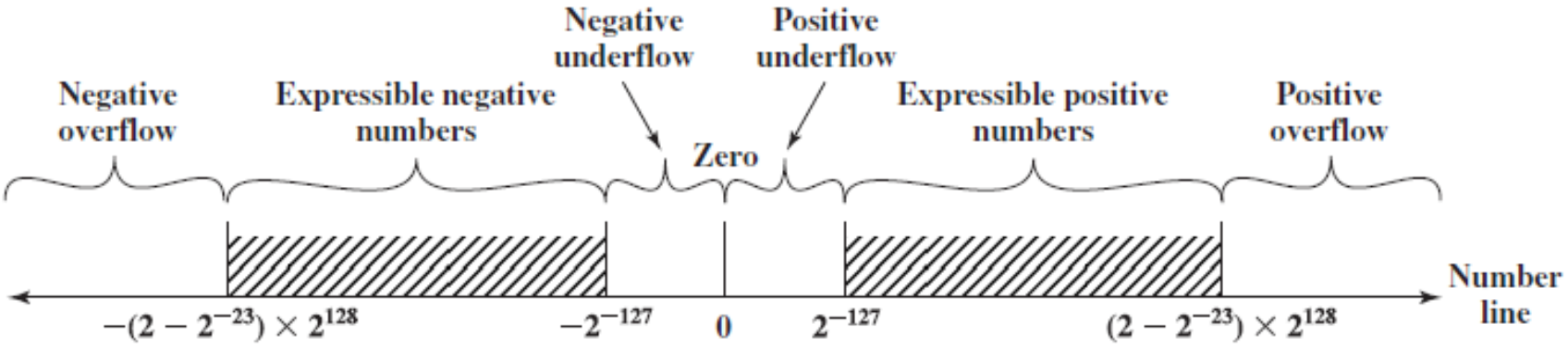
Números negativos entre $-(1 - 2^{-24}) \times 2^{128}$ e -0.5×2^{-127}

Números positivos entre 0.5×2^{-127} e $(1 - 2^{-24}) \times 2^{128}$

Números representáveis

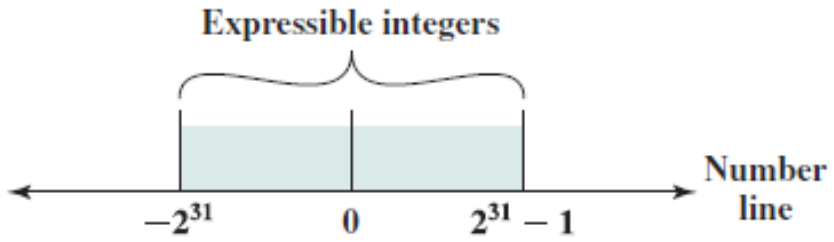


(a) Twos complement integers

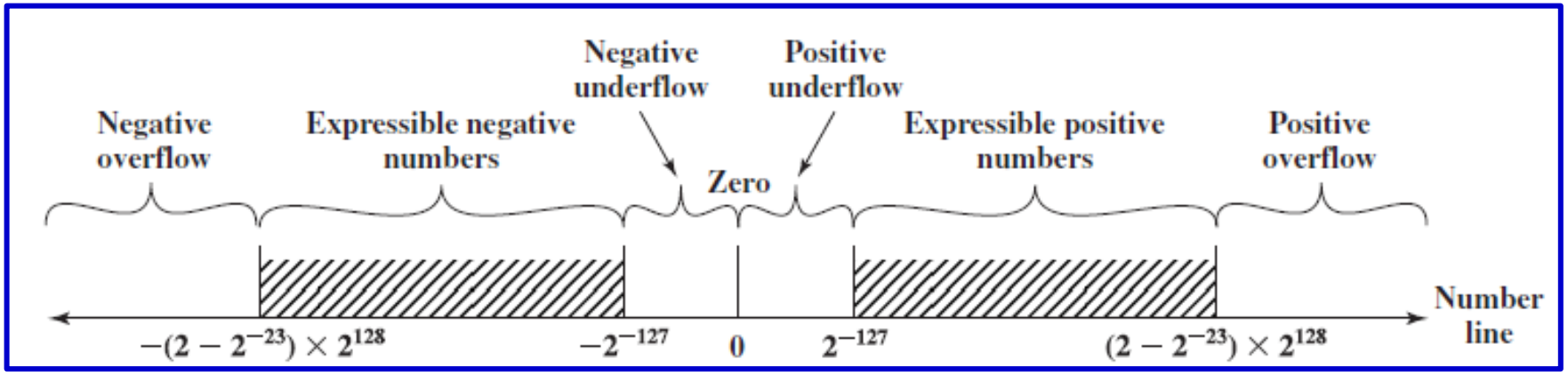


(b) Floating-point numbers

Números representáveis



(a) Twos complement integers

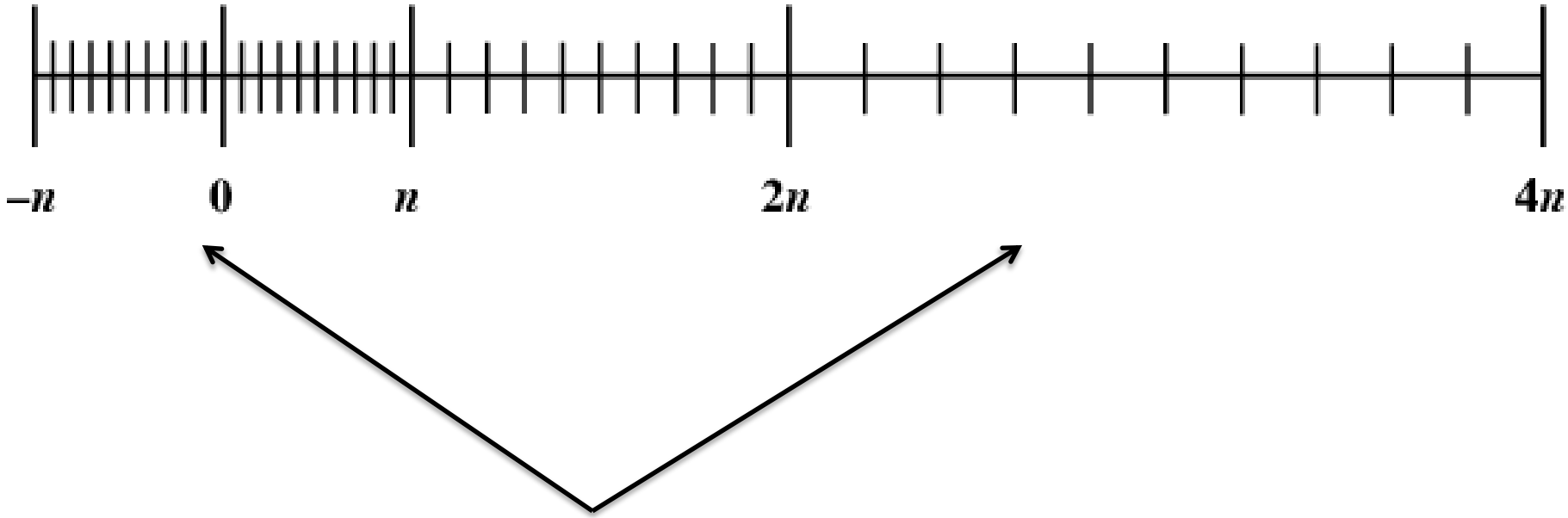


(b) Floating-point numbers

Números negativos entre $-(2 - 2^{-23}) \times 2^{128}$ e -2^{-127}

Números positivos entre 2^{-127} e $(2 - 2^{-23}) \times 2^{128}$

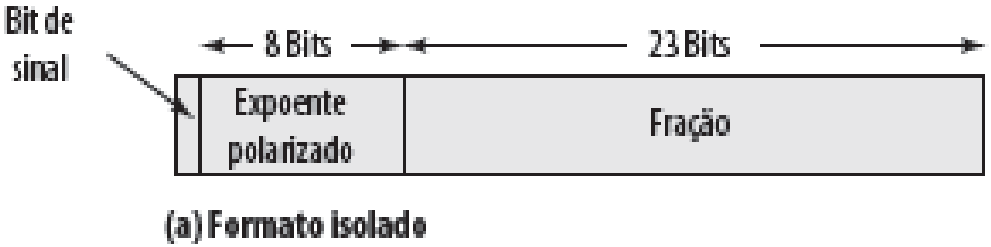
Densidade dos números de ponto flutuante



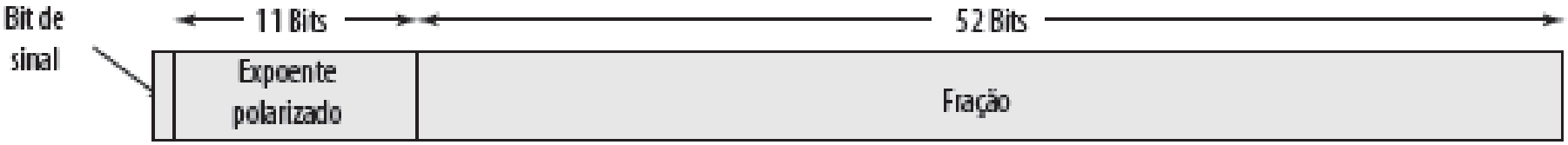
IEEE 754

- Padrão para armazenamento de ponto flutuante
- Padrões de 32 e 64 bits
- Expoente de 8 e 11 bits, respectivamente

Formatos IEEE 754



(a) Formato isolado



(b) Formato duplo

Exercício:

Converter 0,645 para notação em ponto flutuante com expoente polarizado de 8 bits e mantissa de 23 bits

Exercício:

Converter 0,645 para notação em ponto flutuante com expoente polarizado de 8 bits e mantissa de 23 bits

Solucao:

$0{,}645 = 1{,}01001... \times 2^{-1}$;

Exercício:

Converter 0,645 para notação em ponto flutuante com expoente polarizado de 8 bits e mantissa de 23 bits

Solucao:

$0{,}645 = 1{,}01001... \times 2^{-1}$;

Portanto, o significando é 01001 (o primeiro 1 é implícito)

Exercício:

Converter 0,645 para notação em ponto flutuante com expoente polarizado de 8 bits e mantissa de 23 bits

Solucao:

$0{,}645 = 1{,}01001... \times 2^{-1}$;

Portanto, o significando é 01001 (o primeiro 1 é implícito)

O sinal = 0

Exercício:

Converter 0,645 para notação em ponto flutuante com expoente polarizado de 8 bits e mantissa de 23 bits

Solucao:

$0{,}645 = 1{,}01001... \times 2^{-1}$;

Portanto, o significando é 01001 (o primeiro 1 é implícito)

O sinal = 0

E o expoente = -1 + 127 = 126

Resultado: 0 01111110 01001000000000000000000

Leitura recomendada

- Stallings, Capítulo 9
- IEEE 754 no *site* do IEEE