



UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Aritmética binária

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- ▶ Como é feita a representação numérica?
 - ▶ Bases numéricas
 - ▶ 2 (binário)
 - ▶ 10 (decimal)
 - ▶ 16 (hexadecimal)

$$\text{Número} = \sum_{i=0}^{n-1} B_i \times N^i$$

$$\begin{aligned} 33_{10} &= 1 \times 2^5 + 1 \times 2^0 = 100001_2 \\ &= 3 \times 10^1 + 3 \times 10^0 = 33_{10} \\ &= 2 \times 16^1 + 1 \times 16^0 = 21_{16} \end{aligned}$$

Introdução

- ▶ Como o sinal dos números binários é implementado?
 - ▶ Sinal e magnitude
 - ▶ Complemento a 1 e 2

Introdução

- ▶ Como o sinal dos números binários é implementado?
 - ▶ Sinal e magnitude
 - ▶ Complemento a 1 e 2
- ▶ De que maneira as principais operações aritméticas em formato binário são implementadas?
 - ▶ Adição
 - ▶ Subtração
 - ▶ Multiplicação
 - ▶ Divisão

Introdução

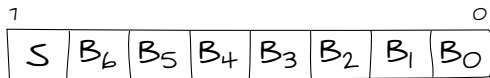
- ▶ Quais são os menores e maiores números que podem ser representados pela arquitetura?
 - ▶ Capacidade de armazenamento
 - ▶ Conceitos de *overflow* e *underflow*

Introdução

- ▶ Quais são os menores e maiores números que podem ser representados pela arquitetura?
 - ▶ Capacidade de armazenamento
 - ▶ Conceitos de *overflow* e *underflow*
- ▶ E as frações e os números reais?
 - ▶ Padrão IEEE 754

Representação do sinal

- ▶ Métodos para representação de sinal em números
 - ▶ Sinal e magnitude
 - ▶ Complemento a 1
 - ▶ Complemento a 2

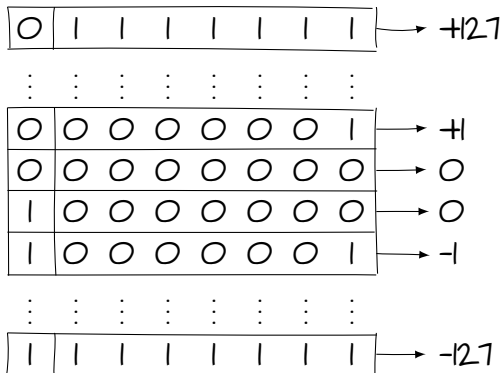


1 Bit de sinal + 7 Bits de dados

- ▶ Sinal e representatividade
 - ▶ $+$ $\leftrightarrow S = 0$ e 2^7 valores
 - ▶ $-$ $\leftrightarrow S = 1$ e 2^7 valores

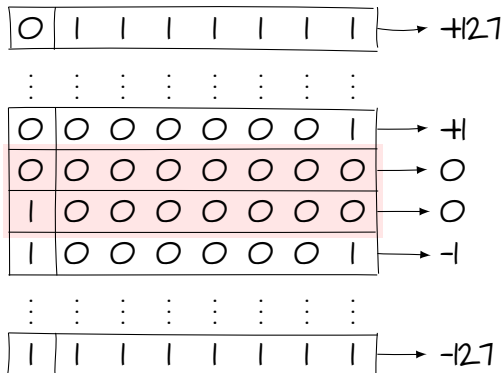
Representação do sinal

- ▶ Método de sinal e magnitude (8 bits)
 - ▶ Primeiro bit indica o sinal e demais bits a magnitude



Representação do sinal

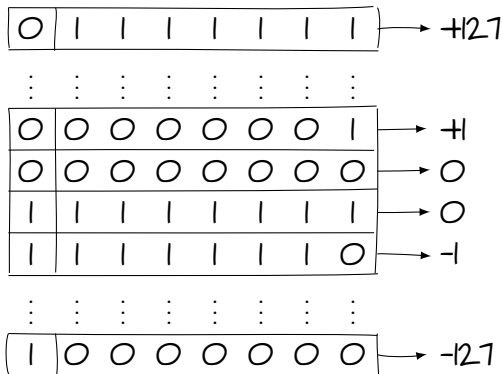
- ▶ Método de sinal e magnitude (8 bits)
 - ▶ Primeiro bit indica o sinal e demais bits a magnitude



Duas representações para o valor 0

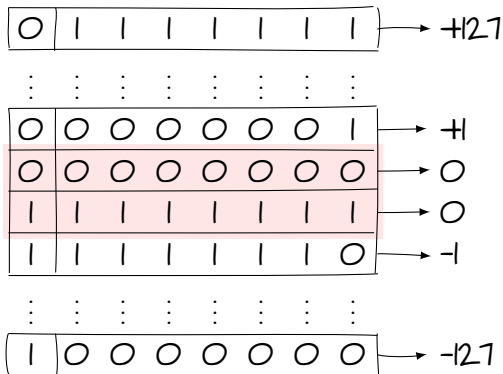
Representação do sinal

- ▶ Método de complemento a 1 (8 bits)
 - ▶ O valor de sinal oposto é o complemento bit a bit



Representação do sinal

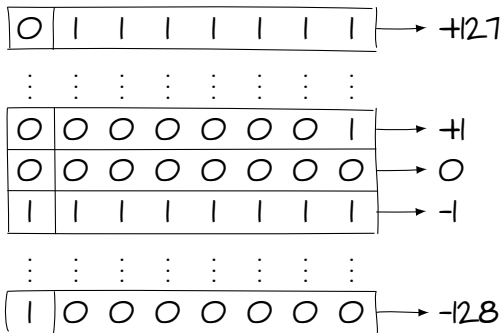
- ▶ Método de complemento a 1 (8 bits)
 - ▶ O valor de sinal oposto é o complemento bit a bit



Duas representações para o valor 0

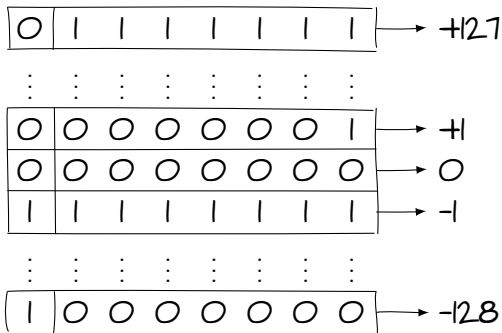
Representação do sinal

- ▶ Método de complemento a 2 (8 bits)
 - ▶ O valor de sinal oposto é o complemento bit a bit + 1



Representação do sinal

- ▶ Método de complemento a 2 (8 bits)
 - ▶ O valor de sinal oposto é o complemento bit a bit + 1



$2^7 = 128$ valores negativos e positivos

Operações sem sinal

► Adição binária de 8 bits

► $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$

► $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$

► $R = A + B = 225_{10} = 11100001_2$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{array}{rcccccccc} & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ + & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Operações sem sinal

► Adição binária de 8 bits

► $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$

► $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$

► $R = A + B = 225_{10} = 11100001_2$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{array}{r} 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\ \hline 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1 \end{array}$$

A adição de números com n bits pode gerar um resultado com até $n + 1$ bits

Operações sem sinal

► Subtração binária de 8 bits

► $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$

► $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$

► $R = A - B = 175_{10} = 10101111_2$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$\begin{array}{r} - \quad 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \\ \quad 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline \quad 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \end{array}$$

Operações sem sinal

► Subtração binária de 8 bits

► $A = 200_{10} = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 = 11001000_2$

► $B = 25_{10} = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 00011001_2$

► $R = A - B = 175_{10} = 10101111_2$

A_i	B_i	R_i	C_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$\begin{array}{r} 11001000 \\ - 00011001 \\ \hline 10101111 \end{array}$$

A subtração de números com n bits pode gerar um resultado com até $n + 1$ bits

Operações sem sinal

► Multiplicação binária de 8 bits

► $A = 11_{10} = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 00001011_2$

► $B = 13_{10} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 00001101_2$

► $R = A \times B = 143_{10} = 10001111_2$

A_i	B_i	R_i
0	0	0
0	1	0
1	0	0
1	1	1

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

Operações sem sinal

► Multiplicação binária de 8 bits

► $A = 11_{10} = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 00001011_2$

► $B = 13_{10} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 00001101_2$

► $R = A \times B = 143_{10} = 10001111_2$

A_i	B_i	R_i
0	0	0
0	1	0
1	0	0
1	1	1

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

A multiplicação de números com n Bits necessita de até $2n$ Bits para armazenar o resultado

Operações sem sinal

► Divisão binária de 8 bits

- $A = 143_{10} = 1 \times 2^7 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 10001111_2$
- $B = 10_{10} = 1 \times 2^3 + 1 \times 2^1 = 00001010_2$
- $Q = A \div B = 14_{10} = 00001110_2$
- $R = A \bmod B = 3_{10} = 00000011_2$

$$\begin{array}{r}
 \begin{array}{cccccccc|cccc}
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & & & & & & & \\
 - & & & & & & & & & & & & & & & & \\
 \hline
 & & 1 & 0 & 1 & 0 & & & & & & & & & & & \\
 \hline
 - & & 1 & & 1 & 1 & 1 & & & & & & & & & & \\
 - & & 1 & 0 & 1 & 0 & & & & & & & & & & & \\
 \hline
 & & & 1 & 0 & 1 & 1 & & & & & & & & & & \\
 - & & & 1 & 0 & 1 & 0 & & & & & & & & & & \\
 \hline
 & & & & & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & & \\
 & & & & & & 1 & 0 & 1 & 0 & & & & & & & \\
 \hline
 & & & & & & & 1 & & 1 & & & & & & & \\
 \hline
 & & & & & & & & & & & & & & & &
 \end{array}
 \end{array}$$

Operações sem sinal

► Divisão binária de 8 bits

- $A = 143_{10} = 1 \times 2^7 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 10001111_2$
- $B = 10_{10} = 1 \times 2^3 + 1 \times 2^1 = 00001010_2$
- $Q = A \div B = 14_{10} = 00001110_2$
- $R = A \bmod B = 3_{10} = 00000011_2$

$$\begin{array}{r} \begin{array}{cccccccc} & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ - & & 1 & 0 & 1 & 0 & & & \\ \hline & & 1 & 1 & 1 & 1 & & & \\ - & & 1 & 0 & 1 & 0 & & & \\ \hline & & & 1 & 0 & 1 & 1 & & \\ - & & & 1 & 0 & 1 & 0 & & \\ \hline & & & & & 1 & 1 & & \\ \hline & & & & & & & 1 & 1 \end{array} & \downarrow & \begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 \end{array} \end{array}$$

A divisão de números com n Bits necessita de até n Bits para o Quociente e o resto

Operações com sinal

- ▶ Adição/subtração binária de 8 bits
 - ▶ Método de sinal e magnitude
 - ▶ $A = -72_{10} = 11001000_2$
 - ▶ $B = +25_{10} = 00011001_2$
 - ▶ $R = A + B = -(|A| - |B|) = -47_{10} = 10101111_2$

$$\begin{array}{r} - \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\ \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \end{array}$$

É feito o ajuste do sinal de acordo com o resultado obtido pela magnitude

Operações com sinal

- ▶ Adição/subtração binária de 8 bits
 - ▶ Método de complemento a 1
 - ▶ $A = -72_{10} = 10110111_2$
 - ▶ $B = +25_{10} = 00011001_2$
 - ▶ $R = A + B = -47_{10} = 11010000_2$

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ + & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

O Bit de sinal é calculado como dado

Operações com sinal

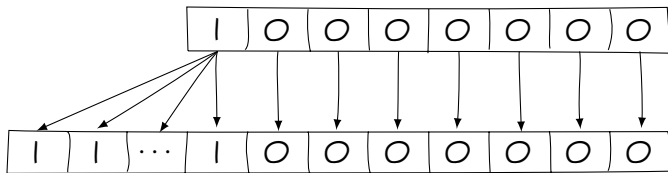
- ▶ Adição/subtração binária de 8 bits
 - ▶ Método de complemento a 2
 - ▶ $A = -72_{10} = 10111000_2$
 - ▶ $B = +25_{10} = 00011001_2$
 - ▶ $R = A + B = -47_{10} = 11010001_2$

$$\begin{array}{r} + \quad \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \\ \hline \quad \begin{array}{cccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \end{array}$$

Uma única representação para zero

Operações com sinal

- ▶ Extensão de sinal dos números
 - ▶ Codificação em complemento a 2
 - ▶ $A[8] = 10000000_2 = -128_{10}$
 - ▶ $B[32] = 1111 \dots 111110000000_2 = -128_{10}$

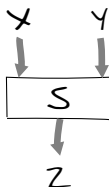


$$A = B$$

Implementação

- ▶ Como implementar de forma escalável e com portas lógicas as operações de adição e de subtração?
 - ▶ Método de complemento a 2

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

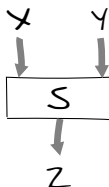


$$Z = X + Y = X \text{ xor } Y$$

Implementação

- ▶ Como implementar de forma escalável e com portas lógicas as operações de adição e de subtração?
 - ▶ Método de complemento a 2

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0



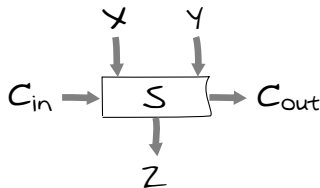
$$Z = X + Y = X \text{ xor } Y$$

Como fazer o "vai a um" (carry)?

Implementação

- ▶ Como implementar de forma escalável e com portas lógicas as operações de adição e de subtração?
 - ▶ Método de complemento a 2

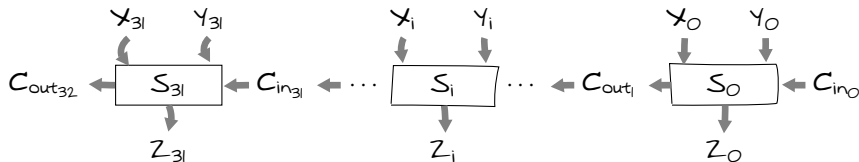
X	Y	C_{in}	Z	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



$$Z = X + Y + C_{in} = X \text{ xor } Y \text{ xor } C_{in}$$
$$C_{out} = (X \text{ and } Y) \text{ xor } (C_{in} \text{ and } (X \text{ xor } Y))$$

Implementação

- ▶ Como implementar de forma escalável e com portas lógicas as operações de adição e de subtração?
 - ▶ Método de complemento a 2



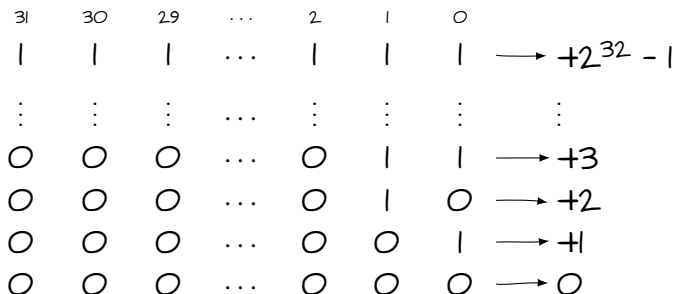
$$Z[32] = X[32] + Y[32]$$

Exemplo

- ▶ Considerando o método de codificação de complemento a 2 e capacidade de armazenamento de 8 bits, realize as operações abaixo passo a passo em codificação binária
 - ▶ $4 + 8 + 16 + 32$
 - ▶ $5 - 3 + 8 - 13$
 - ▶ $2 \times 3 \times 4 \times -5$

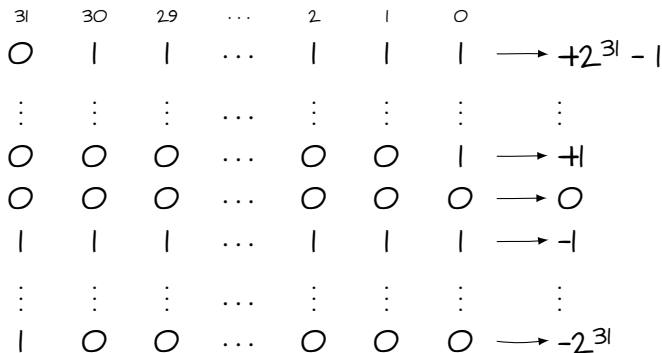
Limites numéricos

- ▶ Quais são os limites para os números inteiros?
 - ▶ Capacidade de 32 bits (sem sinal)



Limites numéricos

- ▶ Quais são os limites para os números inteiros?
 - ▶ Complemento a 2 com 32 bits (com sinal)



Limites numéricos

- ▶ O que acontece quando a capacidade de armazenamento do hardware é extrapolado?

Limites numéricos

- ▶ O que acontece quando a capacidade de armazenamento do hardware é extrapolado?
 - ▶ Capacidade de 8 bits
 - ▶ Sem sinal: é gerado um bit excedente (*carry*)

$$\begin{array}{r} + \quad 1 \ 5 \ 9 \\ \hline 2 \ 8 \ 0 \end{array} \longrightarrow \begin{array}{r} + \quad 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

Limites numéricos

- ▶ O que acontece quando a capacidade de armazenamento do hardware é extrapolado?

- ▶ Capacidade de 8 bits

- ▶ Sem sinal: é gerado um bit excedente (*carry*)

$$\begin{array}{r} + \quad 1 \ 5 \ 9 \\ \quad 1 \ 2 \ 1 \\ \hline 2 \ 8 \ 0 \end{array} \longrightarrow \begin{array}{r} + \quad 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \quad 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

- ▶ Com sinal: ocorre quando o sinal do resultado é diferente para operandos que possuem o mesmo sinal

$$\begin{array}{r} + \quad 1 \ 1 \ 9 \\ \quad 1 \ 0 \ 2 \\ \hline 2 \ 2 \ 1 \end{array} \longrightarrow \begin{array}{r} + \quad 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \\ \quad 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \end{array}$$

Limites numéricos

- ▶ O que acontece quando a capacidade de armazenamento do hardware é extrapolado?

- ▶ Capacidade de 8 bits

- ▶ Sem sinal: é gerado um bit excedente (*carry*)

$$\begin{array}{r} + \quad 1 \ 5 \ 9 \\ \quad 1 \ 2 \ 1 \\ \hline 2 \ 8 \ 0 \end{array} \longrightarrow \begin{array}{r} + \quad 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \quad 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

- ▶ Com sinal: ocorre quando o sinal do resultado é diferente para operandos que possuem o mesmo sinal

$$\begin{array}{r} + \quad 1 \ 1 \ 9 \\ \quad 1 \ 0 \ 2 \\ \hline 2 \ 2 \ 1 \end{array} \longrightarrow \begin{array}{r} + \quad 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \\ \quad 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \end{array}$$

↑ Overflow

↓ Underflow

Números reais

- ▶ Como representar os números reais?

Números reais

- ▶ Como representar os números reais?
 - ▶ Ponto fixo (notação \mathbb{Q})
 - ▶ Hardware padrão
 - ▶ Operações com números inteiros

Números reais

- ▶ Como representar os números reais?
 - ▶ Ponto fixo (notação Q)
 - ▶ Hardware padrão
 - ▶ Operações com números inteiros
 - ▶ Ponto flutuante (IEEE 754)
 - ▶ Hardware dedicado
 - ▶ Operações especializadas

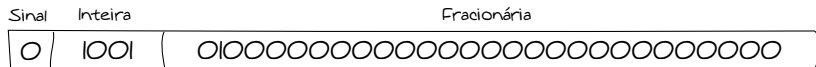
Números reais

- ▶ Como representar os números reais?
 - ▶ Ponto fixo (notação Q)
 - ▶ Hardware padrão
 - ▶ Operações com números inteiros
 - ▶ Ponto flutuante (IEEE 754)
 - ▶ Hardware dedicado
 - ▶ Operações especializadas
 - ▶ Precisão arbitrária
 - ▶ Emulação por software
 - ▶ Números inteiros ou reais

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ▶ Definição da parte inteira e fracionária (32 bits)

$$\begin{aligned} Q4.27 &= 9,25_{10} \\ &= 9_{10} + 0,25_{10} \\ &= 1001_2 + 0,01_2 \end{aligned}$$



Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

Números reais

- ▶ Aritmética de ponto fixo (notação \mathbb{Q})
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = (2^3 + \underline{1}) + 0,25_{10}$$

Números reais

- ▶ Aritmética de ponto fixo (notação \mathbb{Q})
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = (2^3 + 2^0) + 0,25_{10}$$

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + 0,25_{10}$$

Números reais

- ▶ Aritmética de ponto fixo (notação \mathbb{Q})
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + (2^{-2} + \underline{0})$$

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ▶ Entendendo a codificação binária fracionária

$$9,25_{10} = 9_{10} + 0,25_{10}$$

$$9,25_{10} = 1001_2 + 0,01_2$$

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ▶ Representatividade um número $Q2.2$

S	3	2	1	0	
0	1	1	1	1	→ +3,75
0	1	1	1	0	→ +3,50
0	1	1	0	1	→ +3,25
0	1	1	0	0	→ +3,00
⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	1	1	→ -4,00
1	0	0	1	0	→ -4,25
1	0	0	0	1	→ -4,50
1	0	0	0	0	→ -4,75

Números reais

- ▶ Aritmética de ponto fixo (notação \mathbb{Q})
 - ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
 - ✓ Maior precisão da parte fracionária com mesma quantidade de bits

Ponto fixo (Q0.31)

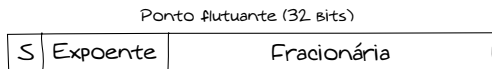
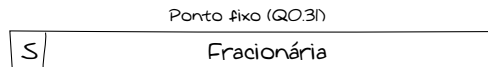


Ponto flutuante (32 bits)



Números reais

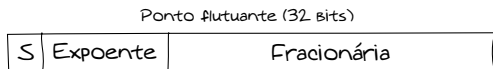
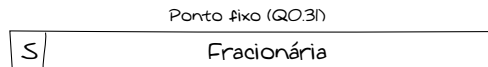
- ▶ Aritmética de ponto fixo (notação Q)
 - ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
 - ✓ Maior precisão da parte fracionária com mesma quantidade de bits



✗ Representatividade limitada

Números reais

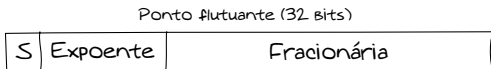
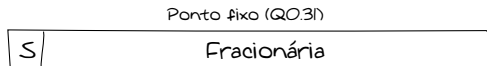
- ▶ Aritmética de ponto fixo (notação Q)
 - ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
 - ✓ Maior precisão da parte fracionária com mesma quantidade de bits



- ✗ Representatividade limitada
- ✗ Problemas com arredondamento e *overflow*

Números reais

- ▶ Aritmética de ponto fixo (notação Q)
 - ✓ São utilizados componentes de aritmética inteira que permitem operações mais rápidas e simples
 - ✓ Maior precisão da parte fracionária com mesma quantidade de bits



- ✗ Representatividade limitada
- ✗ Problemas com arredondamento e *overflow*

Aplicação principal: sistemas de baixo custo sem unidade de ponto flutuante (FPU)

Números reais

- ▶ Aritmética de ponto flutuante (IEEE 754)
 - ▶ Níveis de precisão

Simplex (32 Bits)

S	Expoente (8 Bits)	Fracionária (23 Bits)
---	----------------------	--------------------------

Dupla (64 Bits)

S	Expoente (11 Bits)	Fracionária (52 Bits)
---	-----------------------	--------------------------

Quádrupla (128 Bits)

S	Expoente (15 Bits)	Fracionária (112 Bits)
---	-----------------------	---------------------------

Números reais

- ▶ Aritmética de ponto flutuante (IEEE 754)
 - ▶ Representação dos valores de 32 bits

$$\text{float} = (-1)^{\text{Sinal}} \left(1 + \sum_{i=0}^{22} B_{22-i} 2^{-i} \right) \times 2^{(\text{Expoente}-127)}$$



$$\begin{aligned} 9,25_{10} &= 9_{10} + 0,25_{10} \\ &= 1001_2 + 0,01000000000000000000_2 \\ &= 1,001010000000000000000000_2 \times 2^3 \\ &= (-1)^0 (1_2 + 0,001010000000000000000000_2) \times 2^{(130_{10}-127_{10})} \\ &= (-1)^0 (1_2 + 0,001010000000000000000000_2) \times 2^{(10000010_2-127_{10})} \end{aligned}$$

Números reais

- ▶ Aritmética de ponto flutuante
 - ✓ Maior representatividade de valores

Números reais

- ▶ Aritmética de ponto flutuante
 - ✓ Maior representatividade de valores
 - ✓ Mecanismos de arredondamento e de precisão

Números reais

- ▶ Aritmética de ponto flutuante
 - ✓ Maior representatividade de valores
 - ✓ Mecanismos de arredondamento e de precisão
 - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema

Números reais

- ▶ Aritmética de ponto flutuante
 - ✓ Maior representatividade de valores
 - ✓ Mecanismos de arredondamento e de precisão
 - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema
 - ✗ As operações são mais complexas e demoradas

Números reais

- ▶ Aritmética de ponto flutuante
 - ✓ Maior representatividade de valores
 - ✓ Mecanismos de arredondamento e de precisão
 - ✗ Hardware dedicado que aumenta o custo e o consumo de potência do sistema
 - ✗ As operações são mais complexas e demoradas

Aplicação principal: redução do tempo de projeto em sistemas com unidade de ponto flutuante (FPU)

Números reais

- ▶ Aritmética de precisão arbitrária
 - ▶ Geralmente é utilizada em aplicações com números inteiros com capacidade que depende somente da quantidade de memória disponível, sendo nativamente suportada em linguagens como Python

Números reais

- ▶ Aritmética de precisão arbitrária
 - ▶ Geralmente é utilizada em aplicações com números inteiros com capacidade que depende somente da quantidade de memória disponível, sendo nativamente suportada em linguagens como Python
 - ▶ As linguagens de programação que não suportam diretamente a aritmética de precisão arbitrária, como C/C++, podem utilizar bibliotecas para suportar estas operações (GMP/MPFR)

Números reais

- ▶ Aritmética de precisão arbitrária
 - ▶ Geralmente é utilizada em aplicações com números inteiros com capacidade que depende somente da quantidade de memória disponível, sendo nativamente suportada em linguagens como Python
 - ▶ As linguagens de programação que não suportam diretamente a aritmética de precisão arbitrária, como C/C++, podem utilizar bibliotecas para suportar estas operações (GMP/MPFR)
 - ▶ Aplicações
 - ▶ Cálculo de constantes matemáticas
 - ▶ Criptografia de chave pública
 - ▶ Descoberta de números primos
 - ▶ ...

Exemplo

- Considerando os métodos de complemento a 2, de ponto fixo Q2.5 e de ponto flutuante F8 descritas abaixo, converta os números reais $A = 2,71$ e $B = 3,14$ para estas representações numéricas

$$F8 = (-1)^S \left(1 + \sum_{i=0}^4 B_{4-i} 2^{-i} \right) \times 2^{\text{Expoente}}$$

	Sinal	Inteira	Fracionária
Q2.5	S	i ₁ i ₀	f ₄ f ₃ f ₂ f ₁ f ₀

	Sinal	Expoente	Fracionária
F8	S	e ₁ e ₀	f ₄ f ₃ f ₂ f ₁ f ₀

- Realize a operação $A - B$ para cada representação e compare erro dos resultados obtidos
- Verifique o que seria necessário para implementar as operações de divisão e de multiplicação