Aritmética de computador

Representação de números em diferentes bases

Números inteiros vs. Números reais

- É comum para a maioria dos computadores, o uso de uma base numérica distinta da base decimal
- Em geral, os números são armazenados na
  - **base 2** (**binária**) – mais comum
  - **base 8** (**octal**)
  - **base 16** (**hexadecimal**)
- Um número pode ser representado usando
  - o sistema posicional
  - a forma polinomial

Números inteiros vs. Números reais

- A representação de **números inteiros** é ligeiramente **distinta** da representação de **números reais**
- Representação de números inteiros segue um formato:
  - de Sinal-Magnitude
- Representação de números reais seguem dois formatos:
  - de ponto fixo
  - de ponto flutuante

Números inteiros

- Um número inteiro **N** é representado, numa **base b**, por uma sequência de dígitos $\mathbf{a_i}$ em que
  - $\mathbf{a_i} \in \{ 0, 1, \ldots, b-1 \}$
  - **i** assume um intervalo de valores que depende da base em uso
- As bases mais utilizadas são:

base	$\mathbf{a_i}$
2	0, 1
8	0, 1, 2, 3, 4, 5, 6, 7
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Números inteiros

- Representação em formato de **Sinal-Magnitude**
  - no sistema posicional
  - na forma polinomial
- No **sistema posicional**
  - os dígitos são **agrupados** numa **sequência**
  - a dimensão da contribuição de cada dígito no número depende da posição que ocupa
  - um número **N** é escrito com o seguinte formato:

$$\mathbf{N = (a_n\ a_{n-1}\ \ldots\ a_1\ a_0)_b}$$

- Na **forma polinomial**
  - fica claramente explicitada a contribuição de cada dígito para o valor de N
  - um número N é escrito com o seguinte formato:

$$\mathbf{N = a_n\ b^n + a_{n-1}\ b^{n-1} + \ldots + a_1\ b + a_0}$$

Números reais

- Um número pode ser representado usando dois formatos:
  - **de ponto fixo** (por exemplo, 12.34)
  - **de ponto flutuante** (por exemplo, 0.4273×10^2)
- Representação em formato **de ponto fixo**
  - Um número real **X** é composto por duas partes:
    - a parte inteira $\mathbf{X_i}$ (número inteiro)
    - a parte fracionaria $\mathbf{X_f}$ (número real), em que $\mathbf{X_f} = X - \mathbf{X_i}$
  - Exemplo:

$X = 12.34$

$X_i = 12$

$X_f = 0.34$

Números reais

- Representação em formato **de ponto flutuante**
  - a notação é semelhante à notação científica:

$\pm .d_1 d_2 d_3 \ldots d_p \times b^e$,

em que

- $\mathbf{d_k}$ (k = 1,..., p) são os dígitos da parte fracionária com
  - $d_k \in \{ 0, \ldots, b-1 \}$
  - $d_1 \neq 0$ (se o número for normalizado)
- **b** é o valor da base (geralmente 2, 8, 10 ou 16)
- **p** é o número de dígitos da parte fracionária
- **e** é um expoente inteiro

Números reais

- Representação em formato **de ponto flutuante**
  - um **número real** é composto por três partes:
    - o **sinal**
    - a **parte fracionária** (**significando** ou **mantissa**)
    - o **expoente**
  - as três partes tem um comprimento total **fixo** que **depende**
    - do **computador** e
    - do **tipo de número** (precisão **simples**, **dupla** ou **estendida**)

Números reais

- Representação em formato **de ponto flutuante**
  - Exemplo (na base decimal):

X = 0.4273x10^3

- representado por:
  - $d_1\ d_2\ d_3\ d_4 = 4273$
  - b = 10
  - p = 4
  - e = 3
- composto por:
  - sinal: + (omitido)
  - mantissa: 4273
  - expoente: 3

Números reais

- Representação em computador
  - A representação de um número pode ser diferente entre os fabricantes do computadores, logo
    - o **mesmo programa** implementado em computadores que utilizam **formatos diferentes** pode fornecer **resultados diferentes**

Números reais

- Representação em computador
  - O formato utilizado pela maioria dos computadores é padrão IEEE 754 (para binários):

Propriedades	Precisão		
	Simples	**Dupla**	**Estendida**
Comprimento total	32	64	80
bits na mantissa	23	52	64
bits no expoente	8	11	15
sinal	1	1	1
expoente máximo	127	1023	16383
expoente mínimo	-126	-1022	-16382
maior número (absoluto)	$\approx 3.40 \times 10^{38}$	$\approx 1.80 \times 10^{308}$	$\approx 1.19 \times 10^{4932}$
menor número (absoluto)	$\approx 1.18 \times 10^{-38}$	$\approx 2.123 \times 10^{-308}$	$\approx 3.36 \times 10^{-4932}$
dígitos decimais (precisão)	7	16	19

Conversão de números de decimal para binário

Números inteiros

- Um dos métodos é o das **divisões sucessivas** que consiste em dividir por 2
  - **N** (base decimal) e os sucessivos **quocientes q_i**
  - são guardados os **restos $r_i \in \{ 0, 1 \}$ até** que o **último quociente** seja **$q_n = 1$**
- Método

$$N = 2\ q_1 + r_0$$
$$q_1 = 2\ q_2 + r_1$$
$$\ldots$$
$$q_{n-1} = 2\ q_n + r_{n-1}$$

  - de notar que o último quociente será 0 apenas se N = 0 ($q_n = 0 \Leftrightarrow N = 0$)
- Sintaxe

$$N = (q_n\ r_{n-1}\ \ldots\ r_1\ r_0)_2 \quad \text{(sistema posicional)}$$
$$N = q_n\ 2^n + r_{n-1}\ 2^{n-1} + r_{n-2}\ 2^{n-2} + \ldots + r_1\ 2^1 + r_0 \quad \text{(forma polinomial)}$$

Números reais em formato de ponto fixo

- Dado um número real X, este é composto por
  - uma parte inteira X_i (número inteiro) e
  - uma parte fracionaria X_f
- Para se converter este número X na base binária utiliza-se
  - o **método das divisões sucessivas** para converter $\mathbf{X_i}$ e
  - o **método das multiplicações sucessivas** para converter $\mathbf{X_f}$
- O **método das multiplicações sucessivas** consiste em
  - multiplicar-se $\mathbf{X_f}$ **por 2**, extraindo-se a parte inteira do resultado (que pode ser 0)
  - o **resto** é novamente multiplicado por **2**,
  - repetindo-se o processo até que
    - o resto fracionário seja 0 ou
    - se obtenha um padrão repetitivo (o número fracionário será periódico)

Números reais em formato de ponto fixo

- Exemplo 1:

Seja X_f = 0.8125, então

0.8125 x 2 = **1.**6250;

0.6250 x 2 = **1.**2500;

0.2500 x 2 = **0**.5000;

0.5000 x 2 = **1**.0000

Ou seja,

$0.8125 = (0.1101)_2$

Números reais em formato de ponto fixo

- Exemplo 2:

Seja $\mathbf{X_f = 0.1}$, então

0.1 x 2 = **0.**2

0.2 x 2 = **0.**4

0.4 x 2 = **0.**8

0.8 x 2 = **1.**6

0.6 x 2 = **1.**2

0.2 x 2 = **0.**4

...

o processo repete a sequencia de dígitos **0011** infinitamente

Portanto,

$\mathbf{0.1 = (0.0\underline{0011}00110\underline{0011}...)_2}$

Números reais em formato de ponto fixo

- Estes exemplos mostram que num **computador**, onde o espaço para representar um número é finito, alguns **números** terão que ser **arredondados**
- A **forma polinomial** de um **número fracionário** na **base 2** é dada por:

$$X_f = \alpha_1 2^{-1} + \alpha_2 2^{-2} + \alpha_3 2^{-3} + \ldots$$

- Portanto, um número real $X = X_i + X_f$ pode ser representado **em binário** pelo
  - sistema posicional
  - forma polinomial
- Sistema posicional

$$X = (a_n a_{n-1} \ldots a_1 a_0 . \alpha_1 \alpha_2 \alpha_3 \ldots)_2$$

Forma polinomial

$$X = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0 2^0 + \alpha_1 2^{-1} + \alpha_2 2^{-2} + \alpha_3 2^{-3} + \ldots$$

Números reais em formato de ponto flutuante

- Considere-se um computador com um sistema de numeração (binário) com
  - dois dígitos (p = 2)
  - base b = 2
  - expoente e $\in$ { -1, 0, 1, 2 }
- Como os números reais são normalizados ($d_1 \neq 0$) todos eles serão da forma:

$$\pm .10_2 \times 2^e \quad \text{ou} \quad \pm .11_2 \times 2^e, \; e \in \{-1,0,1,2\}.$$

- Considerando a conversão de binário para decimal de um número positivo menor do que 1:

$$.10_2 = 1 \times 2^{-1} + 0 \times 2^{-2} = 1/2 + 0 = 1/2, \text{ e}$$

$$.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 1/2 + 1/4 = 3/4,$$

Números reais em formato de ponto flutuante

- Os únicos números positivos representáveis neste computador são:

$.10_2 \times 2^{-1} = 1/2 \times 1/2 = 1/4$	$.11_2 \times 2^{-1} = 3/4 \times 1/2 = 3/8$
$.10_2 \times 2^{0} = 1/2 \times 1 = 1/2$	$.11_2 \times 2^{0} = 3/4 \times 1 = 3/4$
$.10_2 \times 2^{1} = 1/2 \times 2 = 1$	$.11_2 \times 2^{1} = 3/4 \times 2 = 3/2$
$.10_2 \times 2^{2} = 1/2 \times 4 = 2$	$.11_2 \times 2^{2} = 3/4 \times 4 = 3$

- O **zero** é **representado** de uma forma especial:
  - todos os dígitos da mantissa são nulos ($d_1 = 0$ e $d_2 = 0$)
  - o expoente é nulo ($e = 0$)
  - ou seja,

    $\mathbf{.00_2 \times 2^0}$

- O mais importante a reter sobre reais em formato de ponto flutuante é que são **discretos** e **não contínuos** como na Matemática

Números reais em formato de ponto flutuante

- O conceito de existir sempre um número real entre dois números reais quaisquer não é válido
- As consequências da falha deste conceito podem ser desastrosas (ver exemplo):
  - considere-se as seguintes representações em binário:

    $0.6_{10} = 0.1001100110011..._2$

    $0.7_{10} = 0.1011001100110..._2$

  - se estes dois números forem armazenados naquele hipotético computador (dois dígitos para a mantissa), então eles serão ambos representados por:

    $\mathbf{.10_2 \times 2^0}$

  - ou seja, tanto 0.6_{10} como 0.7_{10} são vistos como 0.5_{10} por aquele computador
  - esta é uma das grandes causas da ocorrência de **erros de arredondamento** nos processos numéricos

Conversão de números em binário para decimal

Números inteiros

- Considere-se a conversão da base binária para a decimal.
- Seja o número N representado na base binária por

$$N = (a_m\ a_{m-1}\ \ldots\ a_1\ a_0)_2$$

- A sua representação na base decimal pode ser obtida simplesmente pelo polinómio

$$N = a_m\ 2^m + a_{m-1}\ 2^{m-1} + \ldots + a_1\ 2 + a_0$$

- A operacionalização deste polinómio pode ser obtida de duas formas:
  - pelo **Algoritmo de Horner**
  - pela **Divisão de Ruffini**

Números inteiros

- Algoritmo de Horner

Seja

$N = (a_m\ a_{m-1}\ \ldots\ a_1\ a_0)_2$

O número N pode ser obtido na base decimal através do cálculo da sequência:

$b_m = \mathbf{a_m}$

$b_{m-1} = \mathbf{a_{m-1}} + 2 \times b_m$

$b_{m-2} = \mathbf{a_{m-2}} + 2 \times b_{m-1}$

... ...

$b_1 = \mathbf{a_1} + 2 \times b_2$

$b_0 = \mathbf{a_0} + 2 \times b_1$

e então,

$\mathbf{N = b_0}$

Números inteiros

- Algoritmo de Horner (exemplo)

Seja o número N = $(11101)_2$ = $(a_4\ a_3\ a_2\ a_1\ a_0)_2$

$b_4 = a_4 = \mathbf{1}$

$b_3 = a_3 + 2 \times b_4 = \mathbf{1} + 2 \times 1 = 3$

$b_2 = a_2 + 2 \times b_3 = \mathbf{1} + 2 \times 3 = 7$

$b_1 = a_1 + 2 \times b_2 = \mathbf{0} + 2 \times 7 = 14$

$b_0 = a_0 + 2 \times b_1 = \mathbf{1} + 2 \times 14 = \mathbf{29}$

portanto,

$\mathbf{(11101)_2 = 29_{10}}$

Números inteiros

- Divisão de Ruffini
  - Equivalente ao algoritmo de Horner
  - Difere apenas na disposição dos coeficientes a_i e b_i
  - Seja

$N = (a_m\ a_{m-1}\ \ldots\ a_1\ a_0)_b$

  - A sua representação na base decimal pode ser obtida da seguinte forma:

	$\mathbf{a_m}$	$\mathbf{a_{m-1}}$	**...**	$\mathbf{a_2}$	$\mathbf{a_1}$	$\mathbf{a_0}$
2		$2 \times b_m$	...	$2 \times b_3$	$2 \times b_2$	$2 \times b_1$
	$\mathbf{b_m}$	$\mathbf{b_{m-1}}$	**...**	$\mathbf{b_2}$	$\mathbf{b_1}$	$\mathbf{b_0}$
		$a_{m-1} + 2 \times b_m$	...	$a_2 + 2 \times b_3$	$a_1 + 2 \times b_2$	$\mathbf{a_0 + 2 \times b_1}$

então,

$\mathbf{N = b_0}$

Números inteiros

- Divisão de Ruffini
  - Exemplo:

Seja o número **N = $(11101)_2$ =** $(a_4\ a_3\ a_2\ a_1\ a_0)_2$

	a_4	**a_3**	**a_2**	**a_1**	**a_0**
	1	1	1	0	1
2		$2 \times b_4$	$2 \times b_3$	$2 \times b_2$	$2 \times b_1$
		2 x 1	2 x 3	2 x 7	2 x 14
	b_4	**b_3**	**b_2**	**b_1**	**b_0**
	1	1 + 2	1 + 6	0 + 14	1 + 28
		3	7	14	**29**

Portanto,

$(11101)_2 = 29_{10}$

Números reais

- Considere-se um número fracionário com representação finita na base binária:

$$\mathbf{X_f = (0.\alpha_1\alpha_2...\alpha_n)_2}$$

- O valor de X_f na base decimal será dado por

$$\mathbf{X_f = \alpha_1\, 2^{-1} + \alpha_2\, 2^{-2} + ... + \alpha_n\, 2^{-n}}$$

- Esta soma pode ser calculada de duas formas
  - **diretamente**, ou
  - utilizando **Algoritmo de Horner** e a **Divisão de Ruffini** com algumas modificações

Números reais

– Algoritmo de Horner (caso de um número fracionário na base 2)

Seja $N = (0.\alpha_1\alpha_2\ldots\alpha_n)_2$

$b_n = \boldsymbol{\alpha_n}$

$b_{n-1} = \boldsymbol{\alpha_{n-1}} + (1/2) \times b_n$

$b_{n-2} = \boldsymbol{\alpha_{n-2}} + (1/2) \times b_{n-1}$

... ...

$b_2 = \boldsymbol{\alpha_2} + (1/2) \times b_3$

$b_1 = \boldsymbol{\alpha_1} + (1/2) \times b_2$

$b_0 = (1/2) \times b_1$

então,

$\mathbf{N = b_0}$

Números reais

– Algoritmo de Horner (exemplo)

Converter o número $\mathbf{N = (0.10111)_2} = (0.\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4\ \alpha_5)_2$

$b_5 = \mathbf{\alpha_5} = 1$

$b_4 = \mathbf{\alpha_4} + (1/2) \times b_5 = \mathbf{1} + (1/2) \times 1 = 3/2$

$b_3 = \mathbf{\alpha_3} + (1/2) \times b_4 = \mathbf{1} + (1/2) \times (3/2) = 7/4$

$b_2 = \mathbf{\alpha_2} + (1/2) \times b_3 = \mathbf{0} + (1/2) \times (7/4) = 7/8$

$b_1 = \mathbf{\alpha_1} + (1/2) \times b_2 = \mathbf{1} + (1/2) \times (7/8) = 23/16$

$b_0 = (1/2) \times b_1 = (1/2) \times (23/16) = 23/32$

então,

$\mathbf{(0.10111)_2 = 23/32 = 0.71875}$

Números reais

- Divisão de Ruffini (caso de um número fracionário na base 2)

Seja $N = (0.\alpha_1\alpha_2\ldots\alpha_n)_2$

	α_n	α_{n-1}	...	α_2	α_1	
1/2		$1/2 \times b_m$	...	$1/2 \times b_3$	$1/2 \times b_2$	$1/2 \times b_1$
	b_n	b_{n-1}	...	b_2	b_1	b_0
		$\alpha_{n-1} + 1/2 \times b_m$		$\alpha_2 + 1/2 \times b_3$	$\alpha_1 + 1/2 \times b_2$	$1/2 \times b_1$

Então,

$N = b_0$

Números reais

- Divisão de Ruffini (exemplo)

Converter o número N = $(0.10111)_2$ = $(0.\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4\ \alpha_5)_2$

	α_5	α_4	α_3	α_2	α_1	
	1	1	1	0	1	
1/2		1/2 x b_5	1/2 x b_4	1/2 x b_3	1/2 x b_2	1/2 x b_1
		1/2 x 1	1/2 x 3/2	1/2 x 7/4	1/2 x 7/8	1/2 x 23/16
	b_5	b_4	b_3	b_2	b_1	b_0
	1	1 + 1/2	1 + 3/4	0 + 7/8	1 + 7/16	23/32
	1	3/2	7/4	7/8	23/16	**23/32**

Então,

$(0.10111)_2$ = 23/32 = 0.71875

Números reais

- Número binário infinito
  - Uma outra situação que pode ocorrer é quando o **número binário** for **infinito**:

$$X_f = \left(0.\alpha_1\alpha_2\ldots\alpha_n\,\overline{\beta_1\beta_2\ldots\beta_m}\right)_2$$

em que

$\overline{\beta_1\beta_2\ldots\beta_m}$ indica que a sequência de dígitos $\beta_1\beta_2\ldots\beta_m$ se repete infinitamente

  - Em geral, um número fracionário tem representação infinita periódica na base 2 da seguinte forma:

$$X_f = \left(\alpha_1 2^{-1} + \alpha_2 2^{-2} + \ldots + \alpha_n 2^{-n}\right) + \left(\beta_1 2^{-1} + \beta_2 2^{-2} + \ldots + \beta_m 2^{-m}\right)\frac{2^{m-n}}{2^m - 1}$$

onde as expressões entre parênteses podem ser calculadas

    - diretamente ou
    - utilizando qualquer um dos métodos descritos anteriormente

Aritmética de ponto flutuante

Operações com números

- **OVERFLOW**
  - ocorre quando uma operação aritmética resultar num **número** que seja **maior**, em **valor absoluto**, que o **maior número representável**.
- **UNDERFLOW**
  - ocorre quando uma operação aritmética resultar num **número** que seja **menor**, em **valor absoluto**, que o **menor número representável diferente de zero**.

Operações com números em decimal

- Seja um hipotético computador com
  - dois dígitos da mantissa (p = 2)
  - base b = 10
  - expoente e $\in$ {-5, …, 5}
  - formato: $\pm .\mathbf{d_1 d_2 \times 10^e}$
- Quando dois números são **somados** ou **subtraídos**, o processo é o seguinte:
  - os dígitos do número de menor expoente são deslocados para alinhar as casas decimais
  - o resultado é então normalizado (o expoente é ajustado de forma a normalizar a mantissa, $d_1 \neq 0$)
  - por fim, o resultado é arredondado para dois dígitos para caber na mantissa (p = 2).

Operações com números em decimal

- Exemplo 1: **4.32 + 0.064**
  - os números são armazenados no formato especificado,
  - as casas decimais são alinhadas,
  - a operação de adição é efetuada;
  - por fim, o resultado é então normalizado e arredondado para dois dígitos:

$$
\begin{aligned}
4.32 + 0.064 = .43 \times 10^{1} + .64 \times 10^{-1} &= .43 \times 10^{1} \\
&+ .0064 \times 10^{1} \\
&= .4364 \times 10^{1} \\
&\rightarrow .44 \times 10^{1}
\end{aligned}
$$

  - O resultado da adição é **4.4** em vez de **4.384**

Operações com números em decimal

- Exemplo 2: **372 – 371**
  - os números são armazenados no formato especificado
  - as casas decimais são alinhadas
  - a operação de adição é efetuada
  - o resultado é então normalizado e arredondado para dois dígitos:

$$
\begin{aligned}
372 - 371 &= .37 \times 10^3 - .37 \times 10^3 = .37 \times 10^3 \\
&\quad\; - .37 \times 10^3 \\
&= .00 \times 10^3 \\
&\rightarrow .00 \times 10^0
\end{aligned}
$$

  - O resultado da subtração é **0** em vez de **1**

Operações com números em decimal

- Exemplo 3: **1234 x 0.016**
  - os números são armazenados no formato especificado,
  - a multiplicação é efetuada utilizando 2p = 4 dígitos na mantissa;
  - o resultado é então normalizado e arredondado para dois dígitos:

$$
\begin{aligned}
1234 \times 0.016 = .12 \times 10^{4} \times .16 \times 10^{-1} &= .12 \times 10^{4} \\
&\quad \times .16 \times 10^{-1} \\
&= .0192 \times 10^{3} \\
&\rightarrow .19 \times 10^{2}
\end{aligned}
$$

  - O resultado da multiplicação é **19** em vez de **19.744**

Operações com números em decimal

- A perda de precisão quando dois números aproximadamente iguais são subtraídos é das maiores fontes de erro nas operações de ponto flutuante.
- Uma das causas de se cometer erros quando se usa um computador deve-se à conversão de base, pois um número
  - é fornecido ao computador em decimal (base 10) e
  - é armazenado em binário (base 2).
- Enquanto que
  - para um número inteiro, a representação é exata; por exemplo, $44_{10} = 101100_2$
  - para um número real com parte fracionária, a representação a parte fracionária tem que ser arredondada para ser armazenada em formato de ponto flutuante

Operações com números em binário

Adição binária

- Uma adição no sistema binário é realizada da mesma forma que a adição no sistema decimal
- A adição é realizada de acordo com as seguintes regras (considerando os dois operandos positivos):

  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 0 (e "vai 1" para o dígito de ordem superior)
  1 + 1 + 1 = 1 (e "vai 1" para o dígito de ordem superior)

- Para somar números com mais de 2 algarismos
  - utiliza-se o mesmo processo de transporte para a coluna posterior, tal como na adição decimal

Adição binária

- Exemplo:

$101_2 + 011_2 = 1000_2$ ($5_{10} + 3_{10} = 8_{10}$)

[1]	[1]	[1]	
	1	0	1
+	0	1	1
1	0	0	0

- Quando um dos operandos são números negativos, o processo é o seguinte:
  - dois operandos negativos:
    - adicionam-se os dois números considerando o valor absoluto de cada um deles e
    - atribui-se o sinal de negativo
  - um deles é negativo:
    - verifica-se qual dos dois números tem maior valor absoluto
    - subtraí-se o menor valor absoluto ao maior, e
    - atribui-se o sinal do maior em valor absoluto

Subtração binária

- A subtração é análoga à adição e é realizada de acordo com as seguintes regras:

  0 - 0 = 0
  0 - 1 = 1 (e "pede emprestado 1" para o dígito de ordem superior)
  1 - 0 = 1
  1 - 1 = 0

- A operação 0 - 1 resulta em 1, mas com o transporte de 1 para a coluna à esquerda, que deve ser
  - acumulado ao **subtraendo** e, por consequência,
  - subtraído do **minuendo**
  - nota: em a-b, **a** o minuendo e **b** é o subtraendo

Subtração binária

- Exemplo:

$101_2 - 011_2 = 010_2$ ($5_{10} - 3_{10} = 2_{10}$)

	[1]		
	1	0	1
–	0	1	1
	0	1	0

Representação de números em computadores digitais

Representação de inteiros e reais

- As **representações** apresentadas antes **não são suficientes**, pois é necessário distinguir-se, por exemplo, o **sinal do número**
- Para se representar o sinal **+** ou **-** na memória de um computador, acrescenta-se um **bit** ao número para representar o sinal — o denominado de **bit de sinal**
- Representação de inteiros:
  - em Sinal-Módulo (ou Sinal-Magnitude)
  - em Complemento à base (a **b** e a **b-1**)
- Representação de números reais:
  - em formato de ponto flutuante normalizado

Representação de números inteiros em Sinal-Módulo

- A representação mais direta é a em **Sinal-Módulo**, onde
  - o valor absoluto do número é obtido diretamente a partir dos algoritmos apresentados antes, e
  - o sinal é representado por um dígito adicional colocado à esquerda do número
- Na **representação binária** o **bit de sinal** é o bit mais significativo (posicionado mais à esquerda na sequência)
- Supondo que o computador dispõe de **q** dígitos para a representação, um inteiro em binário é representado no computador através da seguinte sequência de dígitos (**palavra**):

  $\mathbf{a_{q-1}\ a_{q-2}\ \ldots\ a_1\ a_0}$

  em que

  $a_0, a_1, \ldots, a_{q-2} \in \{ 0, 1 \}$ e

  $a_{q-1} \in \{ 0, 1 \}$ representa o **sinal do número** (0 para "+" e 1 para "−")

Representação de números inteiros em Sinal-Módulo

- A conversão do número em bininternamente representado por $a_{q-1}\ a_{q-2}\ ...a_1\ a_0$ para o sistema decimal, é feita através de uma fórmula semelhante à forma polinomial:

$$N = (-1)^{a_{q-1}} \times \sum_{k=0}^{q-2} \left(a_k \times 2^k \right),$$

em que,

N o número inteiro na base decimal

q-2 é o índice do dígito mais à esquerda que representa o valor absoluto de **N**

$\mathbf{a_k}$ um dígito válido na representação ($a_k \in \{ 0, 1 \}$), k = 0, 1, …, q-2

$\mathbf{a_{q-1}} \in \{ 0, 1 \}$ e representa o bit de sinal

- Os valores para as quantidades expressas dependem da arquitetura e do compilador utilizado

Representação de números inteiros em Sinal-Módulo

- Por exemplo, um dado compilador possui 4 modelos de representação de inteiros com **1, 2, 4 e 8 bytes**, também denominados de **espécies**
- Para o caso binário, os maiores números em valor absoluto que podem ser representados internamente para cada espécie $\mathbf{N_{max}}(p)$ (p = 1, 2, 4, 8) são:

$$N_{max}(p) = 2^0 + 2^1 + 2^2 + \ldots + 2^{8p-2} = 2^{8p-1} - 1 ,$$

p = 1 (1 byte = 8 bites), em que os últimos 7 bites são 7 **1**'s (**1111111**)

$\mathbf{N_{max}} = \mathbf{1111111}_2 = \mathbf{127}_{10}$ ($\mathbf{1}x2^0 + \mathbf{1}x2^1 + \mathbf{1}x2^2 + \mathbf{1}x2^3 + \mathbf{1}x2^4 + \mathbf{1}x2^5 + \mathbf{1}x2^6$)

p = 2 (2 bytes = 16 bites), em que os últimos 15 bites são 15 **1**'s

$\mathbf{N_{max}} = \mathbf{1\ldots1}_2 = \mathbf{32767}_{10}$ ($\mathbf{1}x2^0 + \mathbf{1}x2^1 + \mathbf{1}x2^2 + \cdots + \mathbf{1}x2^{12} + \mathbf{1}x2^{13} + \mathbf{1}x2^{14}$)

p = 4 (4 bytes = 32 bites), em que os últimos 31 bites são 31 **1**'s

$\mathbf{N_{max}} = \mathbf{1\ldots1}_2 = \mathbf{2147483647}_{10}$ ($\mathbf{1}x2^0 + \mathbf{1}x2^1 + \mathbf{1}x2^2 + \cdots + \mathbf{1}x2^{29} + \mathbf{1}x2^{30}$)

p = 8 (8 bytes = 64 bites), em que os últimos 63 bites são 63 **1**'s

$\mathbf{N_{max}} = \mathbf{1\ldots1}_2 = \mathbf{9223372036854775807}_{10}$ ($\mathbf{1}x2^0 + \mathbf{1}x2^1 + \cdots + \mathbf{1}x2^{61} + \mathbf{1}x2^{62}$)

Representação de números inteiros em complemento a b-1

- A **representação** de números inteiros **positivos** em **complemento a b-1** é idêntica à representação em **Sinal-Módulo**
- A representação dos números **negativos** é obtida efetuando-se:

  (**b - 1**) menos cada algarismo do número
- Por exemplo, calcular o complemento a **b-1** do número **-297_{10}**
  - b = 10, logo complemento a **b-1** será **complemento a 9**
  - como 999 – 297 = 702, o complemento a 9 do número -297 é 702

Representação de números inteiros em complemento a b-1

- Para se obter o **complemento a b−1** de um número binário,
  - deve-se subtrair cada algarismo de 1 (b-1 = 1)
  - como se trata de binários, basta inverter todos os seus bits
- Por exemplo:
  - o complemento a 1 (C1) do número 0011_2 (usando 4 dígitos) é 1100_2:

    $1111_2 - 0011_2 = 1100_2$
- Quantidade de números inteiros diferentes que se podem representar com n posições num sistema de base b: $\mathbf{b^n}$
  - Por exemplo, na base 2, podem-se representar os seguintes números:
    - $2^1 = 2$ números com um dígito (0, 1),
    - $2^2 = 4$ números com dois dígitos (00, 01, 10, 11),
    - $2^3 = 8$ números com três dígitos (000, 001, 010, 011, 100, 101, 110, 111),
    - ...

Representação de números inteiros em complemento a b-1

- Representação dos números binários de 4 dígitos em **complemento a 1** (**C1**):

Decimal (positivo)	Binário em C1 (igual a sinal-módulo)	Decimal (negativo)	Binário em C1
0	0000	0	1111
1	0001	-1	1110
2	0010	-2	1101
3	0011	-3	1100
4	0100	-4	1011
5	0101	-5	1010
6	0110	-6	1001
7	0111	-7	1000

- Note-se que há 2 representações para o zero (0): **0000** e **1111**

Representação de números inteiros em complemento a b-1

- A representação na base 10 com 3 dígitos tem 10^3 representações (**000** ... **999**)
  - representando os números de -499 a -1 com 500 a 998;
  - representando os números de +1 a +499 com 1 a 499;
  - representando o zero (0) com 000 ou 999.
- Faixa de representação em C1 dos números binários com **n** dígitos:
  - menor inteiro negativo: $-(2^{n-1} - 1)$
  - maior inteiro positivo: $2^{n-1} - 1$

Representação de números inteiros em complemento a b-1

- Na aritmética em **complemento a b-1**
  - basta somar os números
  - sendo que um número negativo será representado por seu **complemento a b–1**
- Por exemplo, a soma em decimal de 123 com -418 é:
  - **em sinal-módulo**

    -418 + 123 = -295
  - **em complemento a 9** (b–1 = 10-1 = 9)

    -418 é representado por 999 - 418 = 581

    581 + 123 = 704

    999 - 704 = 295, em que 704 é o C9 de -295 (704 está na faixa negativa)

Representação de números inteiros em complemento a b

- A representação dos números negativos em **complemento a b** é obtida
  - subtraindo-se da base **b** cada algarismo do número
- Por exemplo,
  - na base b = 10 com 3 dígitos: 1000 – x
- Uma forma alternativa é
  - subtrair cada algarismo de (b–1), isto é,
    - calcular o **complemento a b-1** e depois
    - somar 1 ao resultado

Representação de números inteiros em complemento a b

- Exemplo 1

Calcular em **complemento a b** (b = 10) do número 297_{10} (com 3 dígitos):

- em complemento a 10:

  1000 – 297 = 703

- representar o número em complemento a 9 e somar 1 ao resultado:

  999 – 297 = 702 $\Rightarrow$ 702 + 1 = 703

- Exemplo 2

Calcular o **complemento a 2** (b = 2) do número $\mathbf{0011}_2$ (com 4 dígitos):

- usando complemento em 2:

  10000 – 0011 = 1101

- representar o número em complemento em 1 e somar 1:

  1111 – 0011 = 1100 $\Rightarrow$ 1100 + 0001 = 1101

Representação de números inteiros em complemento a b

- Representação dos números binários de 4 dígitos em **complemento a 2** (**C2**):

Decimal (positivo)	Binário em C2 (igual a sinal-módulo)	Decimal (negativo)	Binário em C2
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

Representação de números inteiros em complemento a b

- Comparando com a tabela anterior (para C1),
  - os números positivos têm a mesma representação de C1
  - o zero passou a ter apenas uma representação
- Esta pequena diferença permite representar mais um número
  - neste caso, mais um negativo, o **-8**
- A faixa de representação em complemento a 2 dos números binários com **n** dígitos é a seguinte:
  - menor inteiro negativo: -2^{n-1}
  - maior inteiro positivo: $2^{n-1} - 1$

Representação de números reais

- Representação de números reais em computadores denomina-se por
  - representação em formato de ponto flutuante normalizado.
- Nesta representação um número é representado internamente através de uma notação científica:

  - um bit de sinal **s** (interpretado como positivo ou negativo)
  - um expoente inteiro exato **e**
  - uma mantissa inteira positiva **M**

  sendo que apenas um número limitado de dígitos é permitido para **e** e **M**

Representação de números reais

- Tomando todas estas quantidades juntas, estas representam o número

$$x = s \times (0.d_1 d_2 \ldots d_n) \times b^e,$$

em que,

- **s** é o sinal do número,
- os dígitos **d_1, d_2, ..., d_n**
  - formam a mantissa **M** = $0.d_1d_2...d_n$ e
  - são limitados pela base **b** ($0 \leq d_i \leq b-1$, i = 1, ..., n e $d_1 \neq 0$),
- o expoente **e** é limitado ao conjunto { e_{min}, ..., e_{max} },
- $n \geq 1$
  - denomina-se de número de dígitos do sistema e
  - define o tamanho da mantissa M

- O valor zero não pode ser normalizado e tem representação especial:
  - com mantissa nula (todos dígitos iguais a zero) e
  - expoente o menor possível

Representação de números reais

- O conjunto formado por zero e todos os números em formato de ponto flutuante
  - chamado-se **Sistema de Ponto Flutuante na base b** com **n** algarismos significativos
  - denota-se por **F(b, n, e_{min}, e_{max})**
- Um computador só pode representar os valores de **e** e **M** com dígitos da base **b**
- Um computador digital (b = 2) dispõe sempre de um tamanho de palavra finito
- Para um dado tipo de números reais, é sempre fixo o **número total de bits** que podem ser utilizados para representar
  - o sinal **s** (1 bit)
  - o expoente
  - a mantissa
- Exemplo de um número real de precisão simples que é representado por 32 bits:
  - **1 bit** é utilizado para representar o **sinal**
  - **8 bits** são utilizados para representar o **expoente**
  - os restantes **23 bits** para representar a **mantissa**

Representação de números reais

- Um número real será representado na memória do computador como

$$x = s\, e_7 e_6 \ldots e_1 e_0\, d_1 d_2 \ldots d_{22} d_{23}$$

em que

$$s, e_0, \ldots, e_7, d_1, \ldots, d_{23} \in \{0, 1\}.$$

- Exemplo:

Considere dois números binários com 8 algarismos significativos em F(2, 8, -3, 4):

$n_1 = 0\ 010\ 11100110_2 \Rightarrow (-1)^0 \times 2^2 \times (0.11100110) = 3.59375_{10}$

$n_2 = 0\ 010\ 11100111_2 \Rightarrow (-1)^0 \times 2^2 \times (0.11100111) = 3.609375_{10}$

- Observe-se que, no sistema de representação utilizado,
  - n_1 e n_2 são dois números consecutivos (não há representação de número entre eles)
- Portanto, por exemplo, o número 3.60000
  - não tem representação exata neste sistema, sendo representada por n_1 ou n_2
  - o que vai gerar um erro, denominado **Erro de Arredondamento**

Representação de números reais

- Portanto,
  - os números reais podem ser representados por uma reta contínua
  - em formato de ponto flutuante só se podem representar pontos discretos da reta real
- Conversão de um número **x** representado na base **b** para a base decimal:

$$x = (-1)^s \times b^e \times \sum_{k=1}^{n} \left(d_k \times b^{-k} \right)$$

- A tabela mostra alguns números para um computador que usa o padrão IEEE 754

Espécie	REAL (4)	REAL (8)	REAL (10)
n	23	52	64
e_{min}	-126	-1022	-16382
e_{max}	127	1023	16383
X_{min}	$1.1754944 \times 10^{-38}$	$2.225073858507201 \times 10^{-308}$	$3.3621031431120935 06... \times 10^{-4932}$
X_{max}	3.4028235×10^{38}	$1.797693134862316 \times 10^{308}$	$1.1897314953572317 65... \times 10^{4932}$
X_{eps}	1.1920929×10^{-7}	$2.220446049250313 \times 10^{-16}$	$1.9259299443872358 53... \times 10^{-34}$

Representação de números reais no sistema normalizado

- No **sistema binário** (base = 2) **normalizado** ($d_1 \neq 0$)
  - o primeiro dígito da mantissa no sistema normalizado é sempre igual a 1,
    - e por esta razão não é armazenado (é o denominado **bit escondido**)
  - esta normalização permite um ganho na precisão,
    - pois pode-se considerar que a mantissa é armazenada em 24 bits

Representação de números reais no sistema normalizado

- Os números do sistema F(b, n, e_{min}, e_{max}) têm as seguintes características (1):
  - o menor número positivo que pode ser representado neste sistema é

    $x_{min} = 0.1 \times b^{e_{min}}$

    o que significa que qualquer número real **x** tal que

    $-x_{min} < \mathbf{x} < x_{min}$

    não poderá ser representado pelo computador — ocorre **underflow**
  - Os compiladores, quando detetam **underflow**, são instruídos normalmente para
    - terminar o processamento neste ponto, disparando uma mensagem de erro, ou
    - seguir o processamento arredondando **x** para **0**

Representação de números reais

- Os números do sistema F(b, n, e_{min}, e_{max}) têm as seguintes características (2):
  - O maior número positivo que pode ser representado neste sistema é

$$x_{max} = 0.\underbrace{(b-1)(b-1)\ldots(b-1)}_{\text{n vezes}} \times b^{e_{max}} = (b-1) \times \left(\sum_{k=1}^{n} b^{-k} \right) \times b^{e_{max}} = \left(1 - b^{-n}\right) b^{e_{max}}$$

  Isto significa que qualquer número **x** tal que

$$\mathbf{x} < -x_{max} \quad \text{ou} \quad \mathbf{x} > x_{max}$$

  não poderá ser representado pelo computador — ocorre **overflow**.
  - Os compiladores quando detetam **overflow**, são instruídos normalmente para
    - parar o processamento do programa emitindo uma mensagem de erro, ou
    - continuar o processamento atribuindo a **x** o valor simbólico **-Infinito** ou **Infinito**

Representação de números reais

- Os números do sistema F(b, n, e_{min}, e_{max}) têm as seguintes características (3):
  - O maior número que pode ser somado ou subtraído a **1.0**, em que o resultado permanece inalterado (i.é, a diferença entre **1.0** e o número que lhe sucede em F), é

$$x_{eps} = 0.\underbrace{10\ldots01}_{\text{n vezes}} \times b^1 - 0.\underbrace{10\ldots00}_{\text{n vezes}} \times b^1 = b^{1-n}$$

    em que $\mathbf{x_{eps}}$ é denominada de **epsilon da máquina**, ϵ, ou de **precisão da máquina**
  - O epsilon da máquina, ϵ, também pode ser definido como o menor número em formato de ponto flutuante, tal que:

$$1 + \epsilon > 1$$

  - Esta quantidade é da maior **importância** na análise de **erros de arredondamento**

Representação de números reais

- Duma forma mais geral, upara m número em formato de ponto flutuante $x \in F$ def

ulp(x) = $(0.00...01)_b \times b^e = b^{-n} \times b^e = \varepsilon \times b^e$

em que **ulp** é a abreviatura para **unit in the last place**

- se $x > 0$, então **ulp(x)** é a distância entre **x** e o número que lhe sucede em F
- se $x < 0$, então **ulp(x)** é a distância entre **x** e o número que o antecede em F

- Apenas um conjunto finito R_F de números racionais podem ser representados na forma apresentada:
  - estes números denominam-se de **números de ponto flutuante**.
- Numa representação normalizada ($d_1 \neq 0$), a quantidade de números racionais que este conjunto contém é precisamente

$2 (b - 1) b^{n-1} (e_{max} - e_{min} + 1) + 1$

Representação de números reais

- Exemplo

Considere-se o sistema de representação numérica F(2, 4, -5, 6)

Menor número positivo possível:

$\mathbf{x_{min}} = (0.1000)_2 \times 2^{-5} = 2^{-5-1} = 2^{-6} = \mathbf{1/64} = \mathbf{0.015625}$

assim, a região de **underflow** consiste no intervalo

$-0.015625 < \mathbf{x} < 0.015625$

Maior número positivo possível:

$\mathbf{x_{max}} = (0.1111)_2 \times 2^{6} = (1 - 2^{-4}) \times 2^{6} = \mathbf{60}$

assim, as regiões de overflow consistem nos intervalos

$\mathbf{x} < -60$ e $\mathbf{x} > 60$

Maior número que pode ser somado/subtraído a **1.0** mantendo o resultado inalterado:

$\mathbf{x_{eps}} = 2^{1-4} = 2^{-3} = \mathbf{1/8} = \mathbf{0.125}$

Número de elementos em R_F:

$2 \times 1 \times (6 + 5 + 1) \times 2^{4-1} + 1 = \mathbf{193}$