Software Básico

Representação de Dados
Ponto Flutuante

INSTITUTO DE INFORMÁTICA - UFG

Reconhecimento

- Material produzido por:
  - Noemi Rodriguez – PUC-Rio
  - Ana Lúcia de Moura – PUC-Rio
- Adaptação
  - Bruno Silvestre – UFG

Representação em Ponto Flutuante

- Tipos C: float e double
- Codifica números racionais na forma: $x * 2^y$
  - Adequado para valores grandes ou muito pequenos
- Aproximação de números grandes
- Padrão IEEE-754 (~1985)
  - Representação de número de ponto flutuante e suas operações
  - Portabilidade

Ponto Flutuante

- Base 10
  - Números racionais p/q → *mantissa* x 10^{exp}
  - Ponto (vírgula) “flutua”, compensado pelo expoente

$$129 = 12.9 \times 10^1 = 1.29 \times 10^2$$

Normalização

- Múltiplas representações possíveis

  $129 \times 10^0 = 12.9 \times 10^1 = 1.29 \times 10^2 = 0.129 \times 10^3 = 1290 \times 10^{-1}$
- Escolha de uma representação padrão
  - **Representação normalizada**
  - Para base 10:

    1 <= mantissa < 10

INSTITUTO DE INFORMÁTICA - UFG

Normalização

- Múltiplas representações possíveis

  $129 \times 10^0 = 12.9 \times 10^1 = 1.29 \times 10^2 = 0.129 \times 10^3 = 1290 \times 10^{-1}$
- Escolha de uma representação padrão
  - **Representação normalizada**
  - Para base 10:

    $1 <= \boxed{\text{mantissa}} < 10$ Base

    Exemplo: $\boxed{1.29} \times 10^2$

Conversões

Binário ↔ Decimal

Números Fracionários

- Para base 10

$12.34_{10} = \underbrace{1 * 10^{1} + 2 * 10^{0}}_{\text{Potências POSITIVAS}} + \underbrace{3 * 10^{-1} + 4 * 10^{-2}}_{\text{Potências NEGATIVAS}}$

Números Fracionários

- Para base 10

$12.34_{10} = 1 * 10^1 + 2 * 10^0 + 3 * 10^{-1} + 4 * 10^{-2}$

Potências POSITIVAS . Potências NEGATIVAS

- Para base 2

$101.11_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2}$

Potências POSITIVAS . Potências NEGATIVAS

$101.11_2 = 4 + 0 + 1 + ½ + ¼ = 5.75_{10}$

Conversão: Binário → Decimal

$1.001_2 = 1 * 2^0 + 1 * 2^{-3}$
$= 1 + 1/8$
$= 1.125_{10}$

$101.111_2 = 1 * 2^2 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3}$
$= 4 + 1 + 1/2 + 1/4 + 1/8$
$= 5.875_{10}$

$11.0011_2 = 1 * 2^1 + 1 * 2^0 + 1 * 2^{-3} + 1 * 2^{-4}$
$= 2 + 1 + 1/8 + 1/16$
$= 3.1875_{10}$

Conversão: Decimal → Binário

$$3\,.\,125_{10}$$

$$\underbrace{3_{10}} + \underbrace{0.125_{10}}$$

$$b...bbbbb_2 \quad . \quad bbbbb...b_2$$

Conversão: Decimal → Binário

$$3\,.\,125_{10}$$

$$\underbrace{3_{10}}_{\text{Divisões sucessivas por 2}} + \underbrace{0.125_{10}}_{?????}$$

$$b * 2^n + \ldots + b * 2^1 + b * 2^0 \quad . \quad b * 2^{-1} + b * 2^{-2} + \ldots + b * 2^{-m}$$

Conversão: Decimal → Binário

$$0.125 = 0.125 \times (\frac{2}{2}) = \frac{0.250}{2} = \frac{0+0.250}{2} = \frac{0}{2} + \frac{0.250}{2}$$

Conversão: Decimal → Binário

$$0.125 = 0.125 \times (\frac{2}{2}) = \frac{0.250}{2} = \frac{0+0.250}{2} = \frac{0}{2} + \frac{0.250}{2}$$

$$0.125 = \frac{0}{2} + [(\frac{0.250}{2}) \times (\frac{2}{2})] = \frac{0}{2} + \frac{0.500}{2^2} = \frac{0}{2} + \frac{0}{2^2} + \frac{0.500}{2^2}$$

Conversão: Decimal → Binário

$$0.125 = 0.125 \times (\frac{2}{2}) = \frac{0.250}{2} = \frac{0+0.250}{2} = \frac{0}{2} + \frac{0.250}{2}$$

$$0.125 = \frac{0}{2} + [(\frac{0.250}{2}) \times (\frac{2}{2})] = \frac{0}{2} + \frac{0.500}{2^2} = \frac{0}{2} + \frac{0}{2^2} + \frac{0.500}{2^2}$$

$$0.125 = \frac{0}{2} + \frac{0}{2^2} + [(\frac{0.500}{2^2}) \times (\frac{2}{2})] = \frac{0}{2} + \frac{0}{2^2} + \frac{1.0}{2^3} = \frac{0}{2} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{0}{2^3}$$

INSTITUTO DE INFORMÁTICA - UFG

Conversão: Decimal → Binário

$$0.125 = 0.125 \times (\frac{2}{2}) = \frac{0.250}{2} = \frac{0+0.250}{2} = \frac{0}{2} + \frac{0.250}{2}$$

$$0.125 = \frac{0}{2} + [(\frac{0.250}{2}) \times (\frac{2}{2})] = \frac{0}{2} + \frac{0.500}{2^2} = \frac{0}{2} + \frac{0}{2^2} + \frac{0.500}{2^2}$$

$$0.125 = \frac{0}{2} + \frac{0}{2^2} + [(\frac{0.500}{2^2}) \times (\frac{2}{2})] = \frac{0}{2} + \frac{0}{2^2} + \frac{1.0}{2^3} = \frac{0}{2} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{0}{2^3}$$

$$0.125 = \frac{0}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} = 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

Algoritmo de Conversão

0.625 * 2 = 1 . 250

0.250 * 2 = 0 . 500

0.500 * 2 = 1 . 000

0.000 * 2 = 0 . 000

Algoritmo de Conversão

0.625 * 2 = 1 . 250

0.250 * 2 = 0 . 500

0.500 * 2 = 1 . 000

0.000 * 2 = 0 . 000

$0.625_{10} = 1\ 0\ 1\ 0$

Limitação da Representação

Limitações da Representação

- Algumas frações não possuem representação precisa
  - Exemplo: 1/3 ou 5/7
- Se tivermos espaço infinito, isso não é um problema
  - 1/3 = 0,333333333333333333333333…

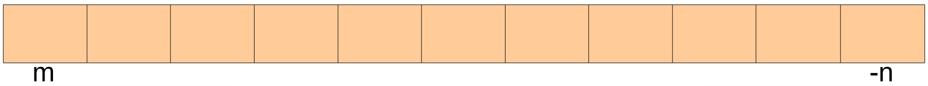
Limitações da Representação

- A notação decimal representa números que podem ser escritos como

$$\sum_{i=-n}^{m} d_i \, . \, 10^i$$

- Espaço limitado para a representação

N = d x 10^m + … + d x 10^1 + d x 10^0 + d x 10^{-1} + d x 10^{-2} + … + d x 10^{-n}

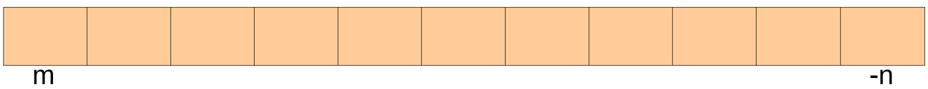


Limitações da Representação

- A notação binária representa números que podem ser escritos como

$$\sum_{i=-n}^{m} b_i \, . \, 2^i$$

- Então, quando não possui representação precisa, os números devem ser aproximados



Exemplo de Sequência Infinita

0.3 * 2 = 0. 6

0.6 * 2 = 1. 2

0.2 * 2 = 0. 4

0.4 * 2 = 0. 8

0.8 * 2 = 1. 6

0.6 * 2 = 1. 2

...

$0.3_{10} = 010011...$

Quanto mais dígitos, maior é a precisão

Representação IEEE 754

Representação IEEE 754

- Representa números na forma: $x * 2^y$
  - Com um par adequado (x,y)
- Existem várias representações possíveis

  $1.5_{10} \rightarrow 1.1 \times 2^0 = 11.0 \times 2^{-1} = 0.11 \times 2^1$
- **Normalização**

  1 <= mantissa < 2

Representação IEEE 754

- Representa números na forma: $x * 2^y$
  - Com um par adequado (x,y)
- Existem várias representações possíveis

  $1.5_{10} \rightarrow 1.1 \times 2^0 = 11.0 \times 2^{-1} = 0.11 \times 2^1$
- **Normalização**

  $1 <= \boxed{\text{mantissa}} < 2$

  $\boxed{1.1} \times 2^0$

  Base

Codificação: Forma Numérica

$$\boxed{FP = (-1)^s * M * 2^E}$$

- Bit de sinal "s" determina se número é positivo ou negativo
- Mantissa "M" é um valor fracionário 1 <= M < 2
- Expoente "E" indica potência positiva ou negativa de 2

Codificação

$$\boxed{FP = (-1)^s * M * 2^E}$$

s	exp	frac

- Campo "s": 0 → positivo, 1 → negativo
- Campo "exp": codificação de "E"
- Campo "frac": codificação de "M"

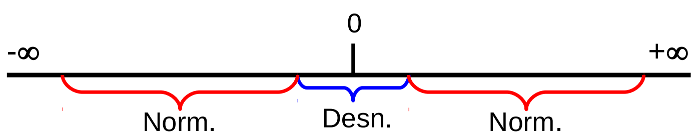
Padrão IEEE 754: Precisão

- float (32 bits): valores de 2^{-126} até 2^{127}
  - s: 1 bit
  - exp: 8 bits
  - fraq: 23 bits
- double (64 bits): valores de 2^{-1022} até 2^{1023}
  - s: 1 bit
  - exp: 11 bits
  - fraq: 52 bits

s	exp	frac

Categorias de Ponto Flutuante

Normalizado	s	exp ≠ 0’s e ≠ 1’s	frac
Desnormalizado	s	0000000..000	frac
Infinito	s	1111111...111	00000000000...000
NaN	s	1111111...111	≠ 0



Valores Normalizados

s	exp	frac

$FP = (-1)^s * M * 2^E$

- Sinal é apenas 1 bit
  - 0 → positivo
  - 1 → negativo

Valores Normalizados

s	exp	frac

$FP = (-1)^s * M * 2^E$

- M está normalizado (1 <= M < 2)

  M = 1.bbbbb...b
- O "1" inicial fica implícito, guardamos o restante

  frac = bbbbb...b

Valores Normalizados

s	exp	frac

$FP = (-1)^s * M * 2^E$

- M está normalizado (1 <= M < 2)

  $M = 1.\boxed{bbbbb...b}$

- O "1" inicial fica implícito, guardamos o restante

  $frac = \boxed{bbbbb...b}$

Valores Normalizados

s	exp ≠ 0's e ≠ 1's	frac

$FP = (-1)^s * M * 2^E$

- *exp* = E + *bias*
  - Representação em excesso de 2^n
  - *exp* é um valor sem sinal
  - *bias* = $2^{n-1} - 1$
- float (precisão simples)
  - bias = 127 $(2^{8-1} – 1 = 27 – 1 = 127)$
- double (precisão dupla)
  - bias = 1023 $(2^{11-1} – 1 = 210 – 1 = 1023)$

Exemplo 1: Precisão Simples

- Como representar um número em float?

  float f = 15213.0;

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0

   $15213.0_{10} = 11101101101101_2$

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0

$15213.0_{10} = 11101101101101_2$

2) Normalizar

$11101101101101_2 = \boxed{1.1101101101101}^{M}_2 * 2^{13}$ (E)

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0

$15213.0_{10} = 11101101101101_2$

2) Normalizar

$11101101101101_2 = 1.\underbrace{1101101101101}_2 * 2^{13}$

3) frac = <u>1101101101101</u>0000000000 (frac → 23 bits)

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0
$15213.0_{10} = 11101101101101_2$

2) Normalizar
E
$11101101101101_2 = 1.1101101101101_2 * 2^{13}$

3) frac = 11011011011010000000000 (frac → 23 bits)

4) exp = E + bias = E + 127 = 13 + 127 = 140_{10}
exp = $140_{10} \rightarrow 1001100_2 = 0\underline{1001100}_2$ (exp → 8 bits)

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0
$15213.0_{10} = 11101101101101_2$

2) Normalizar
$11101101101101_2 = 1.1101101101101_2 * 2^{13}$

3) frac = 11011011011010000000000 (frac → 23 bits)

4) exp = E + bias = E + 127 = 13 + 127 = 140_{10}
exp = $140_{10} \rightarrow 1001100_2 = 0\underline{1001100}_2$ (exp → 8 bits)

5) Positivo → s = 0

Exemplo 1: Precisão Simples

1) Encontrar a representação binária de 15213.0
$15213.0_{10} = 11101101101101_2$

2) Normalizar
$11101101101101_2 = 1.1101101101101_2 * 2^{13}$

3) frac = 11011011011010000000000 (frac → 23 bits)

4) exp = E + bias = E + 127 = 13 + 127 = 140_{10}
exp = $140_{10} \rightarrow 1001100_2 = 0\underline{1001100}_2$ (exp → 8 bits)

5) Positivo → s = 0

1	8	23
0	01001100	11011011011010000000000

Exemplo 2: Precisão Simples

- Como encontrar o valor de um número representado em IEEE 754?
  - Por exemplo: 0x4640E400 (float)

Exemplo 2: Precisão Simples

Hex:	4	6	4	0	E	4	0	0
Binário:	0100	0110	0100	0000	1110	0100	0000	0000

1	8	23
0	10001100	10000000111001000000000000

Exemplo 2: Precisão Simples

1	8	23
0	10001100	10000000111001000000000000

frac = 1000000111001_2 → M = $1.\underline{1000000111001}_2$

exp = $10001100_2 = 140_{10}$ → E = 140 - 127 = 13

s = 0 → Positivo

Exemplo 2: Precisão Simples

1	8	23
0	10001100	10000001110010000000000

frac = 1000000111001_2 → M = 1.1000000111001_2
exp = $10001100_2 = 140_{10}$ → E = 140 - 127 = 13
s = 0 → Positivo

$$FP = (-1)^s * M * 2^E$$
$$= (-1)^0 * (1.1000000111001_2) * 2^{13}$$
$$= (-1)^0 * (11000000111001.0_2)$$
$$= 1 * (1*2^{13} + 1*2^{12} + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^0)$$
$$= 12345_{10}$$

Exemplo 3: Precisão Dupla

- Como representar um número em double?

  double d = 178.125;

Exemplo 3: Precisão Dupla

1) Encontrar a representação binária de 178.125
   $178.125_{10} = 10110010.001_2$

2) Normalizar
   $10110010.001_2 = 1.\underline{0110010001}_2 * 2^7$

3) frac = $\underline{0110010001}0000000000...0$

4) exp = E + bias = E + 1023 = 7 + 1023 = 1030_{10}
   exp = $1030_{10} \rightarrow 10000000110_2$ (11 bits)

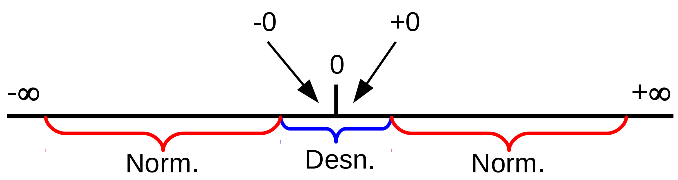
5) Positivo → s = 0

1	11	52
0	10000000110	01100100010000000000...0

Valores Desnormalizados

- Representação de <u>zero</u> e números muito <u>próximos de zero</u> (positivo ou negativo)

s	0000...0	frac



Valores Desnormalizados

- Representação de zero e números muito próximos de zero (positivo ou negativo)
  - M = frac → não há "1" implícito (0 <= M < 1)
  - E = 1 – bias → E = -126 ou E = -1022

s	0000...0	frac

$$FP = (-1)^s * 0.bbbb...b * 2^{-126}$$

$$FP = (-1)^s * 0.bbbb...b * 2^{-1022}$$

Valores Infinitos e NaN

+ ∞	s = 0	exp = 111...111	M = 0
- ∞	s = 1	exp = 111...111	M = 0
NaN	s = ?	exp = 111...111	M ≠ 0

- infinito: permite representação de *overflow*
  - Multiplicação, divisão por 0
- NaN (*Not a Number*)
  - Operações que não resultam em um número real (√-1 , +∞, -∞)
- Na biblioteca <math.h>: NAN, INFINITY