Representação de Ponto Flutuante

Universidade Federal do Ceará - Campus Quixadá

Pedro Botelho
pedro.botelho@ufc.br

02 de Outubro de 2025

Arquitetura de Computadores

Seção 1

Representação dos Números

Revisão de representação dos números

- Como vimos em aulas anteriores, os computadores só conseguem representar zeros e uns.
- Os números positivos são armazenados em binário.
  - Ex.: $41 = 00101001$.
- Consequentemente, o computador não consegue armazenar sinais de magnitude e nem a vírgula dos números reais.
  - Precisamos estabelecer formatos e padronizações.
  - Ex.: Complemento de 2, padrão IEEE 754

Tipos de números

- Já vimos...
  - Naturais.
  - Inteiros.
- Nessa aula veremos...
  - Vírgula fixa.
  - Ponto flutuante
- Para todos, a quantidade de valores possíveis depende do número de bits (N).
  - 2^N valores.

Representação de tipos de números

- Os números são infinitos.
- No computador eles são finitos.
- O computador pode lidar com números até um certo tamanho.
- No computador é preciso representar números com sinal.
  - Usamos o bit mais significativo para indicar o sinal.
- E quanto aos números com parte fracionária?
  - Veremos duas abordagens: Vírgula fixa e virgula flutuante

Seção 2

Ponto Fixo

Representando números reais

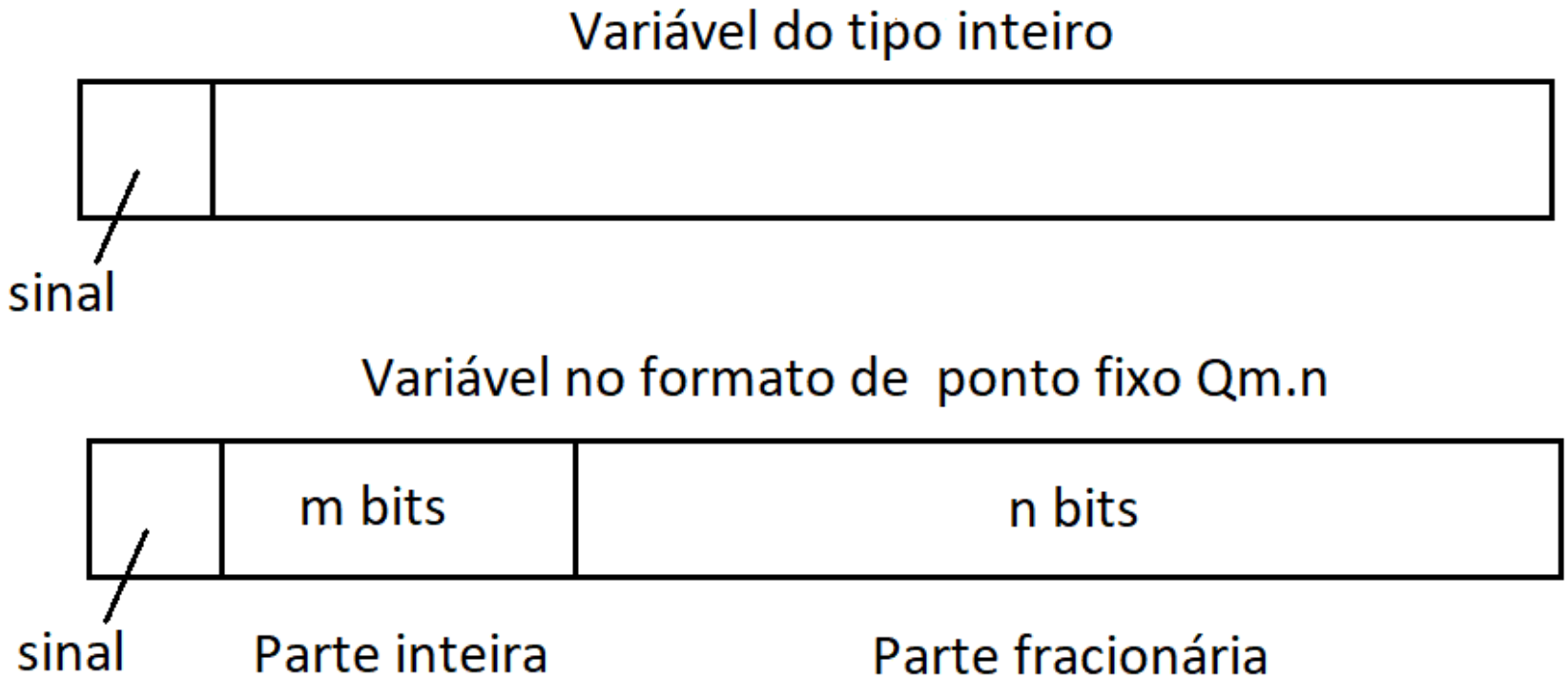
Uma questão:

- Quando se usa lápis e papel para resolver um problema trigonométrico, por exemplo, estamos trabalhando com o universo dos **números reais**.
- Os números reais são infinitos.
- No computador, a memória é finita e o tamanho dos valores também.
- Quanto maior a quantidade de bits para armazenar a variável de ponto flutuante, mais precisa será a aproximação.
- Como representar a vírgula dos números reais?

Representação em ponto fixo

- Usa um número fixo de bits para a parte inteira e para a parte fracionária, onde o "ponto" é sempre posicionado na mesma localidade
- Abordagem mais simples $\rightarrow$ Divide o número em duas partes
  - Parte real $\rightarrow$ Bits tem valor acima de 1
  - Parte fracionária $\rightarrow$ Bits tem valor abaixo de 1
- Permite aritmética facilitada e complemento de 2 para representar o sinal

Formato dos números em ponto fixo



Exemplos (para $Q4.4$):

- $+2,5 \rightarrow 00101000_2$
- $-2,5 \rightarrow 11011000_2$

Conversão entre sistemas de numeração

Conversão de números fracionários:

Número:

$$a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-1} + \ldots + a_0 b^0 + a_{-1} b^{-1} + a_2 b^{-2} + \ldots + a_{-m} b^m$$

Exemplo: $(101,110)_2 = (?)_{10}$

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} = (5,75)_{10}$$

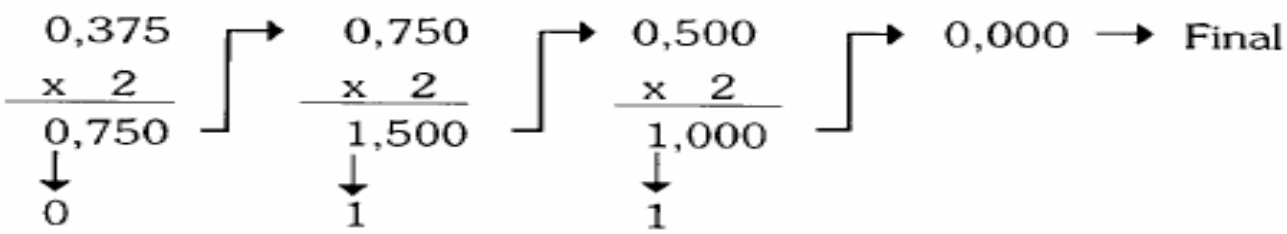
Conversão de números fracionários

Decimal $\Longrightarrow$ Outro sistema

- Operação inversa: multiplicar a parte fracionária pela base até que a parte fracionária do resultado seja zero.

Exemplo: $(8{,}375)_{10} = (\ ?\ )_2$

- parte inteira: $(8)_{10} = (1000)_2$
- parte fracionária:



$(8{,}375)_{10} = (1000{,}011)_2$

Faixa de valores

- Muito simples $\rightarrow$ Possui uma faixa de valores bastante limitada
- Não é possível representar muitos valores ou obter muita precisão
- $Q_{m.n} = -2^m$ até $2^m - 2^{-n}$
- Ex.: $Q_{4.4} = -2^4$ até $2^4 - 2^{-4}$
  - $= -16.0$ (10000000_2) até 15.9375 (11111111_2)
- Portanto $\rightarrow$ Uma boa opção caso precisão não seja um problema!

Aritmética em Ponto Fixo

- Muito simples $\rightarrow$ Não há necessidade de hardware especializado (FPU)
- Computadores sem FPU $\rightarrow$ Programador pode inserir manualmente números em ponto fixo
  - Boa opção para microcontroladores simples
- Operações aritméticas $\rightarrow$ Quase idênticas às com inteiros
  - Soma e subtração $\rightarrow$ Se mantém iguais
  - Multiplicação entre $A_{m.n}$ e $B_{m.n}$ terá o dobro de bits, logo devemos "deslocar n bits", ou seja, $\frac{A_{m.n} \times B_{m.n}}{2^n}$
  - Na divisão deve-se aumentar a escala do dividendo para não perder a fração, logo $\frac{A_{m.n} \times 2^n}{B_{m.n}}$

Seção 3

Ponto Flutuante

Limitações do ponto fixo

- Lembre-se: Quanto maior a quantidade de bits para armazenar a variável de ponto flutuante, mais precisa será a aproximação.
- Com uma notação de ponto fixo, é possível representar um intervalo de inteiros positivos e negativos centrados em 0.
- Assumindo um binário fixo e ponto fracionário, esse formato permite a representação de números também com um componente fracionário.
- Entretanto, números muito grandes e nem frações muito pequenas não podem ser representados.
- Para números decimais, contornamos essa limitação usando a notação científica.
  - Assim, 976.000.000.000.000 pode ser representado como 9,76 $\times$ 10^{14} e 0,000000000000976 pode ser representado como 9,76 $\times$ 10^{-14}.

Notação cientifica para números binários

- Essa mesma técnica pode ser usada com números binários. Podemos representar um número no formato:

$$\pm S \times B^{\pm E} \tag{1}$$

- Esse número pode ser armazenado em uma palavra binária com três campos.
  - Sinal (S) $\rightarrow$ mais ou menos.
  - Mantissa ou significado (B).
  - Expoente (E).

Número normalizado

- Para simplificar as operações sobre números ponto flutuante, normalmente é exigido que eles sejam normalizados.
- Um **número normalizado** é aquele em que o dígito mais significativo do significado é diferente de zero.
- Para a representação na base 2, um número normalizado é, portanto, um número em que o bit mais significativo do significado é 1.
- Primeiro se converte para binário (igual feito anteriormente)
- Ex.: Para 35.25_{10}, temos...
  - Parte **inteira**: $35_{10} = 11110_2$
  - Parte **fracionária**: $0.25_{10} = 0.01_2$
  - Normalizar: $11110.01_2 \rightarrow 1.111001_2 \times 2^4$, onde 4 é o "expoente"

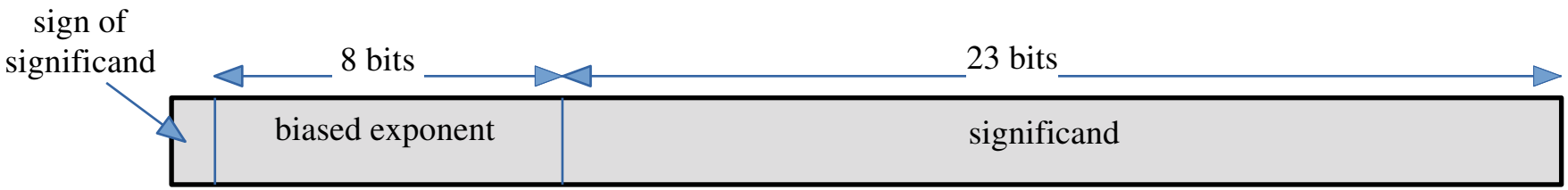
Expoente polarizado

- O expoente é armazenado com uma **representação polarizada**:
  - Basicamente, soma-se um valor fixo ao número (normalmente $2^{k-1} - 1$, sendo k o número de bits) para aí sim armazená-lo.
  - Para se obter o valor verdadeiros do expoente, é só subtrair o mesmo valor do campo de expoente.
- Uma vantagem da representação polarizada é que os números de ponto flutuante não negativos podem ser tratados como inteiros para fins de comparação.
- Em outras palavras $\rightarrow$ Podemos representar números positivos e negativos usando apenas valores não negativos no campo do expoente

Ex.: Se o expoente tem 8 bits, o valor 2 será armazenado como 129!

Ponto flutuante de precisão simples

- Computador precisará manipular o binário de uma maneira específica
- Um *hardware* especializado faz essa manipulação: Unidade de Ponto Flutuante (FPU)
- Número binário efetivo dividido em **campos de bits**:
  - O formato padrão para um número binário de precisão simples, o bit de sinal (S) é o bit mais à esquerda, o expoente (E) corresponde aos próximos 8 bits e a mantissa ou parte fracionária (F) inclui os 23 bits restantes.



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

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

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

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

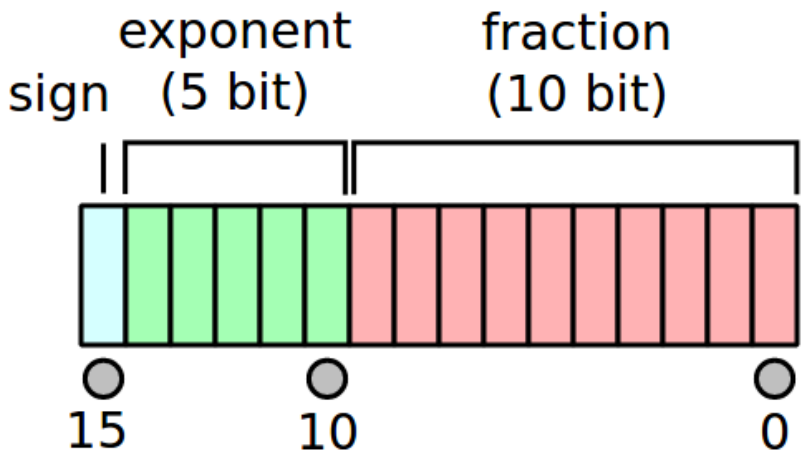
(b) Examples

Passos para a conversão

1. Determinar o sinal
2. Escrever o valor absoluto em binário
3. Normalizar (forma científica binária)
4. Calcular o expoente polarizado (*bias*)
5. Determinar a mantissa
6. Montar o valor final no formato padrão desejado (padrão IEEE-754)

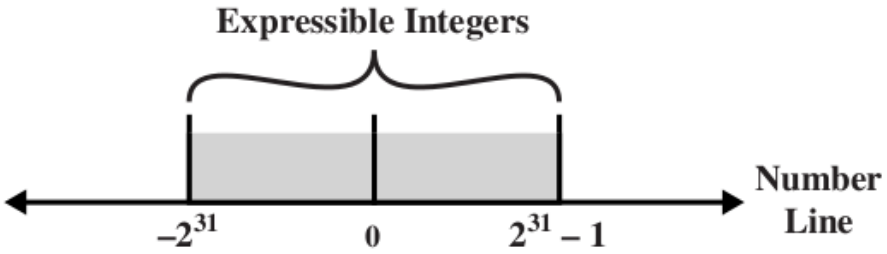
Exemplo de conversão

Converter $+12,5_{10} \rightarrow ?_2$ usando o padrão *half-precision*:

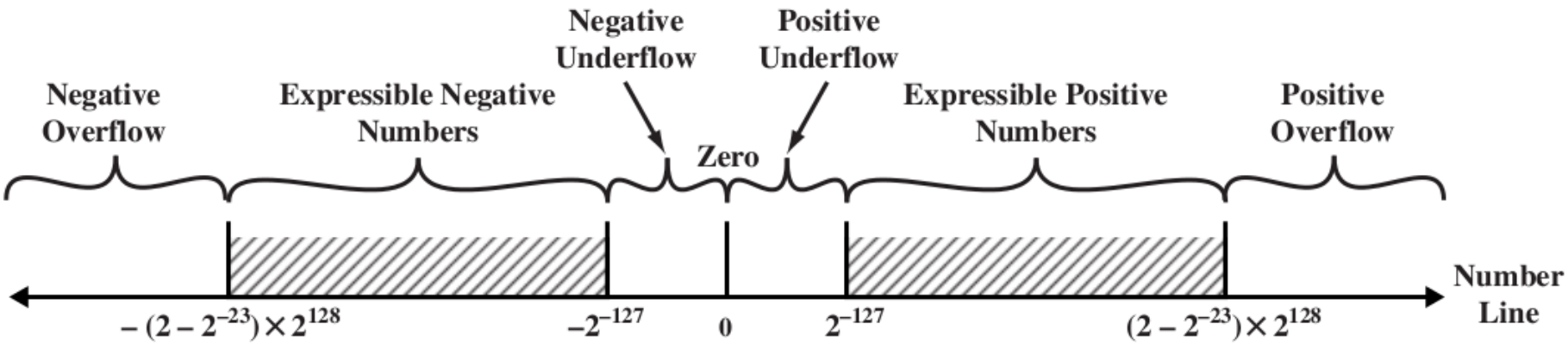


1. Sinal: Positivo, logo, 0
2. Binário: 1100.1_2
3. Normalizar: $1.1001_2 \times 2^3$
4. Expoente: *Bias* é $2^4 - 1 = 15 + 3 = 18$
5. Mantissa: 1001000000_2
6. Valor final: 1(S) 10010(E) 1001000000(B) $\rightarrow 1100101001000000_2$

Números expressos em formatos típicos de 32 bits



(a) Twos Complement Integers



(b) Floating-Point Numbers

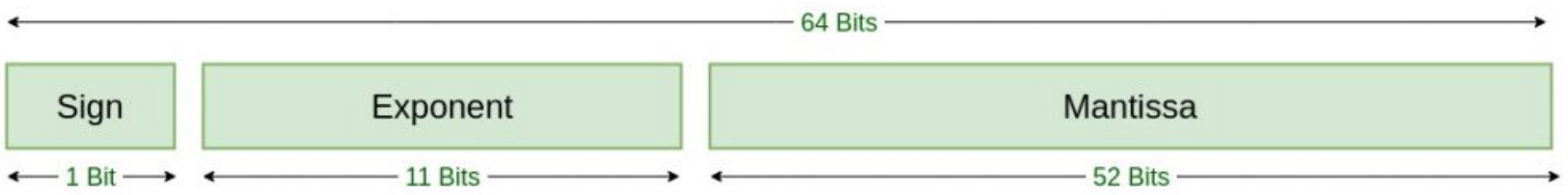
Por que temos dois zero s(+0 e -0)? Ajuda a descobrir a direção do underflow! Recíproca retorna dois infinitos!

O padrão IEEE 754

- Até meados dos anos 1980, cada fabricante de computador tinha seu próprio formato para representar números em ponto flutuante.
- Solução: criação do padrão 754 pela IEEE, em 1985.
- O padrão IEEE 754 procurou uniformizar a maneira como as diferentes máquinas representam os números em ponto flutuante, bem como devem operá-los.
- O padrão IEEE 754 para ponto (vírgula) flutuante é a representação mais comum para números reais em computadores de hoje, incluindo PC's compatíveis com Intel, Macintosh, e a maioria das plataformas Unix/Linux.

Padrão de precisão dupla

Além dos padrões de 16 bits (half precision) e 32 bits (single precision), o IEEE 754 define um padrão de 64 bits (double precision):



Double Precision
IEEE 754 Floating-Point Standard

Exemplos

Transforme os valores a seguir para ponto flutuante de 32 bits padrão IEEE 754:

- $+9{,}5$
- $+2{,}3$
- $+10{,}6$

Exemplos

Transforme os valores a seguir para ponto flutuante de 32 bits padrão IEEE 754:

- $+9{,}5 = 0\ 10000010\ 00110000000000000000000$
- $+2{,}3 = 0\ 10000000\ 00100110011001100110011$
- $+10{,}6 = 0\ 10000010\ 01010011001100110011010$

Aplicações Reais

- Nossos computadores utilizam muito números de ponto flutuante (variáveis float e double)
  - Possuem FPUs poderosas para realizar as operações
- Computadores sem FPU $\rightarrow$ Compilador utiliza funções de *software* que emulam operações de ponto flutuante
  - Muito lento, porém com muita precisão
  - Programador pode usar ponto fixo manualmente se quiser $\rightarrow$ Muito rápido, porém com pouca precisão

Representação de Ponto Flutuante

Universidade Federal do Ceará - Campus Quixadá

Pedro Botelho
pedro.botelho@ufc.br

02 de Outubro de 2025

Arquitetura de Computadores