Sistemas Numéricos
e
Aritmética Binária

Arquitetura e Organização de Computadores

Prof. Patrick H S Brito

Slides baseados no livro texto do curso

Introdução

- **Os computadores compreendem apenas 0 ou 1**
- ⇨ Sistema Binário
- ⇨ Usamos normalmente o Sistema Decimal

Decimal

Sistema Binário

Introdução

- **No sistema binário, temos apenas dois dígitos chamados *bit***
  - ⇨ *bit 0/bit 1*
  - ⇨ A menor quantidade de informação

Introdução

- **Um conjunto de 8 bits forma 1 Byte (Binary Term)**

8 bits = 1 Byte

11011011

- **Um conjunto de 1024 Bytes forma 1 Kbyte (KiloByte)**
- **Um conjunto de 1024 KBytes forma 1MByte (MegaByte)**
- **Um conjunto de 1024 MBytes forma 1GByte (GigaByte)**

Introdução

- ... assim sucessivamente...

8 bits = 1 Byte – 2^0 Byte

1024 Byte = 1 KiloByte (KByte - KB) – 2^{10} Bytes

1024 KByte = 1 MByte (MegaByte - MB) – 2^{20} Bytes

1024 MByte = 1 GByte (GigaByte - GB) - 2^{30} Bytes

1024 GByte = 1 TByte (TeraByte - TB) - 2^{40} Bytes

1024 TByte = 1 PByte (PetaByte - PB) – 2^{50} Bytes

......

Introdução

- **1 byte é a quantidade de informação necessária para armazenar um caractere da nossa linguagem**

⇨ letra, número, espaço, pontuação etc.

C = 01000011
A = 01000001
S = 01010011
A = 01000001

Introdução

- **Podemos transformar valores com operações matemáticas simples**

$$1GB = bits?$$

1. Sabemos que $1GB = 2^{30}$ Bytes
2. Logo, $1GB = 2^{30}$ Bytes * 8 *bits*

   = 8.589.934.592 *bits*

Isto é, $\underbrace{1024}_{MB} * \underbrace{1024}_{KB} * \underbrace{1024}_{B} * \underset{bits}{8}$

Introdução

- **Podemos transformar valores com operações matemáticas simples**

2,5 GB = bits?

1. Sabemos que 1GB = 2^{30} Bytes
2. Logo, 2,5 GB = 2,5 * 2^{30} Bytes * 8 *bits*
   = 2,5 * 8.589.934.592 *bits*
   = 21.474.836.480 *bits*

Isto é, $2{,}5 * \underbrace{1024}_{\text{MB}} * \underbrace{1024}_{\text{KB}} * \underbrace{1024}_{\text{B}} * \underset{\text{bits}}{8}$

Introdução

- Podemos transformar valores com operações matemáticas simples

3000 KBytes = MB?

1. Sabemos que 1MB = 1024 KB

2. Logo, $MB = \dfrac{3000\ KB}{1024}$

$MB \approx 2{,}93\ MB$

Introdução

❑ Resumindo...

$$\text{bits} \underset{x\,*\,8}{\overset{x\,/\,8}{\rightleftarrows}} \text{Byte} \underset{x\,*\,1024}{\overset{x\,/\,1024}{\rightleftarrows}} \text{KByte} \underset{x\,*\,1024}{\overset{x\,/\,1024}{\rightleftarrows}} \text{MByte} \underset{x\,*\,1024}{\overset{x\,/\,1024}{\rightleftarrows}} \text{GByte} \underset{x\,*\,1024}{\overset{x\,/\,1024}{\rightleftarrows}} \text{TByte}$$

Ex.: 3000KB -> GB: $3000 / 1024 = \underbrace{2{,}93}_{\text{MB}} / 1024 = \underbrace{0{,}00286\ \text{GB}}_{\text{GB}}$

Ex.: 10TB -> MB: $10 * 1024 = \underbrace{10240}_{\text{GB}} * 1024 = \underbrace{10485760\ \text{MB}}_{\text{MB}}$

Introdução

Exercícios de Fixação

⇨ 1. Marque a opção que requer o maior espaço de armazenamento

a) 1 Arquivo de 1GB $= 1 * 1 * 2^{30} \approx$ 1 bilhão de bytes

X b) 2000 Arquivos de 1 MB $= 2000 * 1 * 2^{20} \approx$ 2 bilhões de bytes

c) 10 arquivos de 4 KB $= 10 * 4 * 2^{10} =$ 40.960 bytes

d) 2 arquivos de 64 bytes = 128 bytes

e) 1 arquivo de 128 bits = 16 bytes

Conversão de Bases

- **Todo número escrito numa base b pode ser considerado segundo o polinômio**

$$a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0$$

$$a_1 \ldots a_n < b$$

$$143 = 1 \times 10^2 + 4 \times 10^1 + 3 \times 10^0$$

$$143 = 1 \times 100 + 4 \times 10 + 3 \times 1$$

Conversão de Bases

- **De forma geral, no sistema decimal**

$$a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_1 10^1 + a_0 10^0$$

- **Logo, para transformarmos de binário para decimal**

$$a_n 2^n + a_{n-1} 2^{n-1} + \dots + a_1 2^1 + a_0 2^0$$

$$a = 1 \text{ ou } 0$$

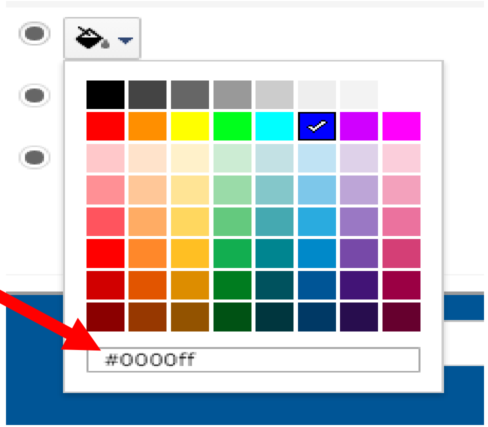
$$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$(13)_{10} = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$

Conversão de Bases

- **Além do sistema binário, o sistema hexadecimal é comumente utilizado na computação**
  - Base 16 – (0 – 15)
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, F

- **Onde encontramos... Exemplo 1**



Conversão de Bases

- **Um número em hexadecimal pode ser transformado para decimal**

$$a_n 16^n + a_{n-1} 16^{n-1} + \ldots + a_1 16^1 + a_0 16^0$$

- **Atentar-se a seguinte conversão**
- A = 10
- B = 11
- C = 12
- D = 13
- E = 14
- F = 15

Conversão de Bases

❑ Exemplo

⇨ $(3BF4C)_{16 \rightarrow 10}$

$= 3 \times 16^4 + B \times 16^3 + F \times 16^2 + 4 \times 16^1 + C \times 16^0$

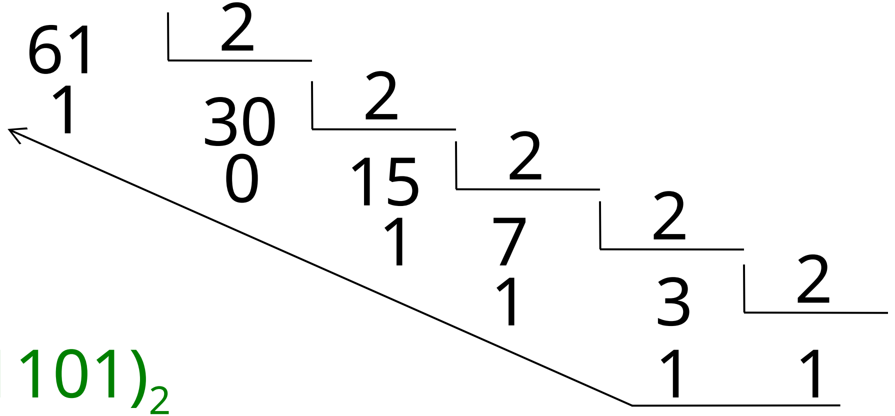
$= 196.608 + 11 \times 4096 + 15 * 256 + 64 + 12 \times 1$

$= 196.608 + 45.056 + 3.840 + 64 + 12$

$= (245580)_{10}$

Conversão de Bases

- **Podemos transformar quaisquer números decimal para a base b (ex.: binária ou hexadecima)**
  - ⇨ Divisões sucessivas do número decimal por **b** enquanto quociente **q** for maior que **b**
  - ⇨ Ex.: $(61)_{10 \rightarrow 2}$



$$(61)_{10} = (111101)_2$$

q menor que b

Conversão de Bases

- **Podemos transformar quaisquer números decimal para a base b (ex.: binária ou hexadecima)**
- ⇨ Divisões sucessivas do número decimal por **b** enquanto quociente **q** for maior que **b**
- ⇨ Ex.: $(61)_{10 \rightarrow 16}$

$$\begin{array}{l|l} 61 & 16 \\ \hline 13 & 3 \end{array}$$

q menor que b

$$(61)_{10} = (3D)_{16}$$

Conversão de Bases – PONTO FLUTUANTE

- **Para números decimais fracionários, devemos separar a parte inteira e fracionária**
- Ex.: $(8{,}375)_{10}$
- Este número pode ser representado
  - 8 + 0,375
- Transforma a parte inteira normalmente e a parte fracionária multiplica por **b** sucessivamente enquanto maior que 0

$$
\begin{array}{l|l}
8 & 2 \\ \hline
0 & 4
\end{array}
\quad
\begin{array}{l|l}
4 & 2 \\ \hline
0 & 2
\end{array}
\quad
\begin{array}{l|l}
2 & 2 \\ \hline
0 & 1
\end{array}
$$

1000

0,375 x 2 = 0,750

0,750 x 2 = 1,500

0,500 x 2 = 1,000

011

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

Conversão de Bases – PONTO FLUTUANTE

- **Para números binários, devemos utilizar a representação de polinômios**

⇨ Separa-se parte inteira e a parte fracionária recebe potências negativas sequencialmente

⇨ $(1000{,}011)_2$

$$1000 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8$$

$$011 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

$$011 = 0 + 1 \times 1/4 + 1 \times 1/8$$

$$011 = 0 + 0{,}25 + 0{,}125$$

$$= 0{,}375$$

$$8 + 0{,}375 = (8{,}375)_{10}$$

Conversão de Bases – PONTO FLUTUANTE

- **A conversão da parte fracionada (ponto flutuante) NEM SEMPRE é exata!**

⇨ $(8{,}260)_{10 \rightarrow 2}$

8 = 1000

0,260 x 2 = 0,520
0,520 x 2 = 1,040
0,040 x 2 = 0,080
0,080 x 2 = 0,160
0,160 x 2 = 0,320
0,320 x 2 = 0,640
0,640 x 2 = 1,280
0,280 x 2 = 0,560

...quando parar?

1000,01000010 ...

Frequente perda de precisão

Conversão de Bases – PONTO FLUTUANTE

- **É necessário fixar uma quantidade de bits!**

⇨ $(8{,}260)_{10 \to 2}$ **1000,01000010100011110...**

- **Se 5 bits: 1000,01000**

⇨ **1000**,01000 $= 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5}$

(8,25000)

- **Se 7 bits: 1000,0100001**

⇨ **1000**,0100001 $= 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5} + 0 \times 2^{-6} + 1 \times 2^{-7}$

(8,2578125)

- **Se 9 bits: 1000,010000101**

⇨ **1000**,0100001 $= 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5} + 0 \times 2^{-6} + 1 \times 2^{-7} + 0 \times 2^{-8} + 1 \times 2^{-9}$

(8,259765625)

Conversão de Bases

- **DICA para conversão de binário para bases “potências de 2”**
  - Se a base destino é 2^x
  - Agrupe o valor binário de x em x algarismos (da direita para a esquerda)
  - Converta o valor de cada um dos grupos
- **Por Exemplo:**
  - $(10010101)_2 \rightarrow (1001\ 0101)_{16} \rightarrow (95)_{16}$
  - $(10010101)_2 \rightarrow (10\ 010\ 101)_8 \rightarrow (225)_8$
  - $(10010101)_2 \rightarrow (10\ 01\ 01\ 01)_4 \rightarrow (2111)_4$

Conversão de Bases

- **Transforme os números abaixo para as referidas bases**

⇨ $(86)_{10}$ para base 2

⇨ $(110110)_{2}$ para base 10

⇨ $(96)_{10}$ para base 16

⇨ $(10,142)_{10}$ para base 2

⇨ $(1011\ 0110)_{2}$ para base 16

⇨ $(10\ 110\ 110)_{2}$ para base 8

⇨ $(10\ 11\ 01\ 10)_{2}$ para base 4

⇨ $(10F)_{16}$ para base 2

Aritmética Binária - SOMA

Segue o mesmo princípio da soma na base 10, mas considerando o teto de um dígito como sendo < 2 (e não < 10):

0 + 0 = 0

1 + 0 = 1

0 + 1 = 1

1 + 1 = 1 (1 vai 1 para a próxima ordem de grandeza)

10010 + 10111?

Aritmética Binária – SUBTRAÇÃO (Complemento a 2)

- **Economia na construção do circuito**
  - **Utiliza o mesmo circuito da soma**
  - **Por exemplo: 7 – 5 ↔️ 7 + (-5)**
- **Limitação: os números devem ter “tamanho” fixo**
  - É necessário fixar a quantidade de bits
- **Caso o resultado exceda esta quantidade de bits, o bit mais à esquerda é desprezado**

- **Procedimento de Complemento a 2:**
  1. Os números negativos devem ter seus bits invertidos
  2. Soma-se 1 ao valor obtido

Aritmética Binária – SUBTRAÇÃO (Complemento a 2)

- **Faça 10 – 5 utilizando complemento a 2. Suponha que seu processador trabalhe com números de 5 bits**
- **Na verdade, deve-se fazer 10 + (-5)**
- **10, em binário é: 01010**
- **5 em binário é: 00101**
- **Aplicando o complemento a 2, obteremos -5:**
  - 00101. Invertendo seus bits, temos: 11010
  - Fazendo 11010 + 1, temos 11011
- **Agora, basta somar: 01010 + 11011.**

100101 (número de 5 bits).

$(00101)_2$ ou $(5)_{10}$

Aritmética Binária – SUBTRAÇÃO (Complemento a 2)

- E se fosse 3 – 5? (processador de 5 bits)
- Convenção (complemento a 2):
  - BIT mais à esqueda
    - 0 → Número Positivo
    - 1 → Número Negativo
- 5 BITS: $\pm(2^{n-1} - 1)$
  - De -15 (11111)
  - Até 15 (01111)

Aritmética Binária – SUBTRAÇÃO (Complemento a 2)

- **E se fosse 3 – 5? (processador de 5 bits)**
- **Na verdade, deve-se fazer 3 + (-5)**
- **3, em binário é: 00011**
- **5 em binário é: 00101**
- **Aplicando o complemento a 2, obteremos -5:**
  - 00101. Invertendo seus bits, temos: 11010
  - Fazendo 11010 + 1, temos 11011
- **Agora, basta somar: 00011 + 11011.**

$(11110)_2$ (número negativo).

$(00001)_2+1$ ↔️ $-(00010)_2$ ↔️ $(-2)_{10}$

Overflow e *Underflow*

Os números manipulados

- grande demais para ser representados provocam um *overflow*.
- pequeno demais para ser representados provocam um *underflow*.

Os sistemas têm feedback diferentes em caso de *over* ou *underflow*. Alguns param a execução, outros dão uma mensagem e outros representam o número de uma forma específica.

Conclusão

- A representação dos números depende do suporte material para representar e calcular (binário com o computador).
- O computador usa representação finita, ele não pode representar de forma exata todos os números reais.
- Quanto mais bits decimais, maior a precisão numérica.
- Processadores atuais:
    - 3 Modos de operação: 7, 17 e 20 bits de precisão (IEEE 754)