Sistemas de Computação

Práticas Laboratoriais

Aula 1

Fev.2019

A.Esteves & B. Medeiros

Sumário

- Sistemas de numeração e conversão de bases
- Operações aritméticas e lógicas em base 2
- Representação binária de inteiros positivos e negativos
  - Sinal e amplitude/magnitude
  - Complemento para 1
  - Complemento para 2
  - Representação por excesso

Sistemas de numeração

- Os números podem ser representados em vários sistemas diferentes de numeração
  - Humanos normalmente usam um sistema de numeração de **base 10**
  - Os computadores, por serem dispositivos elétricos utilizam **dígitos binários (base 2)**, conhecidos por **bits**
    - **0 – desligado** <u>ou</u> tensão baixa
    - **1 – ligado** <u>ou</u> tensão alta

Sistemas de numeração

- Conceito de **ordem** e **base**
  - **Exemplo : $1532{,}64_{10}$**
- A **base** utilizada determina o número de dígitos que podem ser utilizados.
  - Base **10** → 10 dígitos (0..9)
  - Base **2** → 2 dígitos (0, 1)
  - Base **16** → 16 dígitos (0.. 9, A .. F)
- A **ordem** de um dígito é a posição que ele ocupa no número, em relação à vírgula 'decimal' (ou separador das partes inteira e fracionária)

Conversões entre bases

- A conversão de um número na **base b** para a **base decimal** obtém-se somando os resultados de multiplicar cada dígito pela base elevada à ordem do dígito.
- **Exemplos**
- 1532_6 (base 6)

$$= 1 * 6^3 + 5 * 6^2 + 3 * 6^1 + 2 * 6^0 = 416_{10}$$

- $110110{,}011_2$ (base 2)

$$= 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 +$$

$$+ 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} = 54{,}375_{10}$$

Conversão de bases

- Na conversão de um número na **base decimal** para uma **base b**, o processo mais direto é composto por duas partes:

  - **Divisão sucessivas da parte inteira** do número pela respetiva base **b** → cada resto obtido é um dígito na base **b** e o quociente é o dividendo na próxima divisão → pára quando quociente=0

  - **Multiplicação sucessiva da parta fracionária** desse número pela respetiva base **b** → a parte inteira de cada produto é um dígito da base **b** e a parte fracionária é usada na próxima multiplicação → pára quando parte fracionária=0

Exemplo: conversão decimal → binário

- **$235,375_{10}$ → $11101011,011_2$**

235/2 = 117 Resto = 1 (lsb inteiro)
117/2 = 58 Resto = 1
58/2 = 29 Resto = 0
29/2 = 14 Resto = 1
14/2 = 7 Resto = 0
7/2 = 3 Resto = 1
3/2 = 1 Resto = 1
1/2 = 0 →pára Resto = 1 (msb inteiro)
0,375 * 2 = 0,75 P. Int. = **0** (msb fracionário)
0,75 * 2 = 1,5 P. Int. = **1**
0,5 * 2 = 1,0 P. Int. = **1** (lsb fracionário)
↳pára

Subtrações sucessivas

- É outro processo de conversão de base decimal para outra base
- Normalmente utilizado na conversão decimal para binário de valores não muito grandes
- Método mais rápido
- Necessita que o utilizador saiba a tabuada das potências de 2

Tabuada das potências de 2

...

$2^{12} = 4096$	$2^{6} = 64$	$2^{0} = 1$
$2^{11} = 2048$	$2^{5} = 32$	$2^{-1} = 0,5$
$2^{10} = 1024$	$2^{4} = 16$	$2^{-2} = 0,25$
$2^{9} = 512$	$2^{3} = 8$	$2^{-3} = 0,125$
$2^{8} = 256$	$2^{2} = 4$	$2^{-4} = 0,0625$
$2^{7} = 128$	$2^{1} = 2$	...

Subtrações sucessivas

- Como converter de decimal para binário?
  - Procura-se a maior potência de 2 imediatamente inferior (ou igual) ao valor do número decimal, e subtrai-se essa potência do número decimal
  - O <u>expoente da potência</u> indica que o número binário tem o bit dessa ordem a 1
  - Com o <u>resultado da subtração</u> repete-se o processo até chegar ao resultado 0

Subtrações sucessivas

- **Exemplo:** $1081{,}625_{10}$

1081,625 – 2^{10} = 57,625 → bit 10 a 1
57,625 – 2^{5} = 25,625 → bit 5 a 1
25,625 – 2^{4} = 9,625 → bit 4 a 1
9,625 – 2^{3} = 1,625 → bit 3 a 1
1,625 – 2^{0} = 0,625 → bit 0 a 1
0,625 – 2^{-1} = 0,125 → bit -1 a 1
0,125 – 2^{-3} = **0** → bit -3 a 1

1	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1,**	**1**	**0**	**1**
1024	512	256	128	64	32	16	8	4	2	1	0,5	0,25	0,125
$\mathbf{2^{10}}$	2^{9}	2^{8}	2^{7}	2^{6}	$\mathbf{2^{5}}$	$\mathbf{2^{4}}$	$\mathbf{2^{3}}$	2^{2}	2^{1}	$\mathbf{2^{0}}$	$\mathbf{2^{-1}}$	2^{-2}	$\mathbf{2^{-3}}$

Número de bits

- Os processadores utilizam um determinado número de bits para representar os números

- A quantidade de bits utilizados para respresentar os números inteiros determina a gama de valores representáveis

- Para uma **base** **b** com **n** dígitos a gama de valores representáveis é $\mathbf{b^n}$

- Sendo **n** o número de bits utilizados, a gama de valores representáveis em binário, usando **n bits** é $\mathbf{2^n}$

Base hexadecimal

- Normalmente usada como alternativa de representação de valores binários porque ...
  - É fácil converter entre hexadecimal e binário
  - Facilita a leitura e diminui a probabilidade de erro
- **Exemplo**: Converter $\mathbf{4312}_{10}$ para hexadecimal

```
4312 / 16  = 269   Resto = 8
 269 / 16  = 16    Resto = 13 (dígito D)
  16 / 16  =  1    Resto = 0
   1 / 16  =  0    Resto = 1
```

Logo, 4312_{10} = $\mathbf{10D8}_{16}$

Porquê hexadecimal?

- Fácil converter entre hexadecimal e binário
- Hexadecimal utiliza 16 dígitos (0,1,..,9, A, B, C, D, E, F)
- Cada dígito hexadecimal representa um valor entre 0 e 15
- Cada conjunto de 4 bits representa também um valor entre 0 e 15
- Como tirar partido desta característica nas conversões binário→hex e hex→binário?

Conversão bin→hex e hex→bin

- **Exemplo conversão hex→binário**:

  2 A F (hexadecimal)

  0010 1010 1111 (binário)

  $2AF_{16} = 0010\ 1010\ 1111_2$

- **Exemplo conversão binário→hex**:

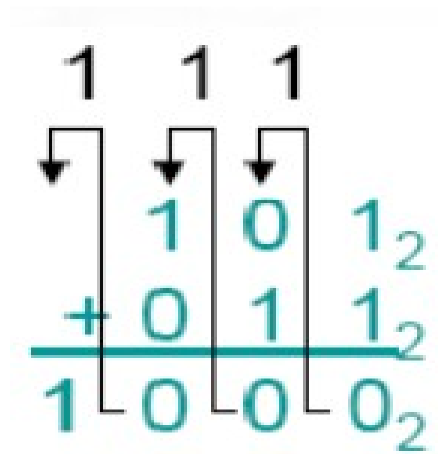
  1101 0101 1011 (binário)

  D 5 B (hexadecimal)

  $1101\ 0101\ 1011_2 = D5B_{16}$ ou 0xD5B ou 0xd5b

Operações aritméticas em base 2

- **Adição binária**
  - **Regras**
    - 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)
  - **Exemplo**: 101 + 011 = ?

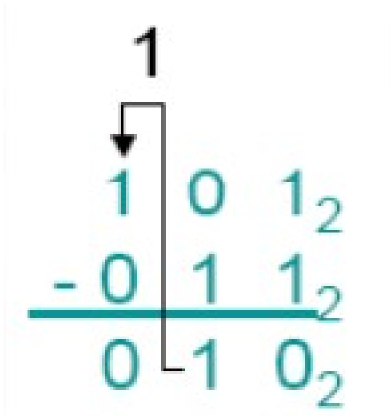


Multiplicação binária

- **Regras**
  - 0 x 0 = 0
  - 0 x 1 = 0
  - 1 x 0 = 0
  - 1 x 1 = 1
- O mesmo método que em decimal: deslocamentos e adições
- O número maior deve ser colocado acima do menor
- **Exemplo**: 011 x 101 = ?

Subtração binária

- **Regras**
  - 0 – 0 = 0
  - 0 – 1 = 1 e "pede emprestado 1" ao dígito de ordem superior
  - 1 – 0 = 1
  - 1 – 1 = 0
- **Exemplo**: 101 – 011 = ?



Números negativos

- Como representar o sinal negativo?

- Existe uma grande variedade de opções, das quais se destacam 4:

  - Sinal e amplitude/magnitude (S+M)

  - Complemento para 1

  - Complemento para 2

  - Notação em excesso

Sinal e magnitude

- Utiliza um bit para representar o **sinal** → o bit mais à esquerda:
  - 0 representa um número positivo
  - 1 representa um número negativo
- Os restantes bits representam a **magnitude**
- **Exemplo**: -109_{10} em S+M com 8 bits?

Complemento para 1

- Nesta representação invertem-se todos os bits de um número para representar o seu complementar
- Assim se converte um valor positivo em negativo, e vice-versa
- Quando o bit mais à esquerda for

  0 → o valor é **positivo**

  1 → o valor é **negativo**

- **Exemplo:**
  - 100_{10} =01100100_{2} (com 8 bits)
  - Invertendo todos os bits → 10011011_{2} = -100_{10}

Complemento para 2

- Um dos problemas da representação em complemento para 1 é existirem dois padrões de bits para representar o zero. Nomeadamente:
  $+0_{10} = 00000000_2$

  $-0_{10} = 1111111_2$

- **Solução**: representar os números em complemento para 2

- Para determinar o negativo de um número negam-se todos os seus bits e soma-se uma unidade

Complemento para 2

- **Exemplo:**

  100_{10} = 01100100_2 (com 8 bits)

  Invertendo todos os bits (complemento para 1):

  10011011_2

  Somando uma unidade:

  $10011011_2 + 1 = 10011100_2 = -100_{10}$

- **Exemplo**: -109_{10} em compl. para 2 com 8 bits?

Complemento para 2

- Características:
  - O bit da esquerda indica o sinal
  - O <u>processo anterior</u> serve para converter um número positivo para negativo e de negativo para positivo
  - O zero tem uma única representação: todos os bits a 0
  - A gama de valores que é possível representar com **n** bits aumenta: -2^{n-1} … $2^{n-1} - 1$

Complemento para 2

- **Exemplo:**
- Qual o número representado por 11100100_2 com 8 bits?
  - Como o bit à esquerda é 1 este número é negativo
  - Invertendo todos os bits fica: 00011011_2
  - Somando uma unidade:
    - $00011011_2 + 1 = 00011100_2 = 28_{10}$
  - Logo : $11100100_2 = \mathbf{-28}_{10}$

Complemento para 2

- Como é que se converte um número representado em complemento para 2 com **n** bits, para um número representado com mais bits?
- **Resposta** : Extende-se o bit de sinal
- **Exemplo**:
  - Converter os seguintes números de 8 para 16 bits:
  - **0**1101010$_2$ → **0**0000000 01101010$_2$ (positivo)
  - **1**1011110$_2$ → **1**1111111 11011110$_2$ (negativo)

Divisão e multiplicação por potências de 2

- Multiplicação por potências de 2
  - deslocamento de bits para a esquerda
- Divisão por potências de 2
  - Deslocamento de bits para a direita
- **Exemplos:**
  - Dividir $11001010_2 = 202_{10}$ por 4
    - Deslocar à direita 2 vezes ($2^2 = 4$)
    - $\mathbf{00}110010{,}10_2 = 50{,}5_{10}$
  - Multiplicar $00001010_2 = 10_{10}$ por 8
    - Deslocar à esquerda 3 vezes ($2^3 = 8$)
    - $01010\mathbf{000}_2 = 80_{10}$

Notação em excesso

- Tem uma vantagem sobre todas as outras referidas
  - O valor em binário com todos os **bits a 0** representa o **menor valor inteiro**, quer este tenha sinal ou não
  - O valor em binário com todos **os** bits a 1 representa o **maior valor inteiro**, com ou sem sinal

- Como o próprio nome indica, esta codificação de um inteiro (negativo ou positivo) em binário com n bits é sempre feita em excesso (de 2^{n-1} ou $2^{n-1}-1$)

Notação em excesso

Exemplo

Binário (8 bits)	**Sinal + Ampl**	**Compl p/ 1**	**Compl p/ 2**	**Excesso** (128)
$0000\ 0001_2$	+1	+1	+1	-127
...				
$1000\ 0001_2$	-1	-126	-127	+1
...				
$1111\ 1110_2$	-126	-1	-2	+126

No exemplo com 8 bits (**n**), o valor $+1_{10}$ é representado em binário, usando a notação por excesso de $2^{n-1}=2^{8-1}=128$ pelo valor:

$+1_{10}$ + excesso = $+1_{10} + 128_{10}$ = $0000\ 0001_2 + 1000\ 0000_2$ =

= $1000\ 0001_2$

Diferenças entre os 4 modos

A tabela que a seguir se apresenta, representando todas as combinações possíveis com 4 bits, ilustra de modo mais completo as diferenças entre estes 4 modos (+1 variante) de representar inteiros com sinal.

Binário (4 bits)	**Sinal + Ampl**	**Compl p/ 1**	**Compl p/ 2**	**Excesso** (7)	**Excesso** (8)
0000	0	0	0	-7	-8
0001	1	1	1	-6	-7
0010	2	2	2	-5	-6
0011	3	3	3	-4	-5
0100	4	4	4	-3	-4
0101	5	5	5	-2	-3
0110	6	6	6	-1	-2
0111	7	7	7	0	-1
1000	-0	-7	-8	1	0
1001	-1	-6	-7	2	1
1010	-2	-5	-6	3	2
1011	-3	-4	-5	4	3
1100	-4	-3	-4	5	4
1101	-5	-2	-3	6	5
1110	-6	-1	-2	7	6
1111	-7	-0	-1	8	7

TPC

- O enunciado do **TPC1** está na página deve ser impresso e levado, **obrigatoriamente** para as próximas aulas Teórico-Práticas