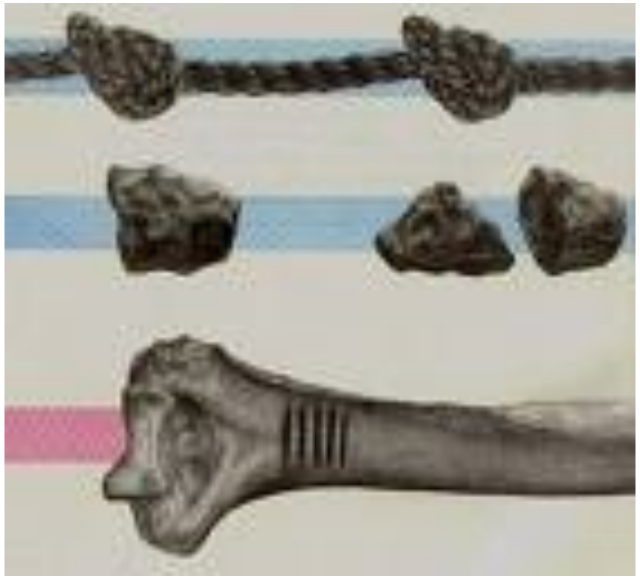
Infraestrutura de Hardware

Aritmética Computacional

Universidade Federal Rural de Pernambuco
Professor: Abner Corrêa Barros
abnerbarros@gmail.com

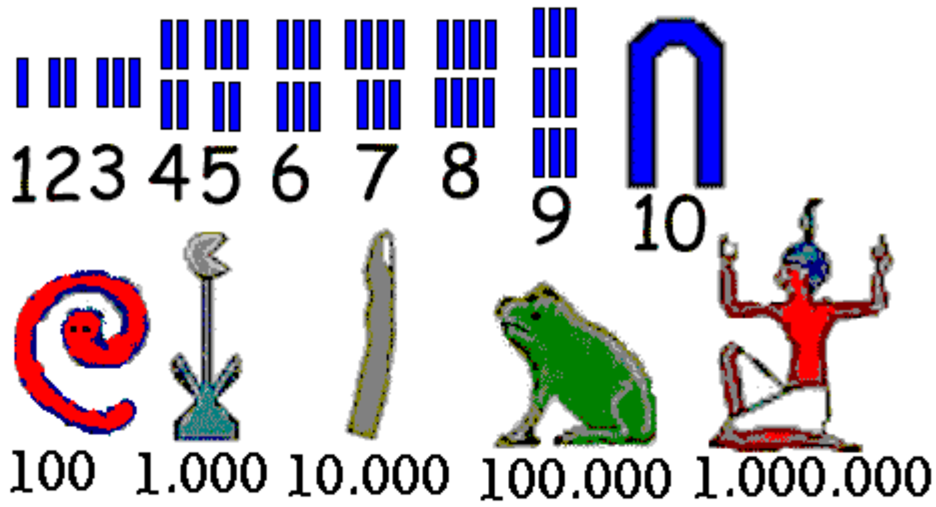
Introdução

- Desde os primórdios da sua história os homens tem se deparado com a necessidade de contar, enumerar e/ou ordenar as coisas que o cercam.



Sistemas de Numeração

- Um dos sistemas de numeração mais antigos que se tem notícia é o Egípcio. De base decimal, utilizava os seguintes símbolos em sua representação gráfica:



Sistemas de Numeração

- Foi no Norte da Índia, por volta do século V da era cristã, que provavelmente nasceu o sistema de notação atual adotado.
- Por ter sido largamente empregado pelos árabes, os quais o introduziram na Europa, este ficou conhecido como sistema de numeração Hindo-Arábico.

Base Numérica

- Conjunto de símbolos reservados à representação de valores numéricos

- Decimal
  - 10 símbolos (0,1,2,3,4,5,6,7,8,9)
- Octal
  - 8 símbolos (0,1,2,3,4,5,6,7)
- Hexa-decimal
  - 16 símbolos (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
- Binária
  - 2 símbolos (0,1)

Mas, como representar todo e qualquer valor numérico utilizando um conjunto tão restrito de símbolos?

Notação Posicional

- O valor é representado como um somatório ponderado dos símbolos utilizados.
- Cada símbolo é ponderado por uma potencia da base adotada, de acordo com a posição que ocupe na seqüência de símbolos utilizados.
- Ex:
  - $1111_{10} = 1x10^3 + 1x10^2 + 1x10^1 + 1x10^0$
  - $1111_2 = 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0$
  - $1234_{10} = 1x10^3 + 2x10^2 + 3x10^1 + 4x10^0$
  - $1234_8 = 1x8^3 + 2x8^2 + 3x8^1 + 4x8^0$
  - $12{,}514_{10} = 1x10^1 + 2x10^0 + 5x10^{-1} + 1x10^{-2} + 4x10^{-3}$
  - $11{,}010_2 = 1x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} + 0x2^{-3}$

Notação Posicional

Inteiros

$$z = \pm \sum_{i=1}^{n} d_i * \beta^{i-1}$$

Ponto Fixo

$$z = \pm \sum_{i=1}^{n} d_i * \beta^{i-1-k}$$

Ponto Flutuante

$$\pm \sum_{i=1}^{n} d_i * \beta^{i-1-k} * \beta^{\exp}$$

Equivalência entre representações

- Apenas valor numéricos inteiros podem ser expresso de forma exata em toda e qualquer base
- Alguns valores numéricos fracionários representáveis de forma exata em uma determinada base podem tornar-se em dízimas quando representados uma outra base qualquer

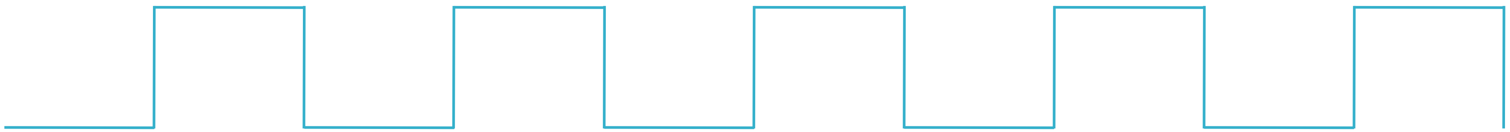
Equivalência entre representações

- Ex:
  - $1{,}5_{10} = 1{,}1_2$
  - $1{,}3125_{10} = 1{,}101_2$
  - $0{,}1_{10} = 0.0011110111001100110011001100 11\ldots_2$
  - $12_{10} = C_{16}$
  - $15_{10} = 17_8$
  - $7_8 = 111_2$
  - $27_8 = 10111_2$
  - $5A_{16} = 01011010_2$
  - $83B_{16} = 100000111011_2$

Representação numérica nos sistemas computacionais

Aritmética computacional

- Computadores são sistemas digitais



- Unidade de informação = Bit
- Bit pode assumir apenas 2 estados
  - 0 – Nível lógico baixo
  - 1 – Nivel lógico alto
- Desta forma, a base numérica natural para os sistemas computacionais é a base binária

Aritmética computacional

- A fim de facilitar a manipulação/visualização por parte do ser humano, pode adotar-se também as bases numéricas octal e hexa-decimal, a quais permitem um mapeamento direto para a base binária.
- Na base octal, ao valor representado em cada grupo de 3 bits (dígitos binário) associa-se um símbolo octal
- Na base hexa-decimal, ao valor representado em cada grupo de 4 bits (dígitos binário) associa-se um símbolo hexa-decimal

Conversão de Base

- Todo e qualquer valor inteiro representável em uma determinada base numérica pode ser livremente convertido para qualquer das outras bases numéricas definidas.
- Nem todo valor que possua uma parcela menor que a unidade em uma determinada base numérica pode ser convertido de maneira exata para as outras bases numéricas definidas.

- Ex:

$$C_{16} = 12_{10} = 14_{8} = 1100_{2}$$
$$35_{16} = 53_{10} = 65_{8} = 00110101_{2}$$
$$0{,}1_{10} = 0.00111101110011001100110011001100..._{2}$$

Algoritmo de conversão para inteiros

- Dado um numero K, inteiro, expresso na base b1 o qual deve ser convertido para um número R na base b2, proceda:

1. verifique se K < b2, neste caso K pode ser expresso diretamente na base b2, caso contrario, vá ao passo 2
2. faça K = K/b2, anote o resto desta divisão, o qual será denominando de Rn, onde n=numero de iterações do algoritmo, começando em n=0
3. verifique se K < b2, neste caso o algoritmo finaliza, sendo atribuído a R a seqüência de dígitos formado por K seguido de R_n até $R_{0,}$, caso contrario, retorne ao passo 2

Conversão de Base

Exemplos

- Conversão da base 10 para base 2
  - 1
  - 9
  - 13
  - 56
  - 125
  - 564

Conversão de Base

Algoritmo de conversão para fracionários

- Dado um numero K, fracionário, expresso na base b1, o qual deve ser convertido para um número R na base b2, proceda:
  1. Converta a parte inteira de K conforme o algoritmo anterior
  2. Verifique a parte fracionária de K, se esta for igual a zero vá ao passo 4, caso contrario vá ao passo 3
  3. faça K = K * b2, verifique a parte inteira do resultado obtido, sendo esta maior que zero, anote o valor da parte inteira de K, o qual será denominada de In, onde n=numero de iterações do algoritmo, começando em n=0. Subtraia In de K. Retorne ao passo 2.
  4. Acrescente a R obtido no algoritmo de conversão da parte inteira a seqüência de dígitos formado de I_0 à I_n. O ponto separador da parte fracionária deverá ser colocado entre o valor obtido em R e a seqüência obtida de I_0 à I_n

Conversão de Base

Exemplos

- Conversão da base 10 para base 2
  - 1,5 = 1,1
  - 9,25 = 1001,01
  - 1,6 = 1,100110011...
  - 125,0625 = 1111101,0001

Operações Aritméticas

- Todas as operações aritméticas, independente da base numérica adotada, se processam da maneira clássica, semelhante ao que o corre com a base decimal.
- Deve se observar apenas o valor no qual ocorre o “*vai um* ” e o “*vem um* ” nos dígitos da base adotada, ou seja, deve se observar quando o resultado de uma operação entre dois algarismos gera um resultado que não pode ser expresso através de um único algarismo.

Operações Aritméticas

- Valor no qual ocorre o vai um em cada base numérica
  - **Decimal**: quando o valor a ser expresso for maior que 9
  - **Binário**: quando o valor a ser expresso for maior que 1
  - **Octal**: quando o valor a ser expresso for maior que 7
  - **Hexadecimal**: quando o valor a ser expresso for maior que 15

Operações Aritméticas

- Exemplos de ocorrência de vai um em diferentes bases numéricas:
  - Base Decimal

$$
\begin{array}{r} 1 \\ 3 \\ +\ \underline{7} \\ 10 \end{array}
\qquad
\begin{array}{r} 1 \\ 5 \\ +\ \underline{8} \\ 13 \end{array}
\qquad
\begin{array}{r} 1 \\ 15 \\ +\ \underline{9} \\ 24 \end{array}
\qquad
\begin{array}{r} 11 \\ 127 \\ +\ \underline{295} \\ 422 \end{array}
$$

  - Base Binária

$$
\begin{array}{rl} 1111 & \\ 01011 & (11) \\ +\ \underline{00111} & (7) \\ 10010 & (18) \end{array}
\qquad
\begin{array}{rl} 11 & \\ 00110 & (6) \\ +\ \underline{10111} & (23) \\ 11101 & (29) \end{array}
$$

Operações Aritméticas

- Exemplos:

  $10_{16}+6_{16}=16_{16}$

  $16_{10}+6_{10}=22_{10}$ (Ocorreu um *vai um* porque 6+6>9)

  $A_{16+}\ 6_{16}\ =\ 10_{16}$ (Ocorreu um *vai um* porque A+6>15)

  $10_{10+}\ 6_{10}\ =\ 16_{10}$

  $3_8+7_8=12_8$ (Ocorreu um *vai um* porque 3+7>7)

  $4_8x2_8=10_8$ (Ocorreu um *vai um* porque 4x2>7)

  $1010_2/10_2=0101_2$

- Algumas outras perguntas ainda podem surgir:
  - Como representar números negativos?
  - Qual o maior número que pode ser representado em uma palavra de computador?
  - O que acontece se uma operação cria um número maior do que o maior valor que a palavra daquela máquina pode acomodar?

números negativos

- *Notação sinal/magnitude*
  - Cada número possui um bit adicional que representa o sinal.

    Ex.

    $$\boxed{\mathbf{1}}\mathbf{0100_2 = -4}$$

    Bit de sinal

  - Problemas
    - Duas representações para o zero.
    - A soma de um número com o seu inverso não resulta em zero.

números negativos

- *Exemplos de números em representação de magnitude e sinal*
  - 1000 = – zero
  - 0000 = zero
  - 1011 = –3
  - 0011 = 3
  - 1111 = –7
  - 0111 = 7

- Exemplo de problema da operação direta neste padrão de representação:
  - 1011 + 0011 = 1110, ou seja,

    –3 + 3 = –6 (ERRADO!!!!)

números negativos

- *Notação complemento a dois*
  - A notação de complemento a dois veio para resolver os problemas já citados da representação de magnitude e sinal.
  - Estes objetivos foram atingidos simplesmente definindo que o inverso de um número é aquele somado ao primeiro resulta em zero. Exatamente como temos na base decimal, ou seja, o inverso de 1 é -1 porque 1+(-1)=0.
  - Desta forma, temos que o inverso de zero é o próprio zero, porque 0 + 0 = 0

números negativos

- *Notação complemento a dois*
  - Da mesma forma que na representação de magnitude e sinal, a representação de complemento a dois também reserva o bit mais a esquerda para a representação do sinal do número.
  - Entretanto, diferentemente da representação em magnitude e sinal, neste caso o bit de sinal também assume um valor no cálculo da magnitude do número representado
  - De um modo geral podemos dizer que, para uma representação com n bits, o bit de sinal deve ser ponderado em -2^{n-1}

números negativos

- *Exemplos de números em notação de complemento a dois*

$000_2 = (\mathbf{0} \times 2^2) + (\mathbf{0} \times 2^1) + (\mathbf{0} \times 2^0) = 0$

$001_2 = (\mathbf{0} \times 2^2) + (\mathbf{0} \times 2^1) + (\mathbf{1} \times 2^0) = 1$

$010_2 = (\mathbf{0} \times 2^2) + (\mathbf{1} \times 2^1) + (\mathbf{0} \times 2^0) = 2$

$011_2 = (\mathbf{0} \times 2^2) + (\mathbf{1} \times 2^1) + (\mathbf{1} \times 2^0) = 3$

$100_2 = (\mathbf{1} \times -2^2) + (\mathbf{0} \times 2^1) + (\mathbf{0} \times 2^0) = -4$

$101_2 = (\mathbf{1} \times -2^2) + (\mathbf{0} \times 2^1) + (\mathbf{1} \times 2^0) = -3$

$110_2 = (\mathbf{1} \times -2^2) + (\mathbf{1} \times 2^1) + (\mathbf{0} \times 2^0) = -2$

$111_2 = (\mathbf{1} \times -2^2) + (\mathbf{1} \times 2^1) + (\mathbf{1} \times 2^0) = -1$

números negativos

- *Observações sobre a Notação complemento a dois*
  - *Em uma palavra com n bits teremos 2^n combinações, divididas em 2^{n-1} negativas, $2^{n-1} - 1$ positivas e uma representação para o 0 (zero), sendo*
    - $2^{n-1} - 1$ o maior número positivo, e
    - -2^{n-1} o menor número negativo.
  - A soma de um número com o seu inverso resulta em zero. Ex: 1101 + 0011 = 10000

Algoritimo para negação

- Considere
  - Se x é um número positivo e y é o seu inverso, temos:
    - $x + y = 0 \Rightarrow y = 0 - x$

Ex. $x = 0011_2 \Rightarrow y = 0 - 0011$

Regra prática para negação

- Assim podemos concluir que :
  - Para representar um número negativo podemos seguir os seguintes passos:
    1. Representar o número positivo
    2. Inverter os bits
    3. Somar 1 à palavra invertida
- Exemplo: Como representar o número −34 em binário?
- X = 34 = *0100010*
- Y = *1011101*
- −x = y+ 1 = 1011101 + 1 = *1011110*

Exercícios de fixação

1- Converta os seguintes números decimais em números binários de 8 bits (1 byte)

a) 57 d) -35

b) 80 e) -100

c) 125 f) - 72

2 - Converta os seguintes números binários em decimais

a) 00101011 c)01101011

b) 10110100 d)11000000

Obs. Considere notação complemento a dois

Exercícios de fixação

Efetue as seguintes operações em base decimal e em base binária e compare os resultados:

1. 3 + 4 – 5
2. 16 – 2
3. 64 – 32
4. 128 – 125

Mult. e Div.

- $2_{10} = 10_2$
- $4_{10} = 100_2$
- $16_{10} = 10000_2$

- Deslocamento para a esquerda equivale a multiplicar pela base
- Deslocamento para a direita equivale a dividir pela base

Multiplicação e Divisão

UFRPE

- Exemplos
- 6*2 => 110*10 = 1100
- 5*5 => 101 * 101 = 11001

```
    101
  * 101
  -----
    101
   000
  101
  -----
  11001
```

- 7*3 => 111 * 011 = 10101

```
    111
  * 011
  -----
    111
   111
  000
  -----
  10101
```

Exercícios de fixação

Efetue as seguintes operações em base decimal e em base binária e compare os resultados:

1. 2 * 4
2. 16 * 2
3. 64 / 4
4. 128 * 8

Verifique as seguintes conversões entre bases numéricas

- $1024_{10} = 400_{16}$
- $64533_{10} = FC15_{16}$
- $43605_{10} = AA55_{16}$
- $101000110101_2 = A35_{16}$
- $1111001101011110_2 = F35E_{16}$
- $11011101_2 = -35_{10}$
- $11000000_2 = -64_{10}$
- $10000000_2 = -128_{10}$
- $1111101,1_2 = 125,5_{10}$
- $101,00101_2 = 5,15625_{10}$

Efetue as seguintes operações em decimal e em binário e compare os resultados

- 35 – 40
- 123 – 122
- 38 + 33 – 14
- 125 + 45 – 124 – 121
- 12,35 + 122,03125
- 6,350 – 9,750
- 7,5 * 4
- 73,9375 * 3
- 12 * 2
- 4 * 5

Efetue as seguintes operações em decimal e em hexa-decimal e compare os resultados

- 100 + 128
- 235 + 72
- 9 + 7
- 37 + 524
- 125 - 45
- -83 + 123
- 432 - 400
- 123 + 77
- 43 + 42
- 37 - 73

Representação de números Racionais

- Existem duas formas de representar os números Racionais nos sistemas computacionais

    - Representação em Ponto Fixo

    - Representação em Ponto Flutuante

Representação em Ponto Fixo

- Dada uma palavra binária com n bits, reserva-se k destes bits à representação da parte fracionária e o restante destes à representação da parte inteira
- A designação de ponto fixo deriva do fato que o ponto decimal permanece fixo dividindo os dois grupos de bits
- Exemplo: n=8, k=4
    - 0011,1000 = 3,5
    - 0100,0100 = 4,25
    - 1100,0110 = 12,375

Representação em Ponto Flutuante

- A representação em ponto flutuante se assemelha à representação de notação científica, sendo formada por:
  - uma representação para a mantissa do número
  - Uma representação para o expoente
- A designação de ponto flutuante deriva do fato que o ponto decimal pode "flutuar", ou ser deslocado, da sua posição original, pela interação da mantissa com o expoente. Exatamente como ocorre na representação de notação científica
- Assim como na notação científica, todos os números devem ser representados "normalizados", ou seja, com a parte intera da mantissa diferente de zero

Representação em Ponto Flutuante

- Uma vez que os números devem ser representados normalizados, e que estamos adotando a base binária, isto significa dizer que todos os números devem ser representados com a parte inteira da mantissa igual a 1.
- Por este motivo, a parte inteira da mantissa não é armazenada juntamente com o número, uma vez que o seu valor já é conhecido a priori

Representação em Ponto Flutuante

- Um outro detalhe importante deste padrão de representação é que o expoente não é representado nem através de magnitude e sinal nem de complemento a dois, mas sim por referência de zero. Desta forma, todos os valores maiores que a referência são considerados positivos e todos os menores são considerados negativos.
- Esta referência é chamada de “bias

Representação em Ponto Flutuante

- Desta forma, uma representação de ponto flutuante é definida da seguinte forma:
    - Tamanho da palavra binária utlizada em sua representação
    - Número de bits da parte fracionária da mantissa
    - Número do bits do expoente
    - Valor do bias

Representação em Ponto Flutuante

- Existem, a princípio, 4 padrões de representação para números em ponto flutuante que são suportados pela maioria das linguagens de alto nível
  - Short (16 bits)
  - Float (32 bits)
  - Double (64 bits)
  - Extended (80 bits)

Representação em Ponto Flutuante

- A representação Float define da seguinte forma a divisão dos seus bits na representação de um número:
  - 1 bit para o sinal
  - 23 bits para a parte fracionária da mantissa
  - 8 bits para o expoente
  - Bias = 127

Representação em Ponto Flutuante

- Exemplos:

- 3,0 em ponto fluante ➔ 1,5 * 2 ➔ 1,1 * 2^1
  - Sinal = 0 (positivo)
  - Mantissa = 1,10000000000000000000000 (1,5)
  - Expoente = 10000000 (128-127=1)

- 3,0 = 01000000010000000000000000000000

Representação em Ponto Flutuante

- Exemplo:

- 4,0 em ponto fluante ➔ 1×2^2
  - Sinal = 0 (positivo)
  - Mantissa = 1,00000000000000000000000 (1,0)
  - Expoente = 10000001 (129-127=2)

- 4,0 = 10000001000000000000000000000000