Representações de Dados

Informação e Dado

- Informação
  - Designa um conhecimento ou saber
  - *Abstrato*
- Dado
  - Codificação de informação
  - Permite manipulação por um computador
  - *Concreto*

Bit

- Abreviação de *Binary Digit*
- Dado que representa a informação mais simples possível:
  - Qual a opção correta quando apenas 2 opções são possíveis?
- O que as duas “opções” denotam deve ser estabelecido por alguma convenção:
  - Ex: 0 ou 1, Verdadeiro ou Falso, branco ou preto, etc

Agregando bits

- Se o número de opções válidas excede 2, é possível representá-las usando + bits
  - 2 bits: 4 opções
    - 00 / 01 / 10 / 11
  - 3 bits: 8 opções
    - 4 opções dos 2 bits originais com o terceiro bit valendo 0
    - 4 opções dos 2 bits originais com o terceiro bit valendo 1
  - N bits: 2^N opções
- 1 **Byte** = 8 Bits → $2^8 = 256$ opções

Vantagens do sistema binário

- Computadores podem manipular eletronicamente os dois estados possíveis com facilidade:
  - Transístores que conduzem ou não,
  - Cargas eletrostáticas positivas ou negativas,
  - Polarizações magnéticas sul/norte ou norte/sul.
- Operações aritméticas podem ser facilmente implementadas
  - Ex: adição requer a implementação de apenas 4 possibilidades:
    - 0+0 = 0
    - 0+1 = 1 + 0 = 1
    - 1+1 = 0 (e “vai um”)

Codificando inteiros

- Usa-se a notação posicional comum, só que em **base 2**
  - Apenas os algarismos 0 e 1 são admitidos
  - Um bit é usado para cada algarismo
- Notação posicional:
  - Cada casa corresponde a uma potência da base
    - Da direita para esquerda: potências 0, 1, 2, … n
    - Algarismos são multiplicados pela potência correspondente e depois somados
  - Ex: 183 em base 10 significa $3\times10^0+8\times10^1+1\times10^2$
  - Ex: 101 em base 2 significa $1\times2^0+0\times2^1+1\times2^2$ $=$ 5 em base 10

Exemplos

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

- $1011_2 = 1+2+8 = 11_{10}$
- $101000_2 = 8+32 = 40_{10}$
- $111000_2 = 8+16+32 = 56_{10}$
- $25_{10} = ?_2$

Conversão de bases

- Se um número X na base B é dado por

$$X = x_{n-1} \ldots x_2 x_1 x_0$$

- Então,
  - $x_0 = X \bmod B$
  - $x_1 = (X \operatorname{div} B) \bmod B$
  - $x_2 = ((X \operatorname{div} B) \operatorname{div} B) \bmod B$
  - Etc
- Onde **div** significa divisão inteira e **mod** significa resto da divisão

Exemplo

- $25_{10} = ?_2$
  - 25 mod 2 = (1)
  - 25 div 2 = 12
  - 12 mod 2 = (0)
  - 12 div 2 = 6
  - 6 mod 2 = (0)
  - 6 div 2 = 3
  - 3 mod 2 = (1)
  - 3 div 2 = 1
  - 1 mod 2 = (1)

$$= 1\times2^0+0\times2^1+0\times2^2+1\times2^3+1\times2^4$$

$$= 11001_2$$

Base Octal

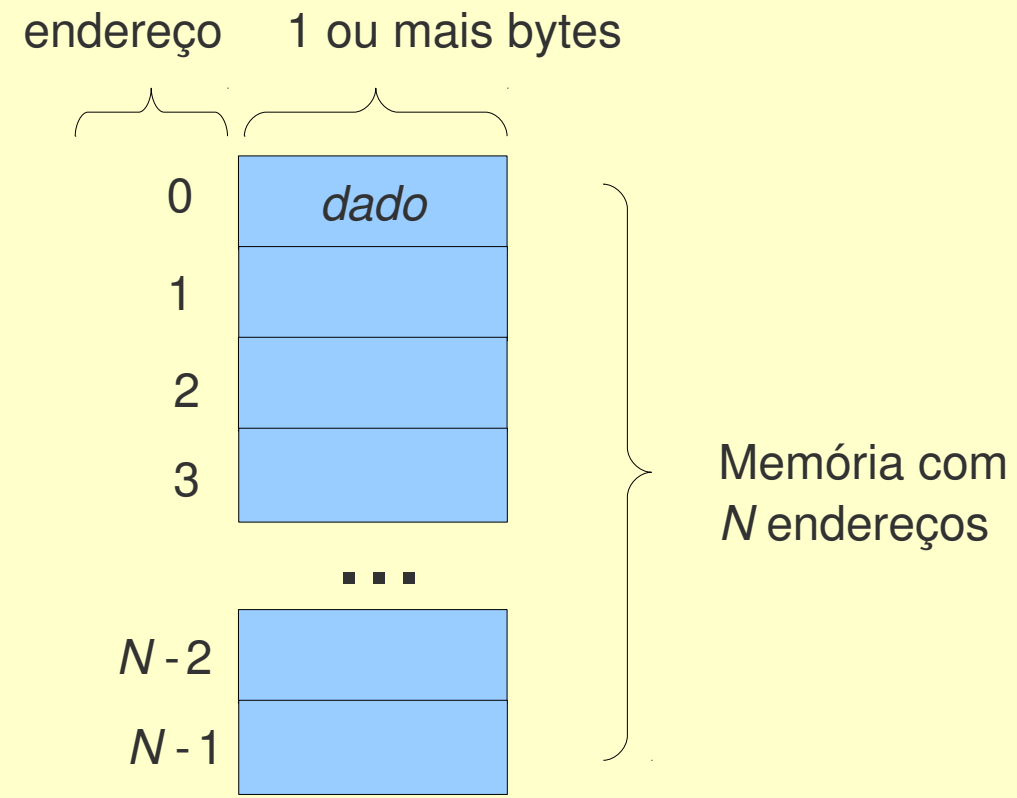
- Algumas bases que são potências de dois são frequentemente usadas para indicar configurações de bits
- A base 8 usa os dígitos de 0 a 7, sendo que cada dígito octal corresponde a 3 dígitos binários:
  - $0_8 = 000_2$
  - $1_8 = 001_2$
  - $2_8 = 010_2$
  - $3_8 = 011_2$
  - $4_8 = 100_2$
  - $5_8 = 101_2$
  - $6_8 = 110_2$
  - $7_8 = 111_2$
- Exemplos:
  - $163_8 = 001\ 110\ 011_2$
  - $527_8 = 101\ 010\ 111_2$

Base Hexadecimal

- Um dígito da base hexadecimal (base 16 = 2^4) corresponde a 4 dígitos binários
- Além dos 10 algarismos arábicos (0 a 9) usa-se as letras A a F para denotar os “algarismos” 10 a 15
  - $A_{16} = 1010_2$
  - $B_{16} = 1011_2$
  - $C_{16} = 1100_2$
  - $D_{16} = 1101_2$
  - $E_{16} = 1110_2$
  - $F_{16} = 1111_2$
- Exemplos:
  - $A2F1_{16} = 1010\ 0010\ 1111\ 0001_2$
  - $1CEE_{16} = 1\ 1100\ 1110\ 1110_2$

Memória

- A memória de um computador corresponde a uma certa quantidade de bytes *endereçáveis*
  - O endereço de um byte é sua localização dentro da memória



Memória

- Bytes são lidos e gravados na memória em grupos chamados de *palavras de memória*
  - Uma palavra é a quantidade que pode ser transportada de uma vez (por ciclo de máquina) de / para a memória
- É comum variáveis simples ocuparem exatamente uma ou duas palavras de memória
  - Ex.: inteiros em arquiteturas de 32 bits ocupam 4 bytes (4 x 8 = 32 bits)

Capacidade de memória

- É medida em múltiplos de bytes
  - 1024 Bytes = 1 Kilobyte
  - 1024 Kilobytes = 1 Megabyte
  - 1024 Megabytes = 1 Gigabyte
  - 1024 Gigabytes = 1 Terabyte
  - 1024 Terabytes = 1 Petabyte
  - 1024 Petabytes = 1 Exabyte
  - 1024 Exabytes = 1 Zettabyte
  - 1024 Zettabytes = 1 Yottabyte
  - 1024 Yottabytes = 1 Brontobyte
  - 1024 Brontobytes = 1 Geopbyte
- Quando se trata de memória secundária (disco), os múltiplos são contados ligeiramente diferente:
  - 1000 Bytes = 1 Kilobyte
  - 1000 Kilobytes = 1 Megabyte
  - 1000 Megabytes = 1 Gigabyte
  - 1000 Gigabytes = 1 Terabyte
  - 1000 Terabytes = 1 Petabyte
  - 1000 Petabytes = 1 Exabyte
  - 1000 Exabytes = 1 Zettabyte
  - 1000 Zettabytes = 1 Yottabyte
  - 1000 Yottabytes = 1 Brontobyte
  - 1000 Brontobytes = 1 Geopbyte

Inteiros negativos

- Para indicar se um número inteiro é positivo ou negativo, pode-se usar um bit extra (*bit de sinal*)
  - Se bit = 0: número positivo
  - Se bit = 1: número negativo
- Um problema disso é que o zero tem duas representações válidas

Complemento a dois

- O esquema mais usado para representar inteiros negativos
- Numa representação com n bits,
  - n – 1 bits representam um valor positivo entre 0 e $2^{n-1} - 1$
  - O n-ésimo bit, se igual a 1, indica que -2^{n-1} tem que ser somado ao total
- Exemplo com 3 bits:
  - 2 bits representam um valor positivo entre 0 e 3
  - Se o terceiro bit é 1, então –4 tem que ser somado
    - $000_2 = 0$
    - $001_2 = 1$
    - $010_2 = 2$
    - $011_2 = 3$
    - $100_2 = -4$
    - $101_2 = -3$
    - $110_2 = -2$
    - $111_2 = -1$

Complemento a dois

- Qual a representação complemento a 2 de -19 (com 6 bits)?
- Começamos com o valor mais negativo possível e somamos as potências positivas sucessivas até encontrar o valor desejado
  - 100000 é -32
  - 110000 é -32+16 = -16 (somamos demais!)
  - 101000 é -32+8 = -24
  - 101100 é -32+8+4 = -20
  - 101110 é -32+8+4+2= -18 (somamos demais!)
  - 101101 é -32+8+4+1= -19 (OK!)

Intervalo de representação

- Se n bits são usados para codificar apenas inteiros *positivos*, então é possível representar valores entre 0 e $2^n - 1$
  - Ex.: com 16 bits é possível representar inteiros no intervalo $[0, 2^{16} - 1] = [0, 65535]$
- Usando complemento a 2, é possível representar inteiros entre -2^{n-1} e $2^{n-1}-1$
  - Ex.: com 16 bits é possível representar inteiros no intervalo $[-2^{15}, 2^{15}-1] = [-32768, 32767]$

Ponto flutuante

- É a codificação mais empregada para representar números fracionários
- Semelhante à notação científica (base decimal):

$$234.45 = 0.23445 \times 10^{3}$$

- Mantissa e expoente são representados em binário

$$1101.101 = 0.1101101 \times 2^{100}$$

  - O ponto é movido para a esquerda da 1a casa não nula (4 casas no exemplo)
  - A potência de 2 correspondente é registrada no expoente (no exemplo: $4_{10} = 100_{2}$)

Padrão IEEE 754

- Especificação de ponto flutuante de 32 bits
  - Expoente em 8 bits representando potências de 2 no intervalo [-126,127]
  - Mantissa em 23 bits + 1 bit de sinal
- Especificação de ponto flutuante de 64 bits
  - Expoente em 11 bits representando potências de 2 no intervalo [−1022, 1023]
  - Mantissa em 52 bits + 1 bit de sinal
- Algumas configurações são usadas para representar valores especiais tais como +inf, -inf, NaN, -0
  - Ajudam a lidar com valores extremos

Precisão de ponto flutuante

- Números de ponto flutuante de 32 bits têm aproximadamente
  - 9 dígitos decimais de precisão
  - Menor positivo não nulo $\sim 10^{-38}$
  - Maior positivo $\sim 10^{38}$
- Nem todas as frações podem ser representadas exatamente
  - Por exemplo, o número 0.1 decimal é uma dízima periódica em binário
    - O número realmente armazenado em 32 bits é 0.100000001490116119384765625

Texto

- Caracteres de texto são codificados em bytes usando uma convenção chamada de *tabela de codificação* (*character code* em inglês)
- O precursor de todas as tabelas modernas é o ASCII (American Standard Code for Information Interchange)
  - Cada caractere ocupa 1 byte
  - 26 letras maiúsculas (códigos 65 a 90) e 26 minúsculas (97 a 122)
  - 10 algarismos arábicos (48 a 57)
  - ~20 sinais de pontuação e caracteres gráficos
  - 32 caracteres *de controle* (0 a 31)
    - Não têm marca gráfica (*não imprimíveis*)
    - Usados originalmente para controlar o fluxo de informação textual

      Ex: Line Feed (10) / Carriage Return (13) / Tab (9) / Backspace (8)

Extensões do ASCII

- Outras tabelas baseadas no ASCII são usadas para suportar letras latinas acentuadas e outros símbolos
  - ISO-8859-1 a ISO-8859-15 → Diversos padrões para línguas ocidentais
  - Windows CP-1252
- Línguas com caracteres não latinos (Cirílico, caracteres de línguas asiáticas) são suportadas por outras tabelas específicas

Unicode

- O desejo de padronizar a representação de caracteres de todos os tipos levou à criação do padrão Unicode (www.unicode.org)
- O padrão completo contém mais de 100 mil caracteres
- É mais comumente implementado usando as tabelas UTF-8 (8 bits) e UTF-16 (16 bits)
  - UTF = *Unicode Transformation Format*
- UTF-8 é a mais empregada
  - Compatível com ASCII
    - Códigos 0-127 denotam os mesmos caracteres
  - Demais caracteres são representados usando até 3 bytes adicionais