

Departamento de Engenharia de Sistemas e Computação

Representação da Informação

Binário, hexadecimal, números com sinal, ponto flutuante e caracteres

---

João Araujo

Algoritmos Computacionais I

Aula 02

Roteiro

- Sistemas de numeração posicionais (Cap. 1.1)
- Números binários, prefixos kilo/mega/giga (1.2)
- Octal e hexadecimal (1.3), Kilo e Kibi (1.4)
- Binário sinalizado: complemento a 2 (1.5)
- Números reais: ponto flutuante IEEE 754 (1.6)
- Caracteres: ASCII e Unicode (1.7)

Referência: Cap. 1 (1.1–1.7) do livro-texto.

Notação posicional

Em base 10, cada algarismo vale uma potência de 10 pela sua posição:

$$5532 = 5 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$$

- O dígito mais à direita é o menos significativo; o mais à esquerda, o mais significativo
- Outras bases existem: os babilônios usavam base 60 (e até hoje: 17h 12min 35s)
- Em uma base b usamos b símbolos e o valor é $\sum d_i \, b^i$

Exercícios 1.1 e 1.2 (site)

E se tivéssemos 8 dedos? E 12 dedos? Como contaríamos e escreveríamos os números?

Por que o computador usa binário?

- Circuitos eletrônicos distinguem bem **dois** estados: ligado/desligado, tensão alta/baixa
- Um dígito binário é um **bit** (*binary digit*)
- 8 bits formam um **byte**
- Com n bits representamos 2^n valores diferentes

Exemplo

10 bits $\Rightarrow 2^{10} = 1024$ valores, de 0 a 1023

(Exercício 1.3)

Binário ↔ decimal

Binário para decimal: some as potências de 2 das posições com bit 1.

$$1011_2 = 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 11_{10}$$

Decimal para binário: divisões sucessivas por 2; leia os restos de baixo para cima.

$$\begin{array}{rll} 13 \div 2 & = 6 & \text{resto } \mathbf{1} \\ 6 \div 2 & = 3 & \text{resto } \mathbf{0} \\ 3 \div 2 & = 1 & \text{resto } \mathbf{1} \\ 1 \div 2 & = 0 & \text{resto } \mathbf{1} \end{array} \quad \Rightarrow \quad 13_{10} = 1101_2$$

Exercícios 1.4 e 1.5 (conversões nos dois sentidos).

Conferindo no Python

```
>>> bin(13)             # decimal -> binário
'0b1101'
>>> int('1101', 2)      # binário -> decimal
13
>>> 0b1101 + 0b11       # literais binários
16
>>> 2**10
1024
```

Atenção

O Python é uma *calculadora de conferência*: converta à mão primeiro (a prova será no papel) e depois verifique no interpretador.

Prefixos: kilo, mega, giga...

Prefixo	Símbolo	Decimal (SI)	Binário (IEC)
kilo / kibi	k / Ki	10^3	$2^{10} = 1024$
mega / mebi	M / Mi	10^6	2^{20}
giga / gibi	G / Gi	10^9	2^{30}
tera / tebi	T / Ti	10^{12}	2^{40}
peta / pebi	P / Pi	10^{15}	2^{50}

Por que um disco de "1 TB" aparece com 931 GiB?

O fabricante usa 10^{12} bytes; o sistema operacional divide por 2^{30}. Veja os Exercícios 1.10 e 1.11 (velocidade de internet e disco externo).

Octal e hexadecimal

- **Octal** (base 8): 3 bits por dígito
- **Hexadecimal** (base 16): 4 bits por dígito; símbolos 0–9 e A–F
- Mais compactos que o binário e fáceis de converter

Dec	Bin	Oct	Hex
0	0000	0	0
7	0111	7	7
9	1001	11	9
10	1010	12	A
15	1111	17	F

Truque

Agrupe os bits de 4 em 4 a partir da direita: $1101\,0110_2 = \mathrm{D6}_{16}$. Exercícios 1.7 a 1.9.

Hexadecimal na prática

```
>>> hex(255)
'0xff'
>>> oct(64)
'0o100'
>>> int('D6', 16)
214
>>> 0xD6 == 0b11010110
True
```

- Cores em páginas web: #FF8800 = vermelho 255, verde 136, azul 0
- Endereços de memória e códigos de erro costumam aparecer em hexadecimal

Números com sinal: o problema

Como representar -5 usando só 0s e 1s? O livro apresenta quatro técnicas (1.5):

1. **Sinal/valor**: o primeiro bit é o sinal (1 = negativo)
2. **Excesso–N**: soma-se um deslocamento fixo
3. **Complemento a 1**: inverte-se todos os bits
4. **Complemento a 2**: inverte-se e soma-se 1 ← usada nos computadores

Por que complemento a 2?

Só existe um zero e a **soma funciona sem tratar o sinal**: o mesmo circuito soma positivos e negativos.

Complemento a 2 com 8 bits

Como obter -5:

$+5$	0000 0101
inverter os bits	1111 1010
somar 1	1111 1011 $= -5$

- Faixa com n bits: de -2^{n-1} a $2^{n-1}-1$ (8 bits: -128 a 127)
- Soma $5+(-5)$: 0000 0101 + 1111 1011 = 1 0000 0000; descartando o *vai-um* sobra 0000 0000
- Overflow: $127+1$ em 8 bits resulta em -128

Exercícios 1.12 a 1.15 (para casa): converter ± 215 e somar $120+8$ em binário.

Números reais: ponto fixo e ponto flutuante

- **Vírgula fixa:** parte inteira e parte fracionária com número fixo de bits (pouca faixa)
- **Ponto flutuante (IEEE 754):** notação científica em binário

$$\text{valor} = (-1)^s \times 1.m \times 2^e$$

Precisão	Sinal	Expoente	Mantissa
Simples (32 bits)	1	8	23
Dupla (64 bits)	1	11	52

O `float` do Python usa precisão dupla. Exercícios 1.16 a 1.18.

Consequência: $0{,}1$ não é exato

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
>>> abs((0.1 + 0.2) - 0.3) < 1e-9
True
```

- $0{,}1$ é uma dízima periódica em binário: o computador guarda uma aproximação
- Nunca compare `float` com ==; compare com uma tolerância

Fonte: `code/Cap2/prog.2.7_erropf.py`

Representação de caracteres

- Cada caractere recebe um **código numérico**
- **ASCII** (7 bits, 128 símbolos): letras sem acento, dígitos, pontuação
- **Unicode**: mais de 140 mil símbolos de todos os idiomas; **UTF-8** é a codificação mais usada

Caractere	A	B	a	0	espaço
Código (dec)	65	66	97	48	32

Caracteres no Python

```
>>> ord('A')        # caractere -> código
65
>>> chr(97)         # código -> caractere
'a'
>>> chr(ord('a') - 32)
'A'
>>> 'ação'
'ação'
```

- Diferença entre maiúscula e minúscula: 32 (apenas um bit)
- Strings do Python 3 são Unicode: `'ç'` e `'ã'` funcionam normalmente

Para praticar: exercícios do livro

Site do livro – Capítulo 1

- Bases: **1.1, 1.2** (outras contagens) e **1.3 a 1.9** (binário, octal e hexadecimal)
- Unidades: **1.10, 1.11** (Mbits/s, disco externo)
- *Para casa:* **1.12 a 1.15** (complemento a 2) e **1.16 a 1.18** (IEEE 754)

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap1/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Computadores usam binário; n bits representam 2^n valores
- Octal e hexadecimal agrupam 3 e 4 bits
- Inteiros com sinal: complemento a 2; reais: IEEE 754 (aproximações!)
- Caracteres são números: ASCII e Unicode
- Próxima aula: arquitetura de von Neumann e o ambiente Python