>>> Programação Orientada a Objetos (POO)

... Armazenamento

Prof: André de Freitas Smaira

[~]$ _

>>> Outros Exemplos

```cpp
#include <iostream>

int main() {
    double b, c;
    std::cout << "Valores a multiplicar: ";
    std::cin >> b >> c;
    std::cout << "Produto: " << b*c
    << std::endl;
    return 0;
}
```

>>> Outros Exemplos

Mas e se eu quisesse escrever o produto com exatamente 4 casas decimais?

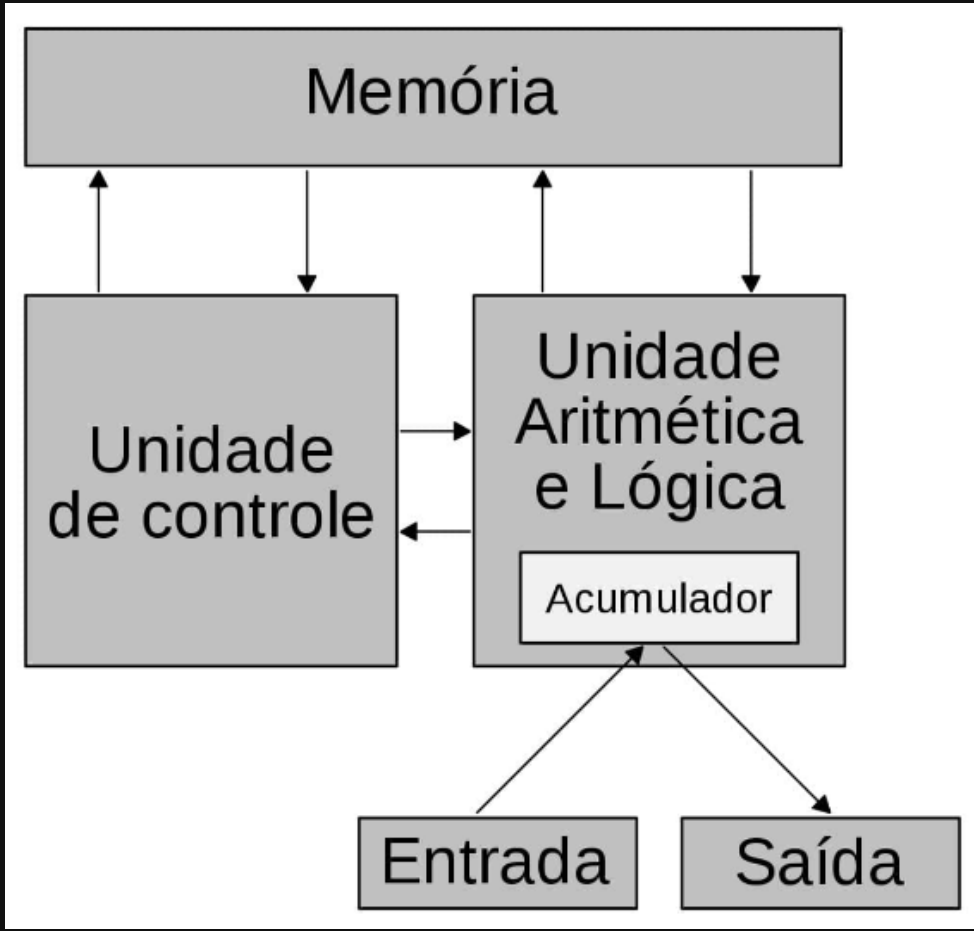
```
#include <iostream>
#include <iomanip>

int main() {
    double b, c;
    std::cout << "Valores a multiplicar: ";
    std::cin >> b >> c;
    std::cout << std::fixed << std::setprecision(4);
    std::cout << "Produto: " << b*c
    << std::endl;
    return 0;
}
```

>>> Armazenamento

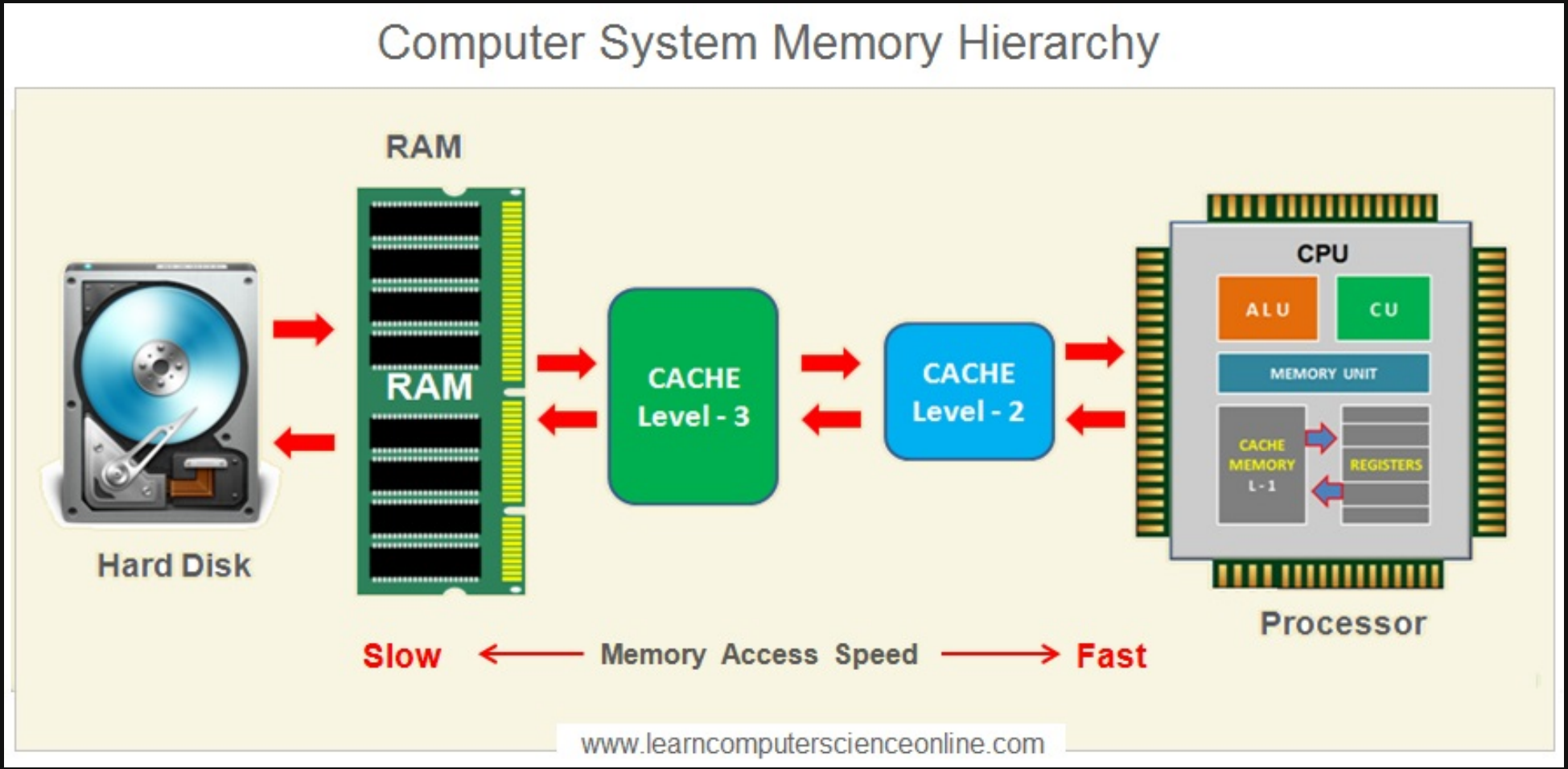
>>> Introdução - Computador Básico

Modelo de Von Neumann



* **Entrada**: dados recebidos pelo programa
* **Saída**: dados fornecidos pelo programa
* **UAL**: cálculo
* **UC**: próxima operação a ser executada
* **Memória**: armazenamento temporário

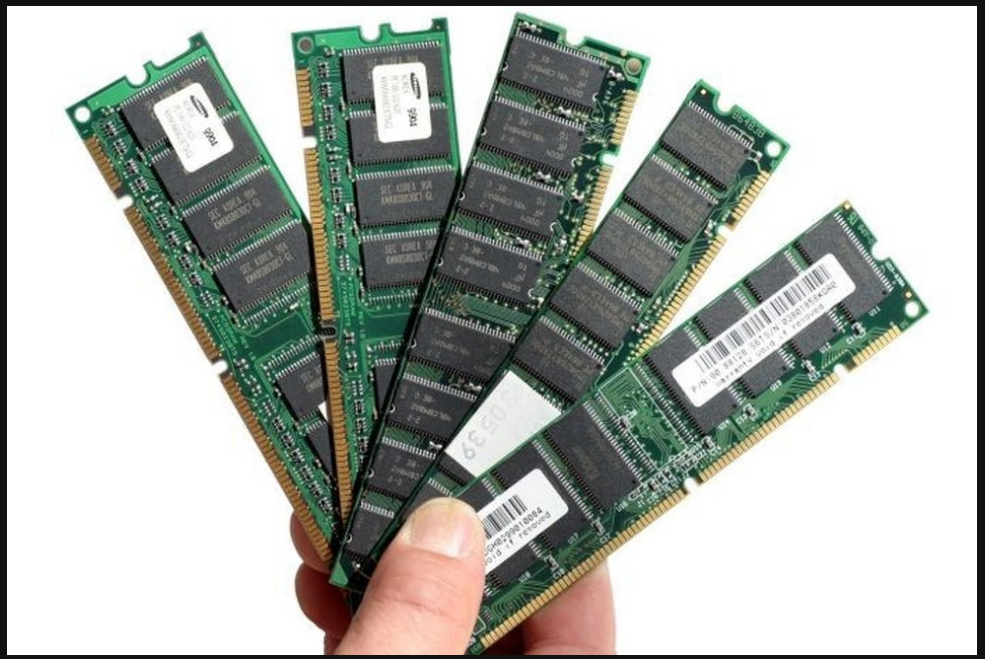
>>> Memória


Computer System Memory Hierarchy
RAM
RAM
CACHE Level - 3
CACHE Level - 2
CPU
ALU
CU
MEMORY UNIT
CACHE MEMORY L-1
REGISTERS
Hard Disk
Processor
Slow
Memory Access Speed
Fast
www.learncomputerscienceonline.com

>>> Memória Cache

* Mais próxima do processador
* Muito rápida
* Alto custo => pequena
* Volátil

>>> Memória Principal



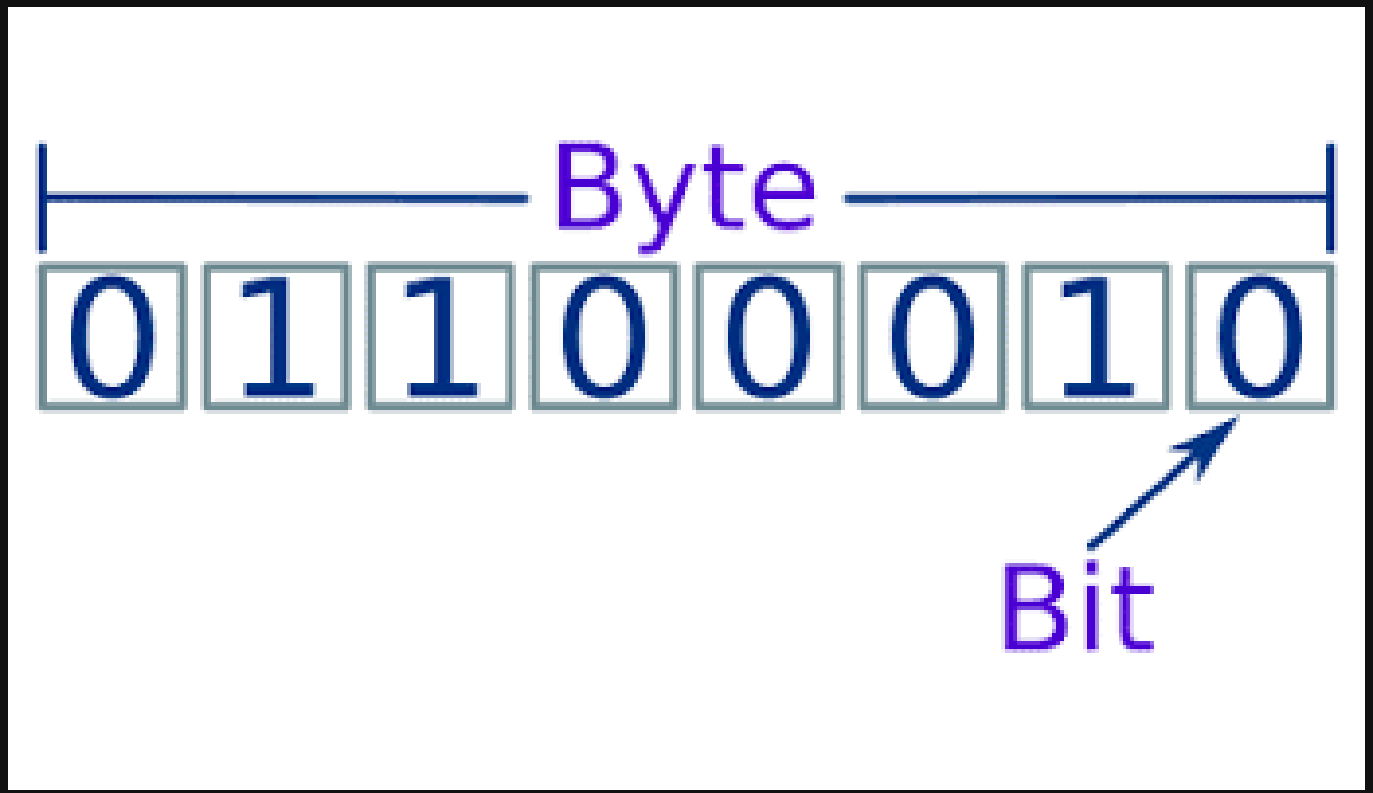
* Dados que não cabem na **memória cache**
* Mais lenta
* Custo inferior => maior
* Volátil

>>> Memória Auxiliar

* Dados que não cabem na **memória principal**
* Extremamente lenta
* Custo baixo => bem grande
* Não volátil
* Dados e programas devem ser movidos para a memória principal para serem processados
* HD, SSD, CD, DVD, pen-drive, etc

>>> Armazenamento

* BIT: dígitos binários (0 e 1)
* BYTE (8 bits): unidade básica de informação



>>> Números Decimais (base 10)

* Sistema de **numeração posicional**
* Sistema decimal (dia-a-dia): exemplo 2562

$$
\begin{aligned}
2562_{10} &= 2000 + 500 + 60 + 2 \\
&= 2 \cdot 1000 + 5 \cdot 100 + 6 \cdot 10 + 2 \cdot 1 \\
&= 2 \cdot 10^3 + 5 \cdot 10^2 + 6 \cdot 10^1 + 2 \cdot 10^0
\end{aligned}
$$

>>> Números Binários (base 2)

* Sistema de **numeração posicional**
* Sistema binário: exemplo 11010

$$
\begin{aligned}
11010_2 &= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\
&= 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 \\
&= 16 + 8 + 2 = 26_{10}
\end{aligned}
$$

* Fizemos a conversão de **base 2 para base 10**: 11010 -> 26
* E pra voltar?

>>> Números Binários (base 2)

* 26 decimal para base 2:

$$26/2 = 13 \text{ (resto 0)}$$
$$13/2 = 6 \text{ (resto 1)}$$
$$6/2 = 3 \text{ (resto 0)}$$
$$3/2 = 1 \text{ (resto 1)}$$
$$1/2 = 0 \text{ (resto 1)}$$

* Fizemos a conversão de **base 10 para base 2**: 26 -> 11010

>>> Bases de numeração

* Todos os números de **2 a infinito podem ter suas bases** de numeração
* As mais usadas:
    * **10 (decimal)**: dia-a-dia
        * 10 dígitos: 0 1 2 3 4 5 6 7 8 9
    * **2 (binária)**: computadores
        * 2 dígitos: 0 1
    * **16 (hexadecimal)**: computação
        * 16 dígitos: 0 1 2 3 4 5 6 7 8 9 A B C D E F
    * **64 (??)**: transmissão de dados
        * 64 dígitos: números, letras maiúsculas, letras minúsculas, + /

>>> Conversão Bases

* Decimal -> **Hexadecimal**

$$62011/16 = 3875 \text{ (resto 11 ou B)}$$
$$3875/16 = 242 \text{ (resto 3)}$$
$$242/16 = 15 \text{ (resto 2)}$$
$$15/16 = 0 \text{ (resto 15 ou F)}$$

* $62011_{10} = F23B_{16}$
* Esse processo se aplica a **qualquer transformação decimal -> base**

>>> Conversão Bases

* **Hexadecimal** -> Decimal

$$
\begin{aligned}
F23B &= F \cdot 16^3 + 2 \cdot 16^2 + 3 \cdot 16^1 + B \cdot 16^0 \\
&= 15 \cdot 4096 + 2 \cdot 256 + 3 \cdot 16 + 11 \cdot 1 \\
&= 61440 + 512 + 48 + 11 = 62011
\end{aligned}
$$

* $F23B_{16} = 62011_{10}$
* Esse processo se aplica a **qualquer transformação base -> decimal**

>>> Memória - Unidades de medida

* Kilobyte (kB) - 1024 bytes
* Megabyte (MB) - 1024^2 bytes
* Gigabyte (GB) - 1024^3 bytes
* Terabyte (TB) - 1024^4 bytes

>>> Entrada e Saída

* Tudo que permite interação com o usuário

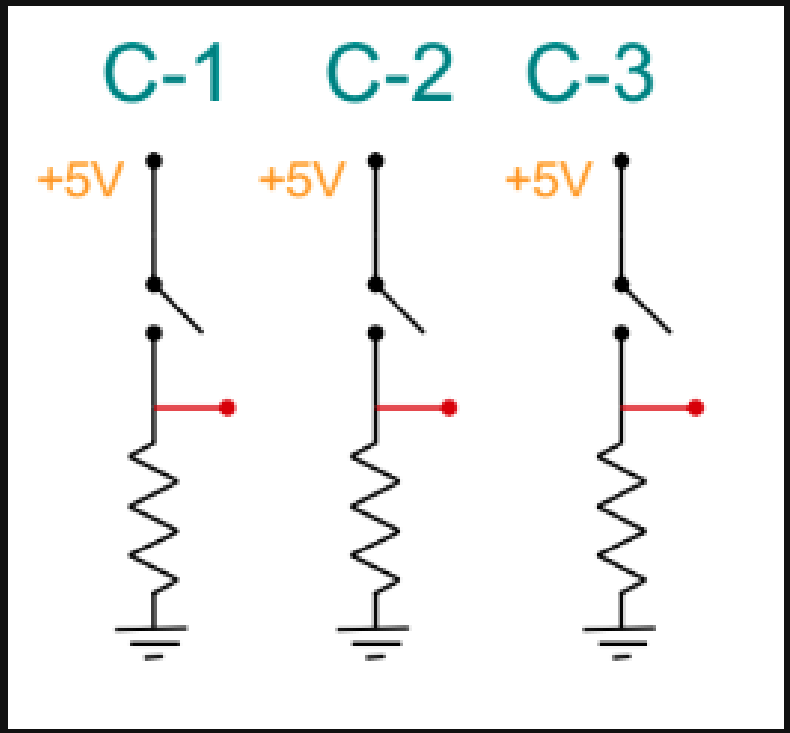
>>> Unidade Central de Processamento (CPU)

* Aqui são executadas as instruções
    * **Instrução**: Comando que define uma operação
    * **Programa**: Instruções ordenadas de forma lógica
* CPU tem duas subunidades
    * **Unidade de Controle**: controla execução e interpretação dos dados a serem processados
    * **Unidade lógica e aritmética**: recebe os dados da memória para processá-los quando uma instrução aritmética ou lógica é executada

>>> Sistemas Analógico e Digital

* **Analógico**: valores são variações proporcionais de um sinal de representação
* **Digital**: valores são um conjunto finito de símbolos que representam quantidades predefinidas (1, 2, 3, etc)

>>> Sistemas Analógico e Digital



C-1	C-2	C-3	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

```
ian@aspireE1571:/tmp$ xxd /bin/bash | grep cafe
00045490: ffe9 cafe ffff 662e 0f1f 8400 0000 0000  ......f.........
00052e60: 8628 feff ff48 85c0 0f84 cafe ffff 89c7  .(...H..........
00069b10: 488b 7b08 e8d7 cafe ff48 8943 0848 8b5b  H.{......H.C.H.[
00079b10: e7e8 9af6 ffff a802 0f84 cafe ffff e9aa  ................
0009f2d0: 8b05 cafe 2600 85c0 7426 85ff 488d 051d  ....&...t&..H...
000a0450: 562a 0400 4531 ffe9 cafe ffff 0f1f 4000  V*..E1........@.
000a0740: 0249 89ed 4c89 7c24 08e9 cafe ffff 6690  .I..L.|$......f.
000a86f0: 0f85 cafe ffff 5b31 c05d 415c 415d 415e  ......[1.]A\A]A^
000c91c0: 85d2 0f8e cafe ffff 4183 e41f 4901 c483  ........A...I...
000cafe0: 0800 3b00 0a00 0b00 0c00 0d00 ffff ffff  ..;.............
```

Organizando dados na memória

>>> A memória

* Fita com espaços de 1 Byte

Endereço	8-bits
0x0000000:	00000000
0x0000001:	00001000
0x0000002:	01000100
0x0000003:	01111100
0x0000004:	00010110
...	...
0xfffffff:	00000000

Tem aproximadamente 2 GB de memória aqui xD

* **Alocar** variável = **pedir ao SO** alguns destes espacinhos

>>> Como e o quê é armazenado ali?

* Uma sequência de dígitos em binário
* Usarmos decimal => arbitrário

>>> Operações em binário para humanos

soma	produto	subtração	divisão
100 + 110 --- 1010	100 x 110 --- 000 100 100 --- 11000	100 - 110 --- - 010	100 |_10_ 10 10 00 0

[3. Fui tapeado! O que escondemos de vocês]$ _

>>> Como o computador representa números e faz conta

* Inteiros com sinal: Complemento de 2
* Ponto flutuante: IEEE754

>>> Complemento de 2

* Vamos imaginar um computador com 4-bits => 16 números

0: 0 0 0 0	4: 0 1 0 0	8: 1 0 0 0	c: 1 1 0 0
1: 0 0 0 1	5: 0 1 0 1	9: 1 0 0 1	d: 1 1 0 1
2: 0 0 1 0	6: 0 1 1 0	a: 1 0 1 0	e: 1 1 1 0
3: 0 0 1 1	7: 0 1 1 1	b: 1 0 1 1	f: 1 1 1 1

* E os negativos?

>>> Jeito mais direto

* **Metade** para os não-negativos e a outra para os negativos...

0: 0 0 0 0	4: 0 1 0 0	-0: 1 0 0 0	-4: 1 1 0 0
1: 0 0 0 1	5: 0 1 0 1	-1: 1 0 0 1	-5: 1 1 0 1
2: 0 0 1 0	6: 0 1 1 0	-2: 1 0 1 0	-6: 1 1 1 0
3: 0 0 1 1	7: 0 1 1 1	-3: 1 0 1 1	-7: 1 1 1 1

* Problemas:
  * O **ZERO** aparece duas vezes
  * **Somar** não mantém o algoritmo

>>> Complemento de 2

* **Resolve** os problemas

0: 0 0 0 0	4: 0 1 0 0	-8: 1 0 0 0	-4: 1 1 0 0
1: 0 0 0 1	5: 0 1 0 1	-7: 1 0 0 1	-3: 1 1 0 1
2: 0 0 1 0	6: 0 1 1 0	-6: 1 0 1 0	-2: 1 1 1 0
3: 0 0 1 1	7: 0 1 1 1	-5: 1 0 1 1	-1: 1 1 1 1

* Os circuitos elétricos mais simples
* A - B = A + (-B)
* -C = ~C+1

Exemplo 1:

1 - 7 = -6
0001 - 0111
0001 + (~0111 + 0001)
0001 + (1000 + 0001)
0001 + 1001 = 1010

Exemplo 2:

Quanto é 7 + 7 aqui?
0111 + 0111 = 101000
Como só tem 4 bits:
7 + 7 = -8 (Overflow xD)

>>> O retorno do tipo de dados

* **Quantos bytes** são usados para cada tipo?

```cpp
#include <iostream>
int main()
{
    int a;
    auto b = 1;
    auto c = 1.0;
    auto d = 'c';
    auto e = 1ULL;
    std::cout << "Um inteiro tem " << sizeof(int) << " bytes\n";
    std::cout << "Um char    tem " << sizeof(char) << " bytes\n";
    std::cout << "Um float   tem " << sizeof(float) << " bytes\n";
    std::cout << "Um double  tem " << sizeof(double) << " bytes\n";

    std::cout << "a          tem " << sizeof(a) << " bytes\n";
    std::cout << "1          tem " << sizeof(b) << " bytes\n";
    std::cout << "1.0        tem " << sizeof(c) << " bytes\n";
    std::cout << "'c'        tem " << sizeof(d) << " bytes\n";
    std::cout << "1ULL       tem " << sizeof(e) << " bytes\n";
    return 0;
}
```

>>> Overflow dos inteiros

```cpp
#include <iostream>
int main()
{
	int i = 46685435; // Numero qualquer
	// Na memória (inteiro com sinal de 4 Bytes)
	// 00000010   11001000   01011100   11111011
	// Em hex: 02 c8 5c fb
	int j = 4294967295;     // 2^32 - 1
	int k = 4294967295 + 1; // 2^32
	std::cout << i << " " << j << " " << k << std::endl;
	// Troque int por unsigned int (sem sinal)
	// Use o "%u" para imprimir o tipo unsigned int
	return 0;
}
```

>>> Representações de inteiros

* **unsigned**: sem sinal
* **signed**: complemento de 2
* **short**, **int**, **long** e **long long**: **signed**, a menos que se indique **unsigned**
* **char**: varia com o compilador

>>> Ponto flutuante

* Valores com **casas decimais** e **notação científica**
* 3 partes:
    * **Sinal (S)**: positivo ou negativo
    * **Mantissa (M)**: valor base
    * **Expoente (E)**: potência de 2 na multiplicação da mantissa

$$(-1)^S \times M \times 2^E$$

>>> Referências

* Apostila e Aulas do Gonzalo Travieso (IFSC/USP)
* Aulas do Grupo Maratona IFSC (Ian Giestas Pauli e eu)