

# Software Básico (INF1018) 2012.1

<http://www.inf.puc-rio.br/~inf1018>

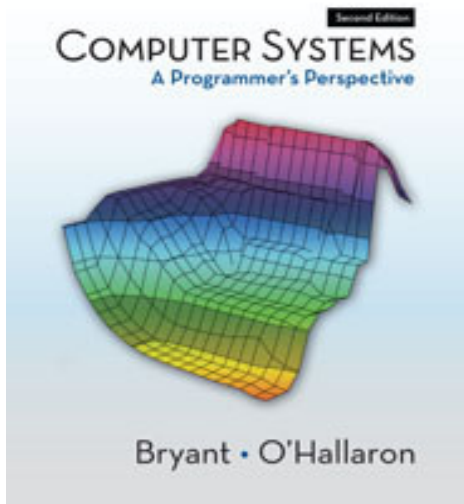
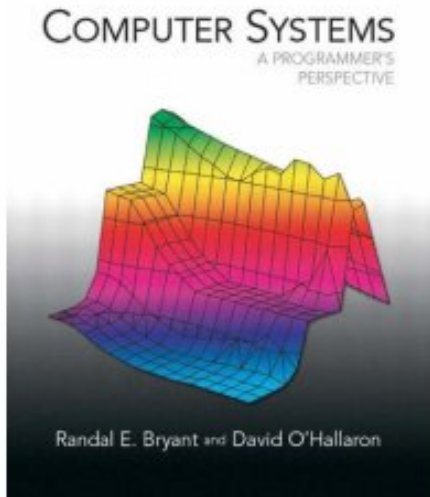
Turma 3WA – Noemi Rodriguez

([noemi@inf.puc-rio.br](mailto:noemi@inf.puc-rio.br))

Turma 3WB – Ana Lúcia de Moura

([amoura@inf.puc-rio.br](mailto:amoura@inf.puc-rio.br))

# Material básico de referência



- Computer Systems, A Programmer's Perspective. Randal Bryant and David O'Hallaron. Prentice Hall.
  - primeira edição 2003 (CS:APP)
  - segunda edição 2010 (CS:APP2e)
  - <http://csapp.cs.cmu.edu/>
- Resumos, slides e exercícios no site do curso  
<http://www.inf.puc-rio.br/~inf1018>

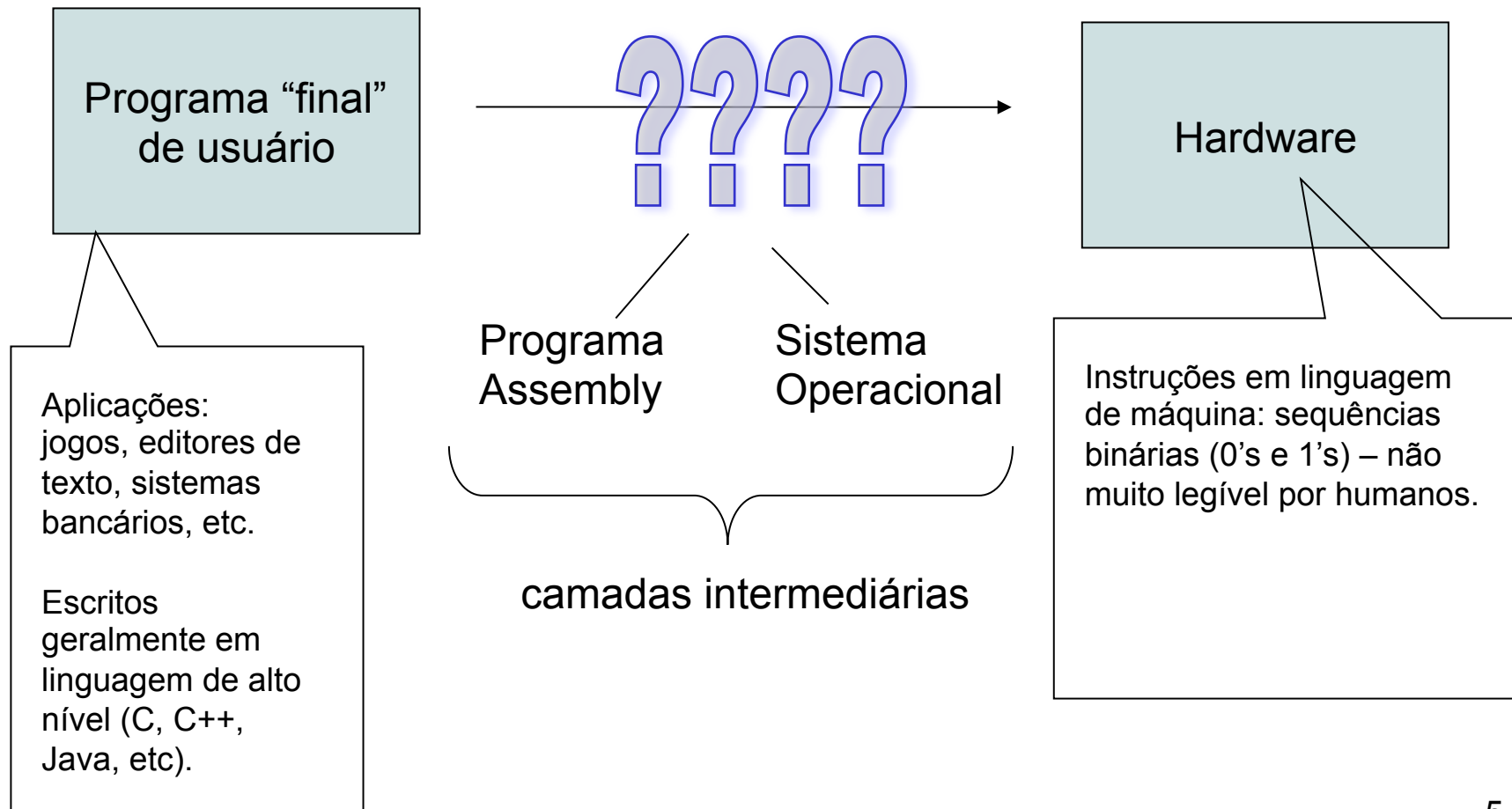
# Critério de avaliação

- Cada grau é calculado como a média geométrica de uma prova (peso 2) e um trabalho (em dupla)
  - $G1 = (P1^2 \times T1)^{\{1/3\}}$
  - $G2 = (P2^2 \times T2)^{\{1/3\}}$
- $M = (G1 + G2)/2$ 
  - Se **G1 e G2 ≥ 3.0** e **M ≥ 6.0**, M é a nota final (NF)
  - Caso contrário:
    - $NF = (G1 + G2 + 2xPF)/4$


# Objetivo do curso

- Entender como funciona um computador típico, visto pelo nível de Linguagem de Montagem e pela Linguagem de Máquina
  - foco no **programador**, e não no projetista de computadores
- Idéia é compreender como os componentes básicos de hw e sw de um sistema de computação funcionam, oferecendo o suporte à hierarquia de abstrações implementada por esse sistema

# Hierarquia de abstrações em um sistema de computação



# Linguagem Assembly

- A linguagem de montagem (assembly language) é um mapeamento bastante direto da linguagem de máquina, mas com várias facilidades para o programador.
  - uso de *mnemônicos* para representar as instruções
    - uso de “nomes” das instruções
  - cada linha do código fonte possui apenas uma instrução para o processador (CPU)
    - ex.: `mov %eax, %edx`
      -   
**mnemônico**

# Assembly e Assembler

- Um programa **assembly** fica acima da camada do sistema operacional, podendo fazer chamadas a ele para requisitar serviços, por exemplo de entrada e saída.
- Um programa **montador** ou **assembler** faz a tradução da linguagem assembly para a linguagem de máquina (uma espécie de compilador, porém bastante restrito).
- Esse curso lida com os aspectos de software da **programação em assembly** e da **linguagem de máquina**.
  - o hardware só nos interessa na medida em que influencia essa programação.

# Processo de geração de um executável

- Hello world

---

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```

*code/intro/hello.c*

---

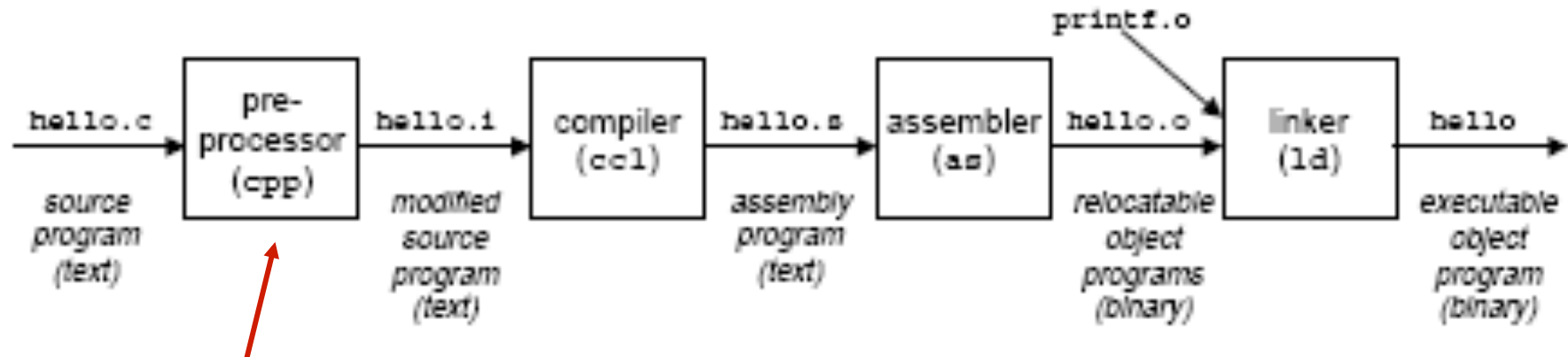
*code/intro/hello.c*

- O programa fonte (arquivo texto) deve ser “traduzido” em uma sequência de instruções de linguagem de máquina, armazenada como um executável (arquivo binário)
- Essa tradução é realizada em 4 fases

# Fase 1: pré-processamento

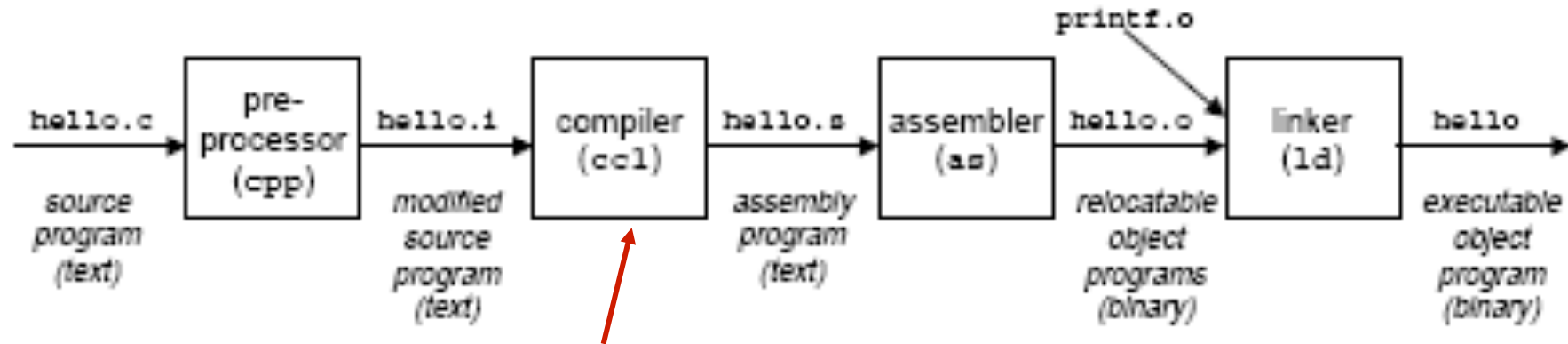
```
unix> gcc -o hello hello.c
```

(usaremos o compilador gcc do Linux no curso)



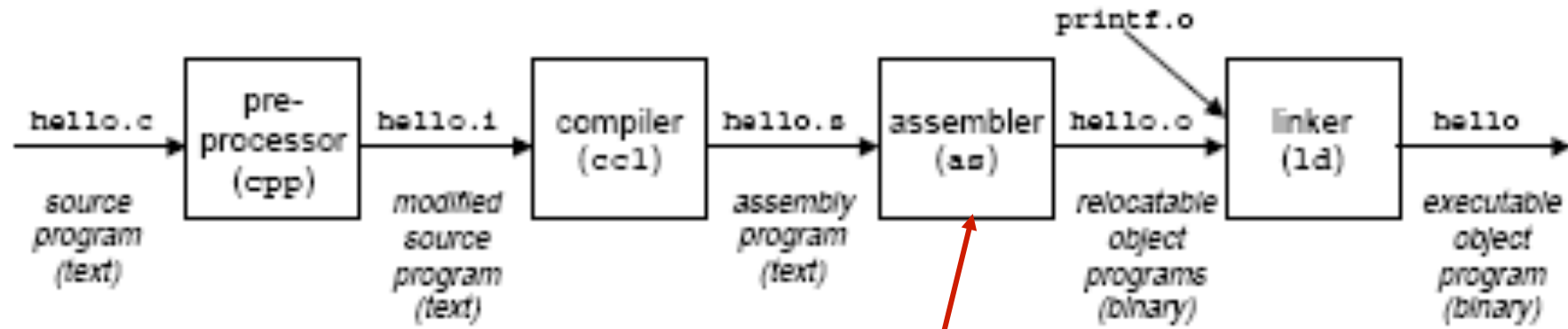
- Modifica o programa em C de acordo com diretivas começadas com #
  - #include <stdio.h> diz ao pré-processador para ler o arquivo stdio.h e inseri-lo no programa fonte (o resultado é um programa expandido em C, normalmente com extensão .i, em Unix)

# Fase 2: compilação



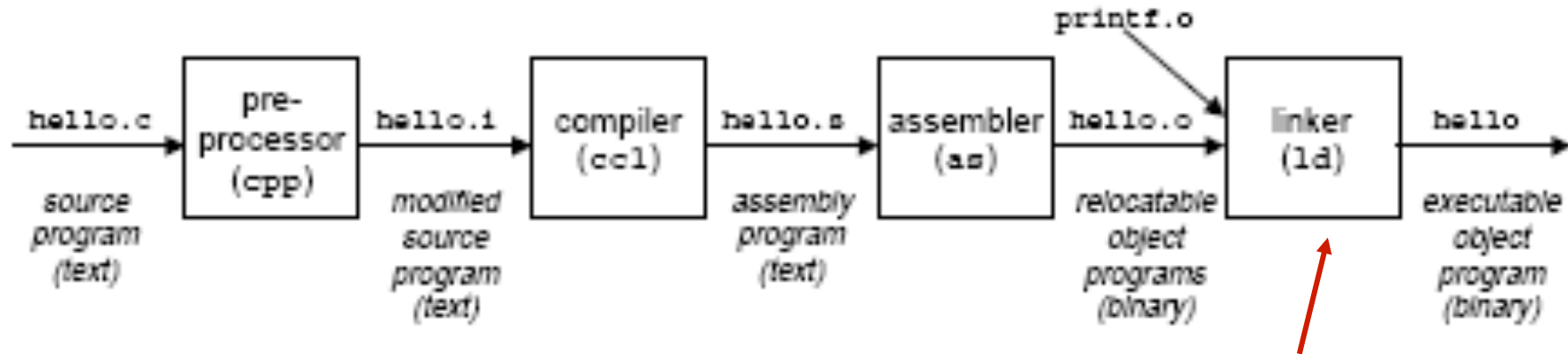
- Compilador traduz o programa .i em programa assembly.
  - é um formato de saída comum para os compiladores de várias linguagens de programação de alto nível

# Fase 3: montagem



- Transforma o programa assembly em instruções em linguagem de máquina (chamado programa objeto)
  - armazenado como um arquivo **binário**, com a extensão `.o`

# Fase 4: ligação

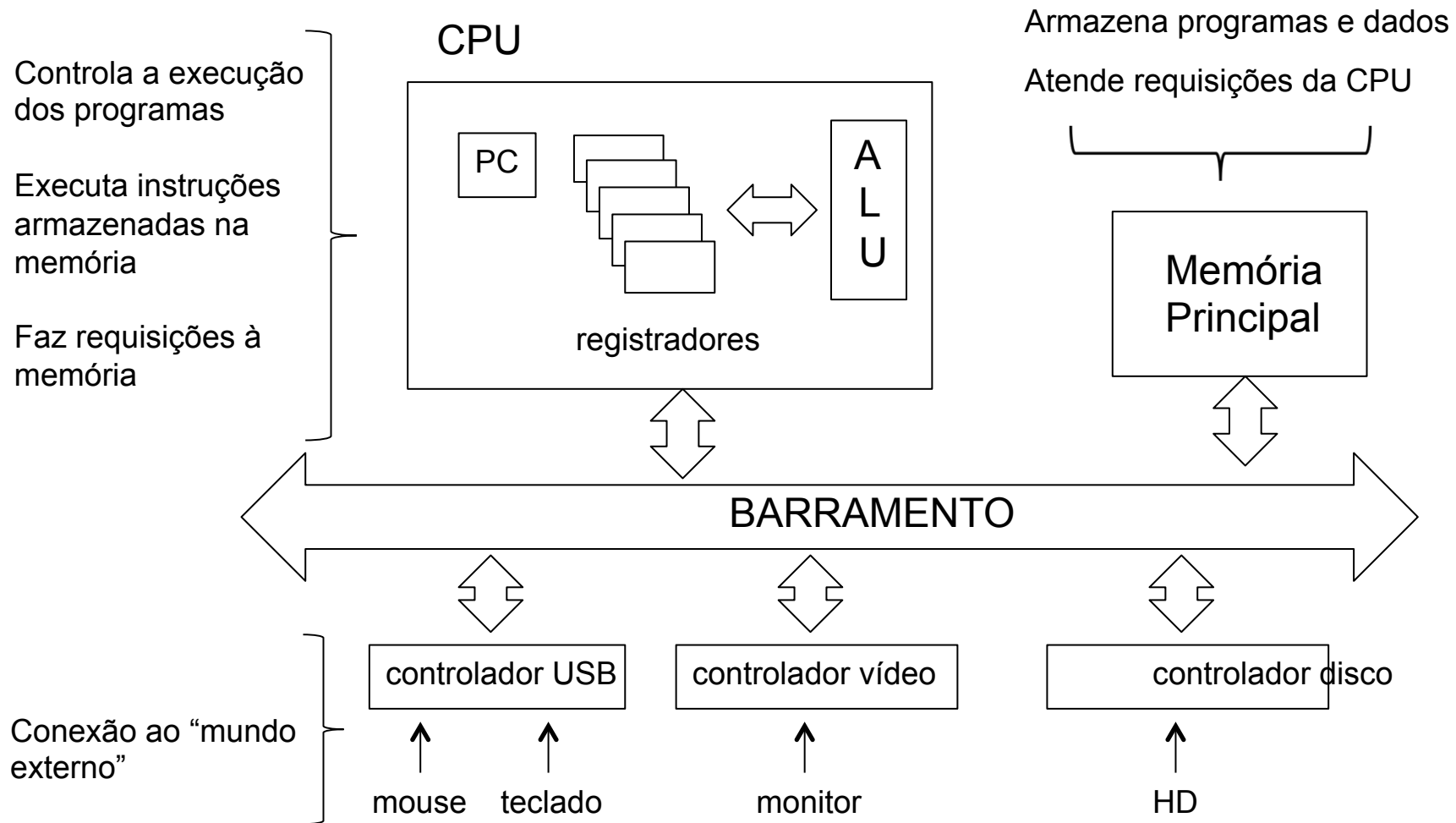


- O ligador (linker) gera o programa executável a partir do programa objeto, gerado pelo assembler.
  - Pode haver funções (ex., printf) não definidas no programa, mas em outro arquivo .o (ou biblioteca) pré-compilado
  - O ligador faz a junção dos programas objeto necessários para gerar o executável.

# Por que entender o processo?

- Saber otimizar uso de recursos/desempenho de programas
- Entender e evitar bugs relativos, por exemplo, às limitações das representações dos tipos e à manipulação da memória
- Entender erros de ligação
- Entender e poder evitar vulnerabilidades de segurança, como em situações de “buffer overflow”

# Arquitetura típica de uma máquina



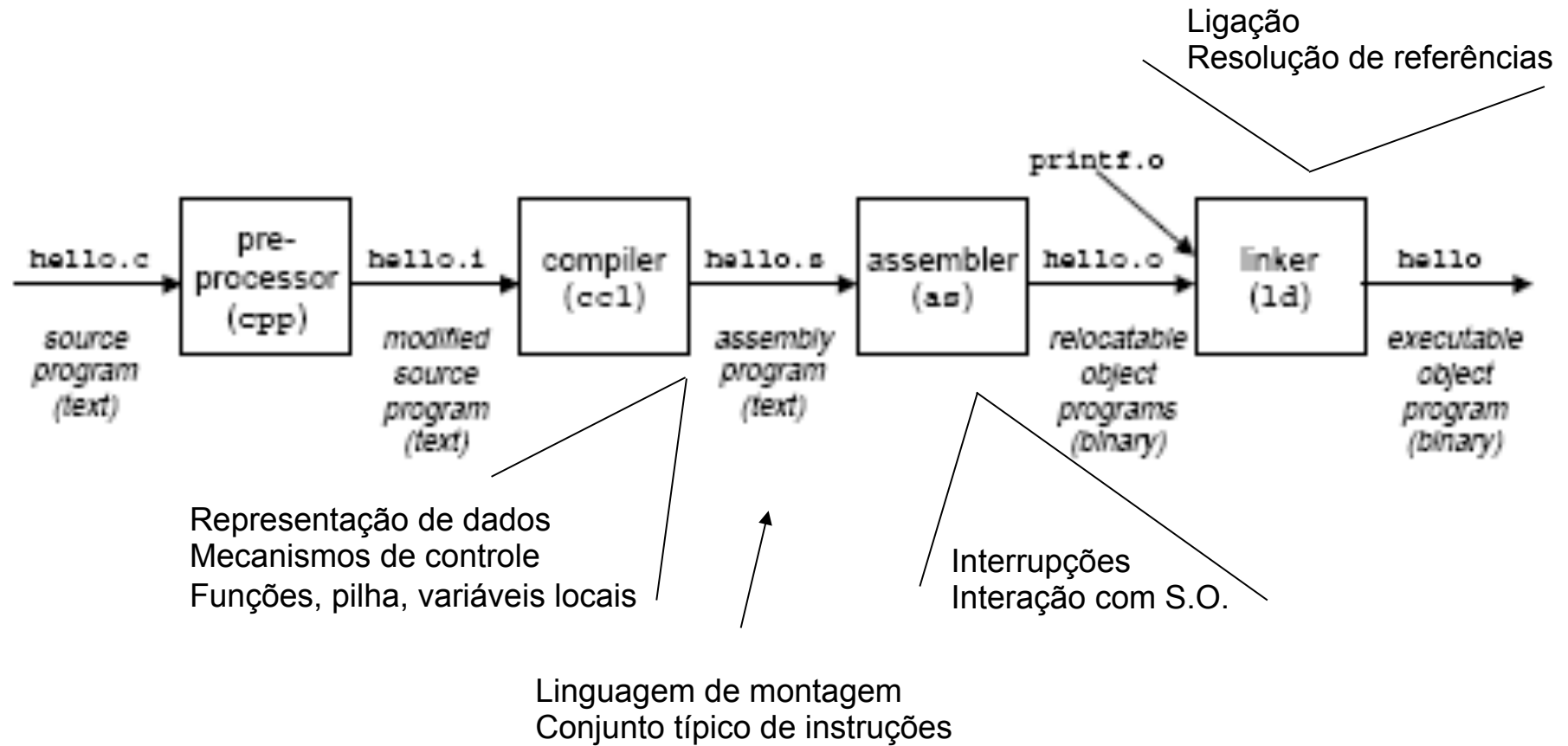
# CPU – Central Processing Unit

- Unidade de Controle
- Unidade Aritmética e Lógica (ALU)
- Conjunto de registradores
  - Funcionam como **uma memória de acesso extremamente rápido** pois estão contidos na própria CPU
    - instruções de *transferência de dados* transferem dados entre memória e registradores
    - operandos de diversas instruções devem estar em registradores
  - Exemplos de registradores
    - PC (program counter): contém o endereço da próxima instrução
    - IR (Instruction register): onde é copiada a instrução a ser executada
    - Registradores de propósito geral

# Memória

- Armazena instruções e dados durante a execução de um programa
- A memória principal pode ser vista como um “array” de **bytes**, cada um com seu endereço (“índice” do array), iniciando com 0
  - Instruções tem um número variável de bytes
  - Dados ocupam um número de bytes que depende do seu tipo
- Os registradores da CPU podem ser usados para armazenar endereços de memória.
  - o tamanho (em bits) dos registradores limita o número de posições de memória endereçáveis.
  - Na IA32, cada registrador tem 32 bits (4 bytes)
    - tamanho máximo da memória endereçável é  $2^{32}$

# Conteúdo do curso



# Programa do curso

- Armazenamento e representação de tipos de dados
- Linguagem de Montagem (IA32)
  - conjunto típico de instruções
- Modelo de execução de linguagem de alto nível (C)
  - implementação de expressões, atribuições, estruturas de controle
- Chamada de procedimentos
  - pilha de execução, registro de ativação, passagem de parâmetros
- Representação de números em ponto flutuante
- Interação com o Sistema Operacional, interrupções
- Compilação e Ligação