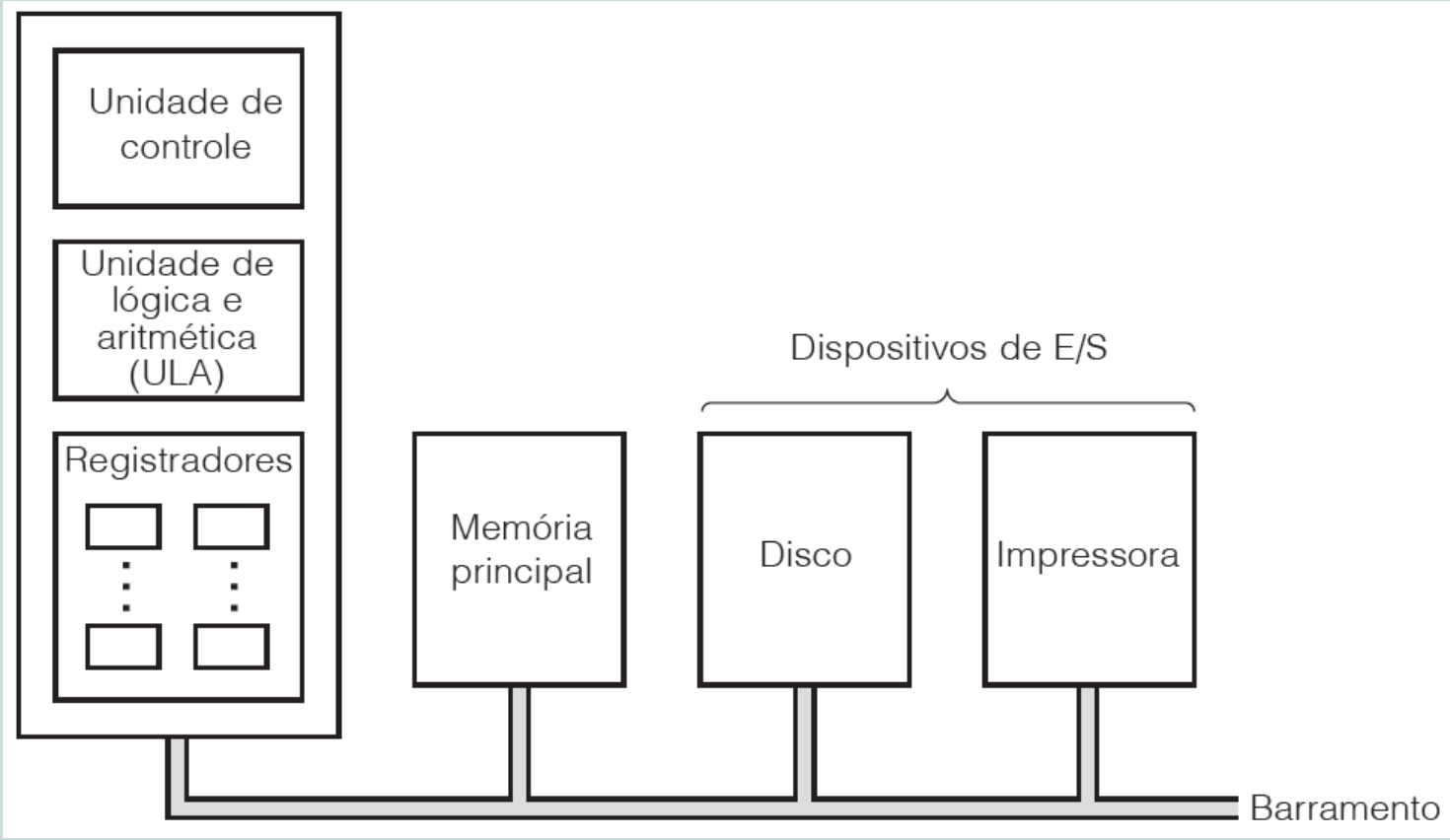
Seção

Processadores

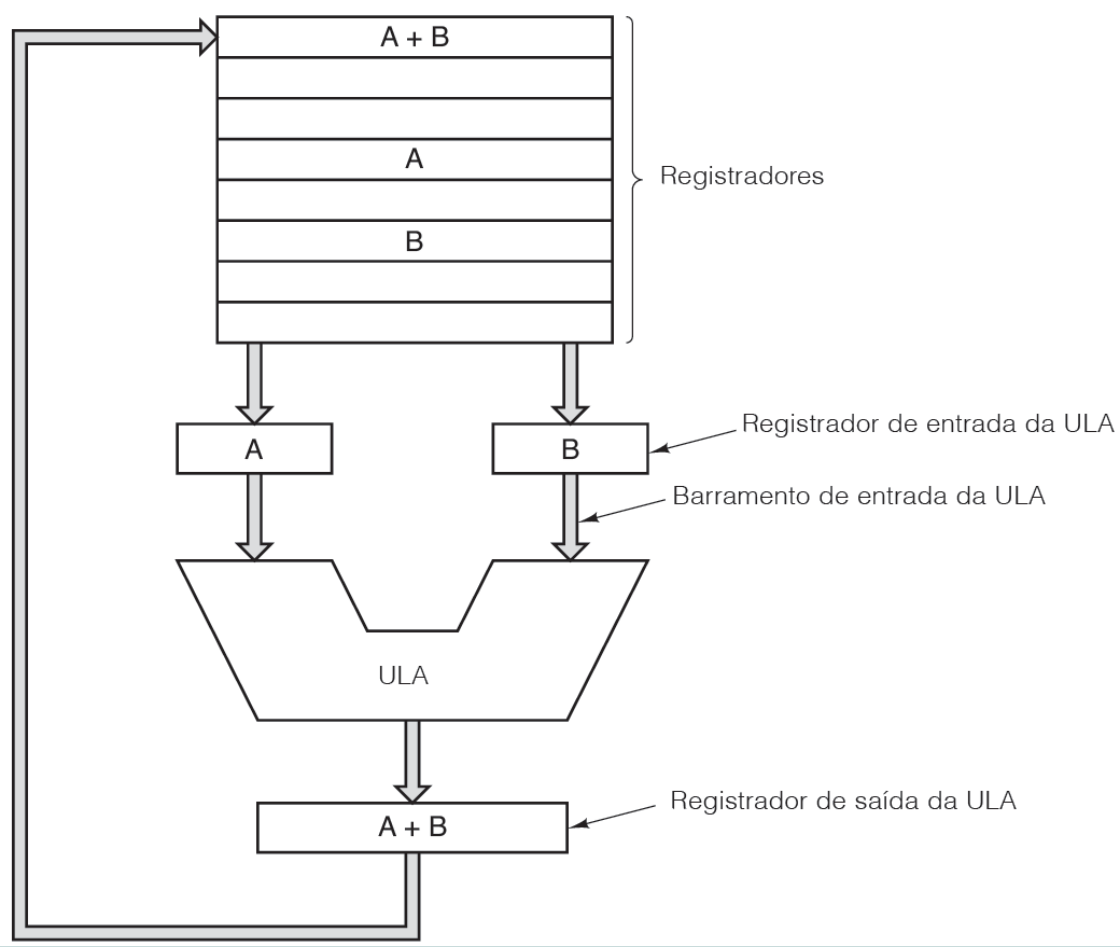
Processadores

- A CPU (Central Processing Unit – unidade central de processamento) é o “cérebro” do computador.



Organização da CPU

- O caminho de dados de uma típica máquina de von Neumann.



Execução de instrução

- A CPU executa cada instrução em uma série de pequenas etapas. Em termos simples, as etapas são as seguintes:

1. Trazer a próxima instrução da memória até o registrador de instrução.

2. Alterar o contador de programa para que aponte para a próxima instrução.

3. Determinar o tipo de instrução trazida.

4. Se a instrução usar uma palavra na memória, determinar onde essa palavra está.

Execução de instrução

5. Trazer a palavra para dentro de um registrador da CPU, se necessário.

6. Executar a instrução.

7. Voltar à etapa 1 para iniciar a execução da instrução seguinte.

- A figura a seguir mostra esse programa informal reescrito como um método Java (isto é, um procedimento) denominado *interpret*.

Execução de instrução

Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO

```
public class Interp {

    static int PC;                                  // contador de programa contém endereço da próxima instr
    static int AC;                                  // o acumulador, um registrador para efetuar aritmética
    static int instr;                               // um registrador para conter a instrução corrente
    static int instr_type;                          // o tipo da instrução (opcode)
    static int data_loc;                            // o endereço dos dados, ou –1 se nenhum
    static int data;                                // mantém o operando corrente
    static boolean run_bit = true;                  // um bit que pode ser desligado para parar a máquina

    public static void interpret(int memory[ ], int starting_ address) {
        // Esse procedimento interpreta programas para uma máquina simples com instruções que têm
        // um operando na memória. A máquina tem um registrador AC (acumulador), usado para
        // aritmética. A instrução ADD soma um inteiro na memória do AC, por exemplo.
        // O interpretador continua funcionando até o bit de funcionamento ser desligado pela instrução HALT.
        // O estado de um processo que roda nessa máquina consiste em memória, o
        // contador de programa, bit de funcionamento e AC. Os parâmetros de entrada consistem
        / na imagem da memória e no endereço inicial.

    PC = starting_address;
    while (run_bit) {
        instr = memory[PC];                         // busca a próxima instrução e armazena em instr
        PC = PC + 1;                                // incrementa contador de programa
        instr_type = get_instr_type(instr);         // determina tipo da instrução
        data_loc = find_data(instr, instr_type);    // localiza dados (–1 se nenhum)
        if (data_loc >= 0)                          // se data_loc é –1, não há nenhum operando
           data = memory[data_loc];                 // busca os dados
        execute(instr_type, data);                  // executa instrução
        }
    }
    private static int get_instr_type(int addr) { ... }
    private static int find_data(int instr, int type) { ... }
    private static void execute(int type, int data) { ... }
}
```

Execução de instrução

- Computadores simples com instruções interpretadas tinham benefícios, entre os quais os mais importantes eram:

1. A capacidade de corrigir em campo instruções executadas incorretamente ou até compensar deficiências de projeto no hardware básico.

2. A oportunidade de acrescentar novas instruções a um custo mínimo, mesmo após a entrega da máquina.

3. Projeto estruturado que permitia desenvolvimento, teste e documentação eficientes de instruções complexas.

RISC *versus* CISC

- Em 1980, um grupo em Berkeley, liderado por David Patterson e Carlo Séquin, começou a projetar chips para CPUs VLSI que não usavam interpretação.
- Eles cunharam o termo RISC para esse conceito e deram ao seu chip de CPU o nome RISC I CPU, seguido logo depois pelo RISC II.
- A característica que chamou a atenção de todos era o número relativamente pequeno de instruções disponíveis, em geral cerca de 50.
- Número muito menor do que os *mainframes* da IBM.

RISC *versus* CISC

- Mesmo que uma máquina RISC precisasse de quatro ou cinco instruções para fazer o que uma CISC fazia com uma só, se as instruções RISC fossem dez vezes mais rápidas, o RISC vencia.
- A Intel conseguiu empregar as mesmas ideias mesmo em uma arquitetura CISC.
- Mesmo que essa abordagem híbrida não seja tão rápida quanto um projeto RISC puro, ela resulta em desempenho global competitivo e ainda permite que softwares antigos sejam executados sem modificação.

Princípios de projeto para computadores modernos

- Há um conjunto de princípios de projeto, às vezes denominados princípios de projeto RISC, que os arquitetos de CPUs de uso geral se esforçam por seguir:

  - Todas as instruções são executadas diretamente por hardware.
  - É preciso maximizar a taxa de execução das instruções.
  - Instruções devem ser fáceis de decodificar.
  - Somente LOAD e STORE devem referenciar a memória.
  - É preciso providenciar muitos registradores.

Paralelismo no nível de instrução

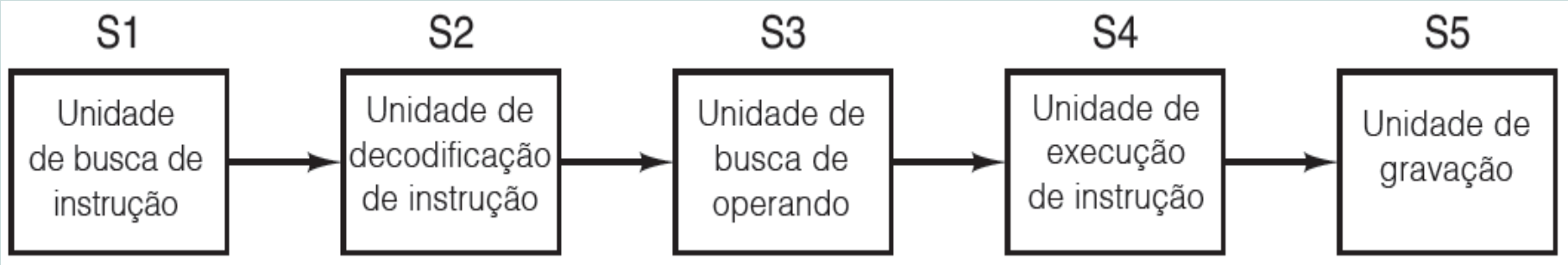
Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO

- O paralelismo tem duas formas gerais:
- **No nível de instrução**
  - O paralelismo é explorado dentro de instruções individuais para obter da máquina mais instruções por segundo.
- **No nível de processador**
  - Várias CPUs trabalham juntas no mesmo problema. Cada abordagem tem seus próprios méritos.

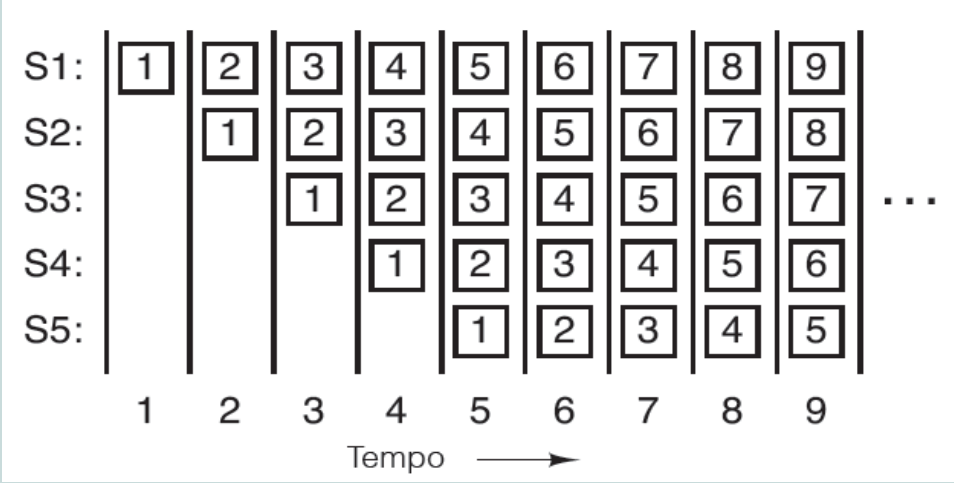
Pipelining (paralelismo)

Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO

- *Pipeline* de cinco estágios.



- Estado de cada estágio como uma função do tempo.



Pipelining (paralelismo)

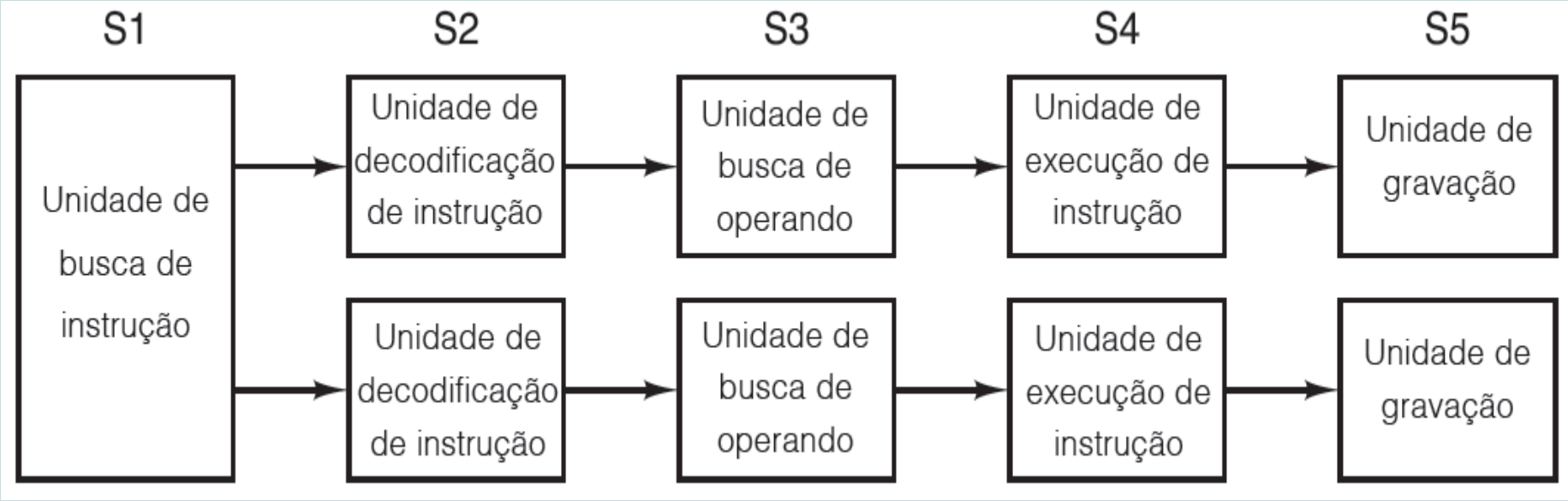
- O *pipelining* permite um compromisso entre **latência** (o tempo que demora para executar uma instrução) e **largura de banda de processador** (quantos MIPS a CPU tem).

- Com um tempo de ciclo de T ns e n estágios no *pipeline*, a latência é nT ns porque cada instrução passa por n estágios, cada um dos quais demora T ns.

- Visto que uma instrução é concluída a cada ciclo de *clock* e que há $10^9/T$ ciclos de *clock* por segundo, o número de instruções executadas por segundo é $10^9/T$.

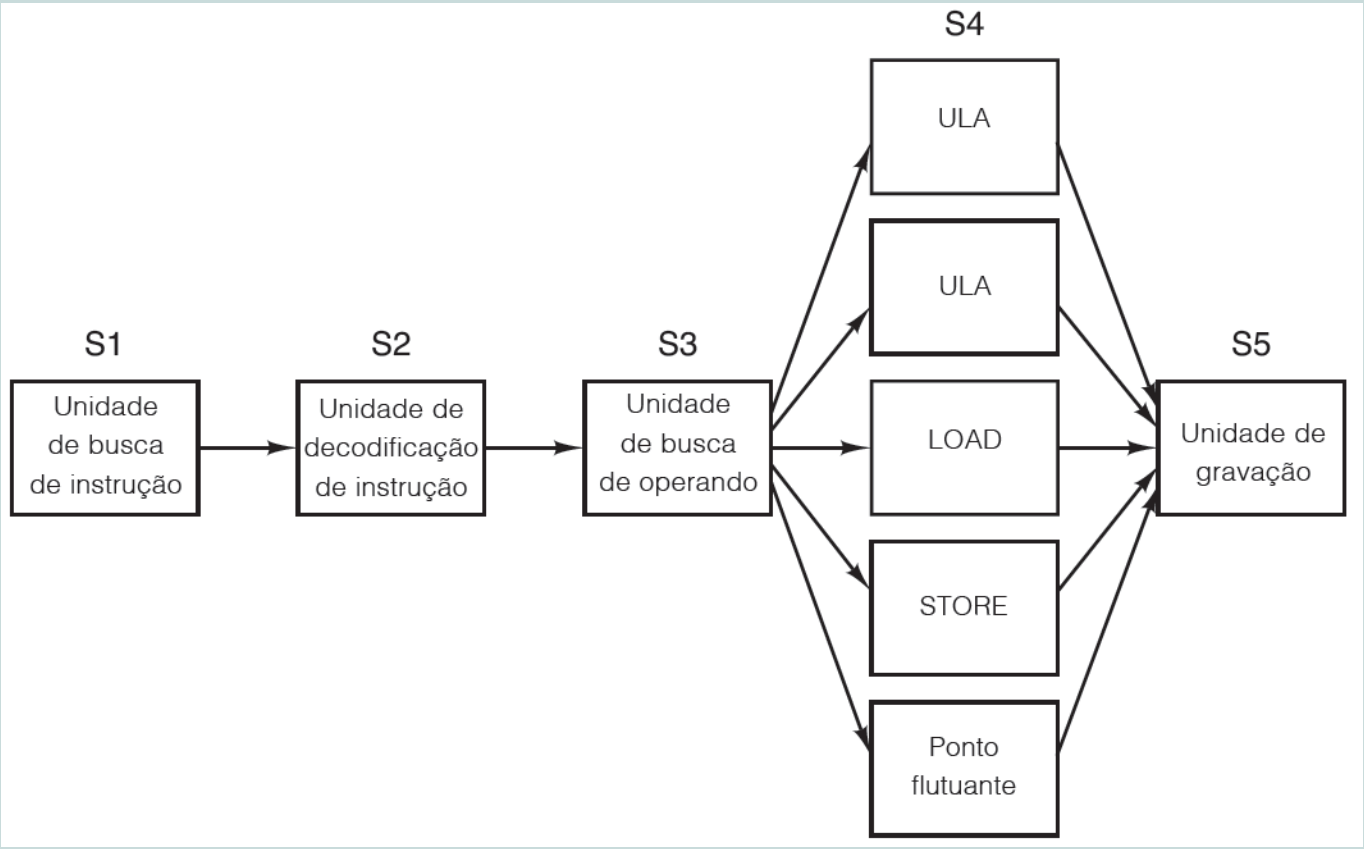
Pipelining (paralelismo)

- *Pipelines* duplos de cinco estágios com uma unidade de busca de instrução em comum.



Pipelining (paralelismo)

- A ideia básica é ter apenas um único *pipeline*, mas lhe dar várias unidades funcionais:

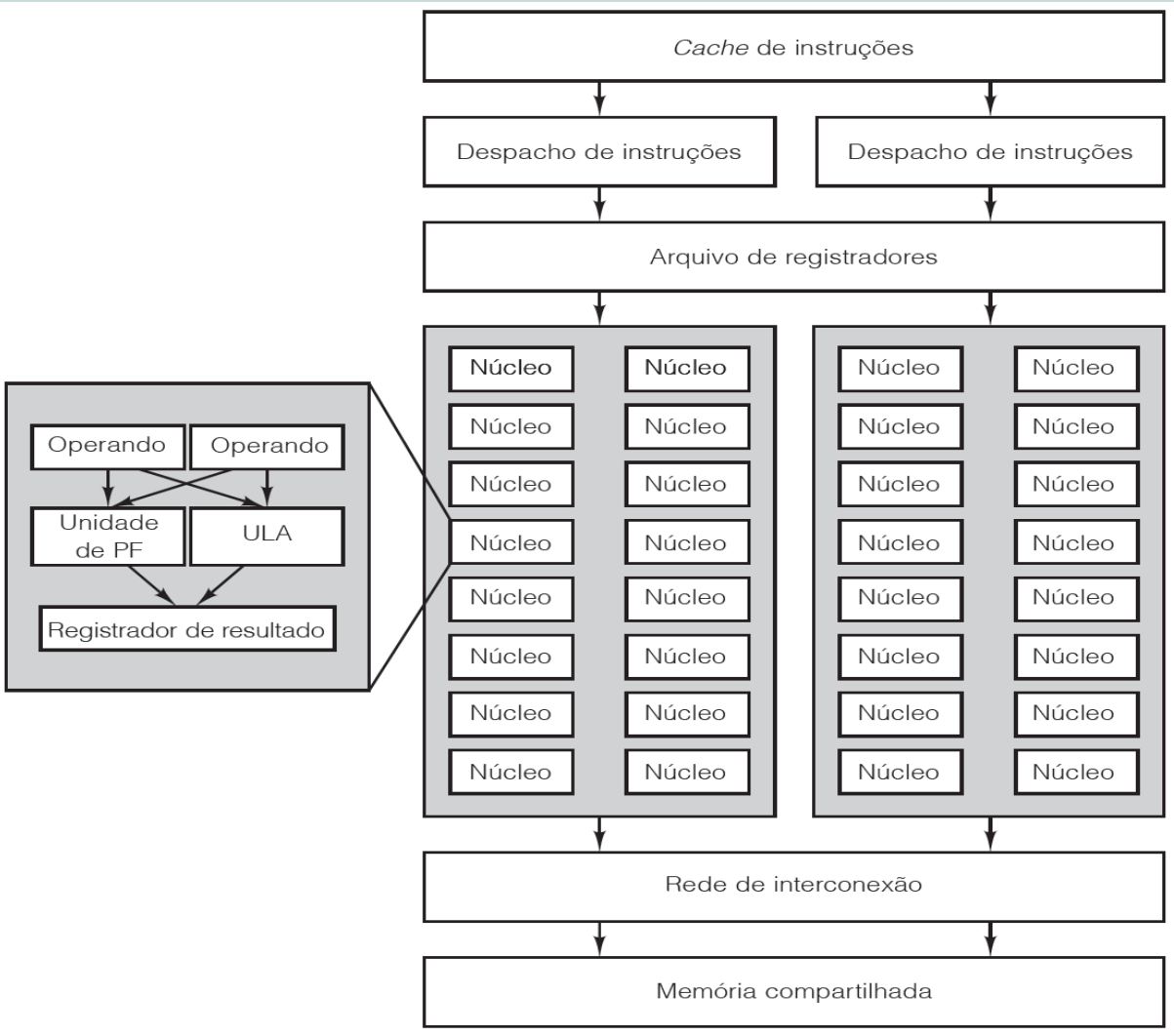


Paralelismo no nível do processador

- Um **processador SIMD** consiste em um grande número de processadores idênticos que efetuam a mesma sequência de instruções sobre diferentes conjuntos de dados.
- As modernas unidades de processamento de gráficos (GPUs) contam bastante com o processamento SIMD para fornecer poder computacional maciço com poucos transistores.
- A figura a seguir mostra o processador SIMD no núcleo da GPU Fermi da Nvidia.

Paralelismo no nível do processador

Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO



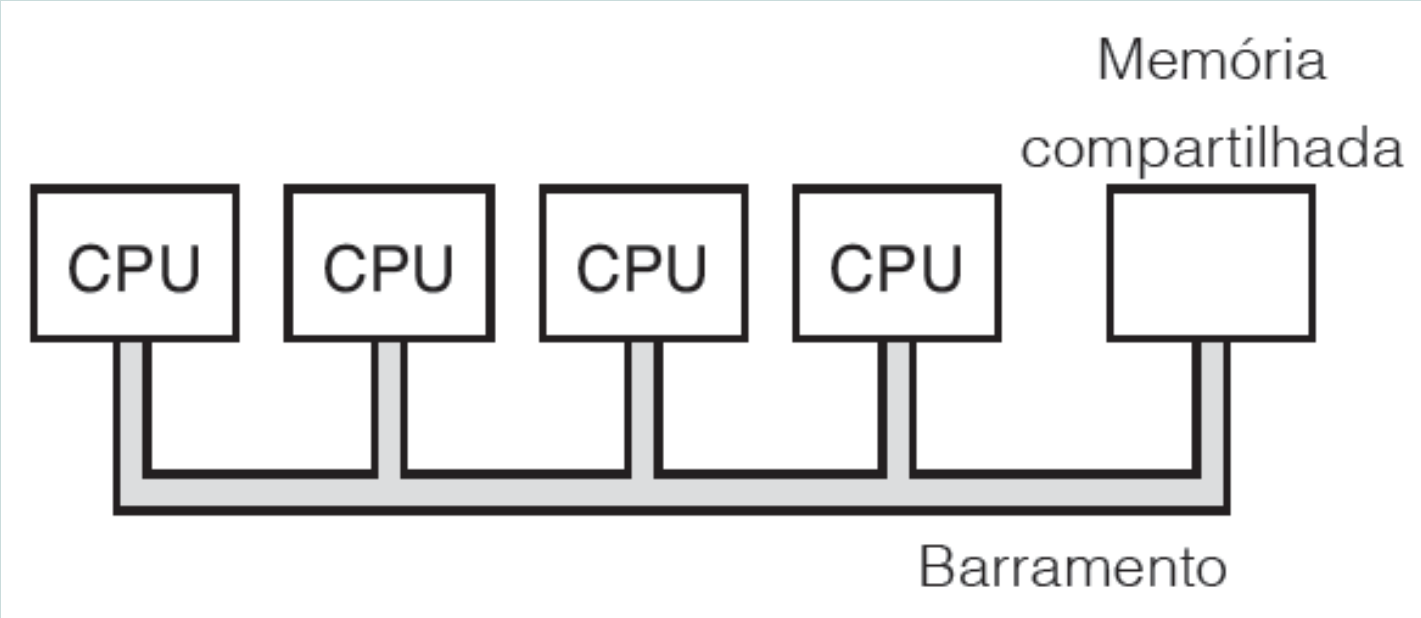
Paralelismo no nível do processador

Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO

- Para um programador, um **processador vetorial** se parece muito com um processador SIMD.
- Ele é muito eficiente para executar uma sequência de operações em pares de elementos de dados.
- Porém, todas as operações de adição são efetuadas em uma única unidade funcional, de alto grau de paralelismo.
- Nosso primeiro sistema paralelo com CPUs totalmente desenvolvidas é o **multiprocessador**, um sistema com mais de uma CPU que compartilha uma memória em comum.

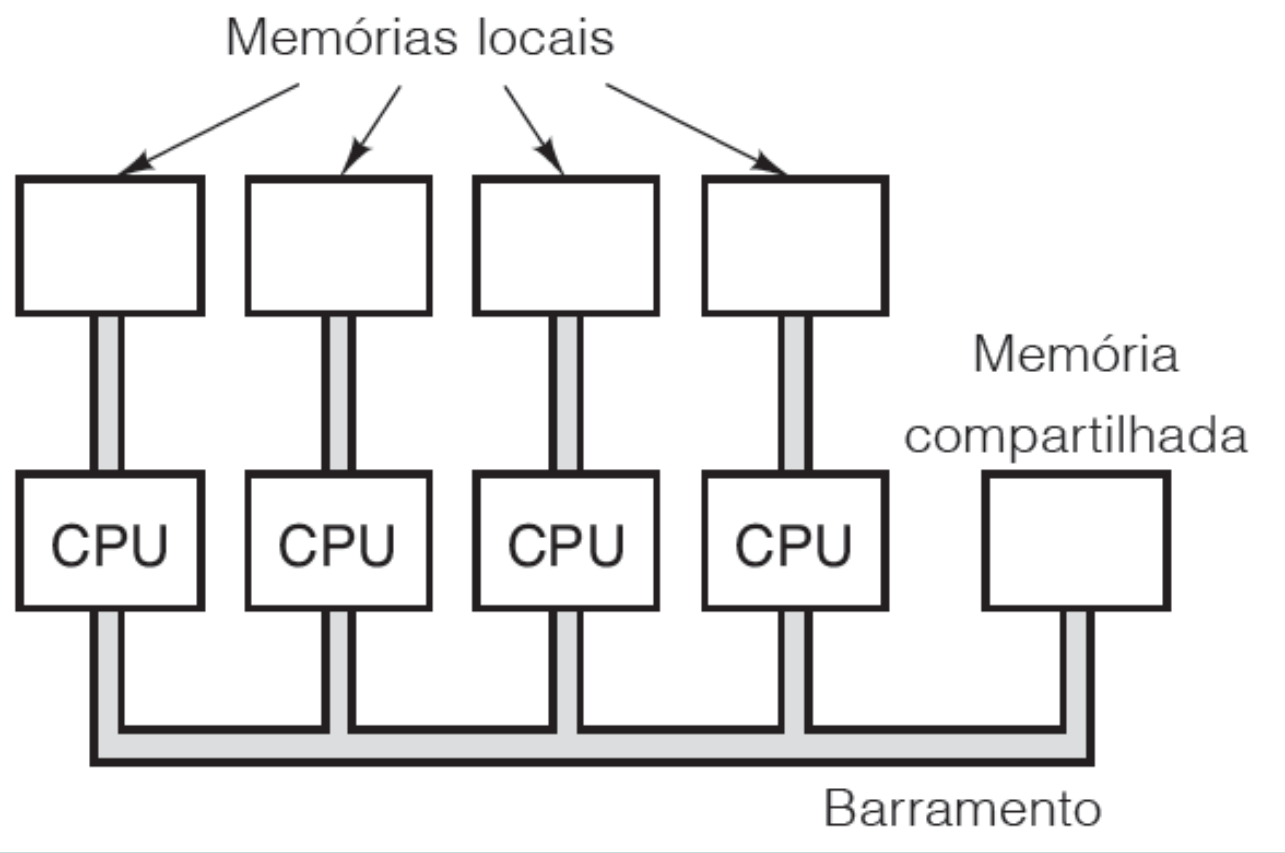
Paralelismo no nível do processador

- Multiprocessador de barramento único.



Paralelismo no nível do processador

- Multicomputador com memórias locais.



Paralelismo no nível do processador

Tanenbaum • Austin
Organização estruturada de computadores
6ª EDIÇÃO

- Costuma-se dizer que as CPUs de um **multicomputador** são fracamente acopladas, para contrastá-las com as CPUs fortemente acopladas de um multiprocessador.
- As CPUs de um multicomputador se comunicam enviando mensagens umas às outras, mais ou menos como enviar e-mails, porém, com muito mais rapidez.
- Multiprocessadores são mais fáceis de programar.
- Multicomputadores são mais fáceis de construir.