Arquitetura e Organização de computadores

ENGENHARIA DA COMPUTAÇÃO – UFC/SOBRAL

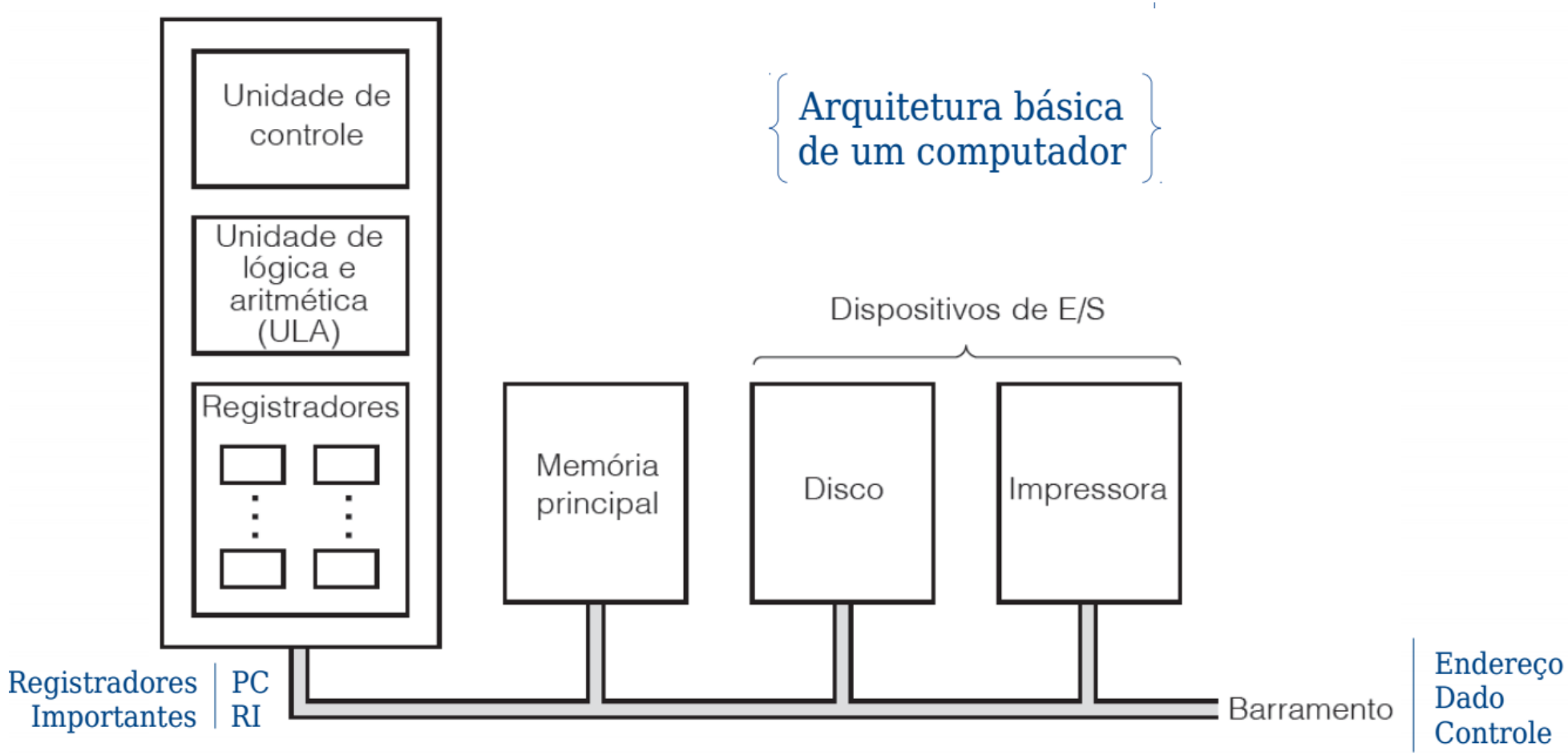
Prof. Danilo Alves

danilo.alves@alu.ufc.br

Organizações de sistemas de computadores

- Processadores
- Memória primária
- Memória secundária
- Entrada e saída

ARQUITETURA DE VON NEUMANN

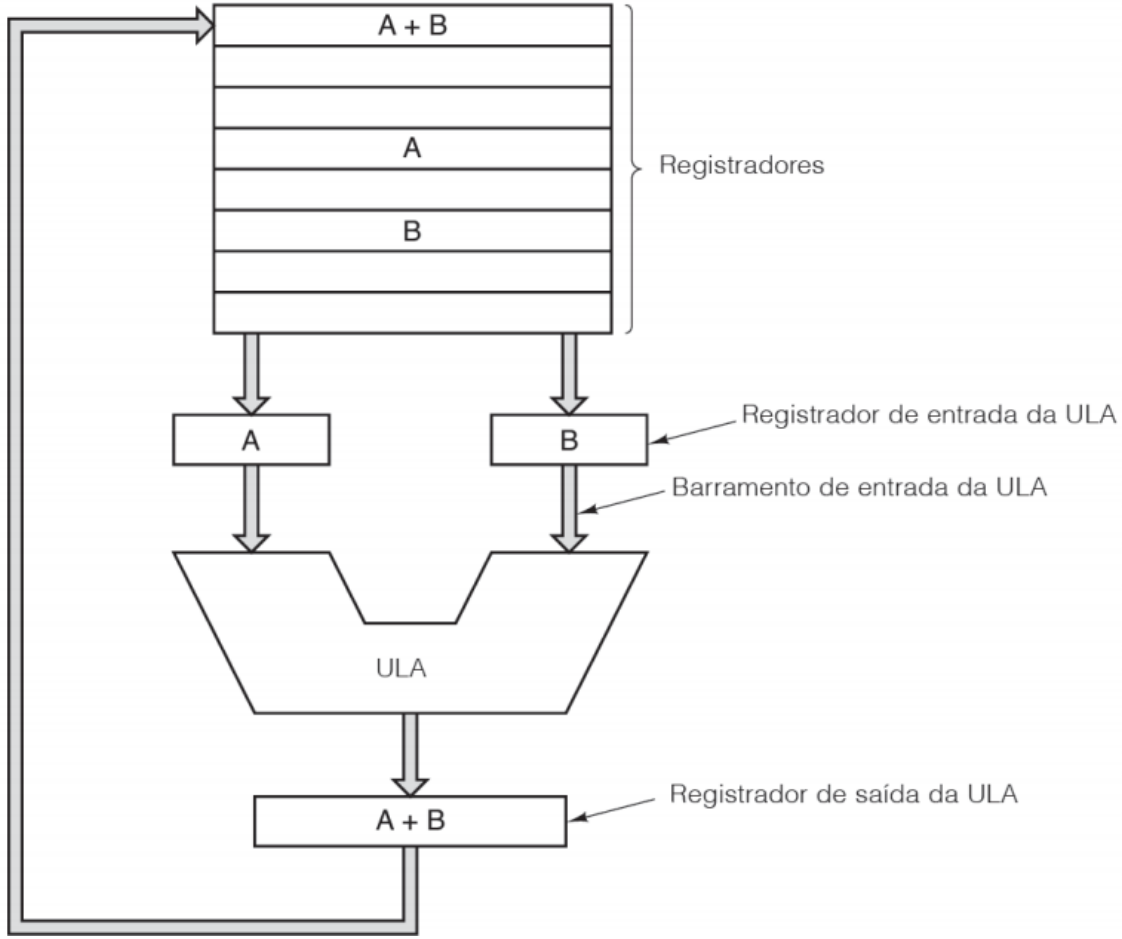


ARQUITETURA DE VON NEUMANN

- ULA: Um circuito digital que realiza operações lógicas e aritméticas. A ULA é uma peça fundamental da unidade central de processamento (CPU)
- Banco de registradores: é uma pequena área de armazenamento para os dados que o processador está usando, são acessados mais rapidamente do que os dados armazenados no sistema de memória, geralmente suportam vários acessos simultâneos
- Lógica de controle: Responsável pelo controle do resto do processador, determinando quando as instruções podem ser executadas e quais operações são necessárias para executar cada instrução
- Contador de Programa (PC)
- Registrador de Instrução (IR)

Organização da CPU

- Caminho de dados
- Tipos de instruções:
  - Registrador-memória
  - Registrador-registrador



EXECUÇÃO DE INSTRUÇÕES

- 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
  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

- Ciclo **buscar-decodificar-executar**

EX

```
public class Interp {

    static int PC;                          // contador de programa contém endereço da próxima ir
    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) { ... }
}
```

EQUIVALÊNCIA NO PROJETO DO COMPUTADOR

- A equivalência entre processadores de hardware e interpretadores é de grande importância para a organização de computadores
- Após especificar uma linguagem de máquina L para o computador, os projetistas devem decidir se irão desenvolver um processador de hardware ou um interpretador
  - Se a equipe decide por utilizar um interpretador, ela deve providenciar a máquina que irá executar o interpretador
  - É possível ainda uma construção híbrida
- Quando usar um interpretador?
  - Quando o conjunto de instruções que a linguagem possui é muito complexa e com muitas opções, o que significa alto custo para o desenvolvimento do hardware
  - Um interpretador subdivide as instruções da máquina em diversas etapas
- Consequência: a máquina na qual o interpretador roda deve ser muito mais simples e menos cara do que seria um processador de hardware

EQUIVALÊNCIA NO PROJETO DO COMPUTADOR

- Máquinas interpretadas custariam menos para serem produzidas, pois o custo de desenvolvimento de hardware, é maior do que de desenvolvimento de software
- No entanto, máquinas que executam as instruções diretamente são mais rápidas
- Essa equivalência permitiu criar computadores de diferentes custos e velocidade que fazem basicamente as mesmas coisas
  - Implementação em hardware era usada em modelos mais caros (mais rápidos), enquanto que a interpretação era usada em modelos mais baratos (mais lentos)

EXECUÇÃO DE INSTRUÇÕES

- 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

- Tais benefícios permitiu a criação de computadores mais baratos e com conjuntos de instruções cada vez mais complexos

  – O VAX da *Digital Equipment Corporation* (DEC) tinha várias centenas de instruções e mais de 200 modos diferentes de especificar os operandos a serem usados em cada instrução

  – Entretanto, a falta de desempenho foi fatal para o VAX e para a DEC

LINHAS DE ARQUITETURA

- Arquitetura **CISC** – *Complex Instruction Set Computer*
  - – Arquitetura cujo **conjunto de instruções era bastante complexo**
  - – Capaz de executar centenas de instruções complexas diferentes sendo extremamente versátil
  - – Exemplos de processadores CISC são os 386 e os 486 da Intel

- Arquitetura **RISC** – *Reduced Instruction Set Computer*
  - – Arquitetura cujo conjunto de instruções era simplificado (reduzido)
  - – Favorece um conjunto **simples e pequeno** de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas
  - – Exemplos de processadores RISC são os Alpha, SPARC, MIPS, e PowerPC

- Computadores atuais utilizam uma implementação híbrida, misturando as duas linhas

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

- Eliminar um nível de interpretação dá alta velocidade para a maioria das instruções
- Em computadores CISC, instruções complexas podem ser subdivididas em partes separadas que então podem ser executadas como uma sequência de microinstruções

  – Torna a máquina mais lenta, mas pode ser aceitável para instruções que ocorrem com menos frequência

PROJETO PARA COMPUTADORES MODERNOS

– É preciso maximizar a taxa de execução das instruções – MIPS

– Instruções devem ser fáceis de decodificar

- Instruções regulares, de comprimento fixo, pequeno número de campos

– Somente LOAD e STORE devem referenciar a memória

– É preciso providenciar muitos registradores

PARALELISMO NO NÍVEL DE INSTRUÇÃO

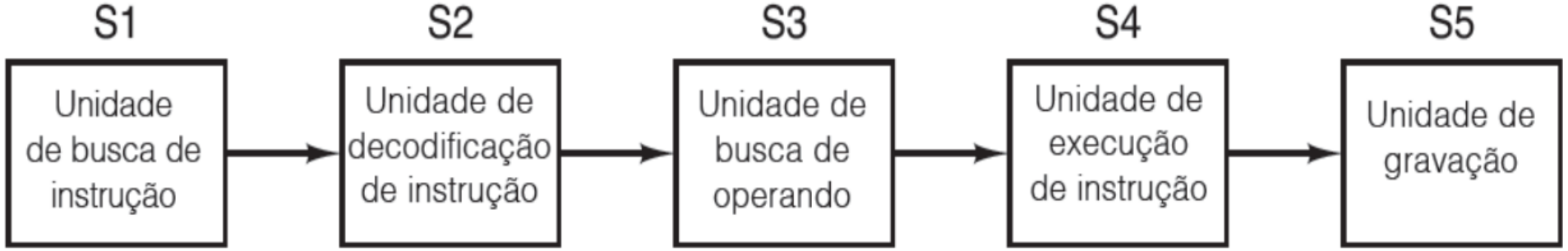
- Duas formas gerais:
  - **No nível de instrução** – O paralelismo é explorado dentro de instruções individuais para obter da máquina mais instruções por segundo
  - **No nível de processador** – Várias CPUs trabalham juntas no mesmo problema
- Cada abordagem tem seus próprios méritos

FÁBRICA DE BOLOS

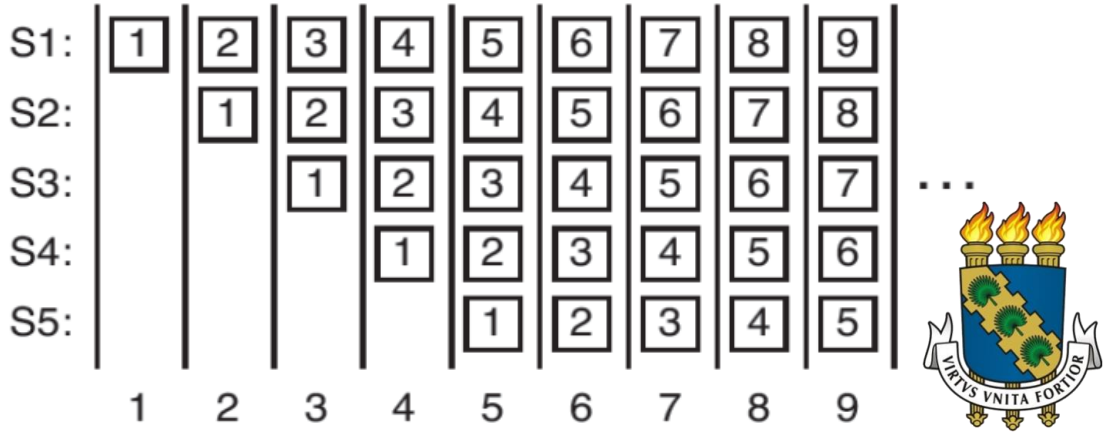
- 1. Aquisição da matéria-prima
- 2. Processo de produção do bolo
- 3. Processo de embalagem para expedição

PIPELINING (PARALELISMO, CANALIZAÇÃO)

- Pipeline de cinco estágios



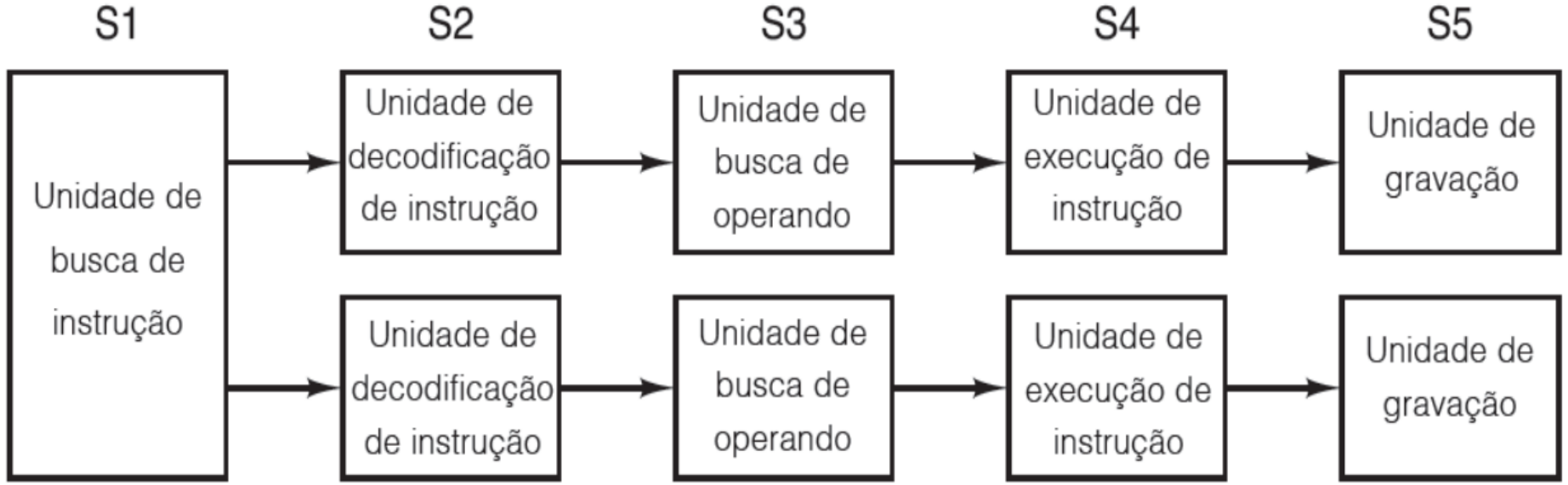
- Estado de cada estágio como uma função do tempo



- Latência
  - o tempo que demora para executar uma instrução

PIPELINING (PARALELISMO, CANALIZAÇÃO)

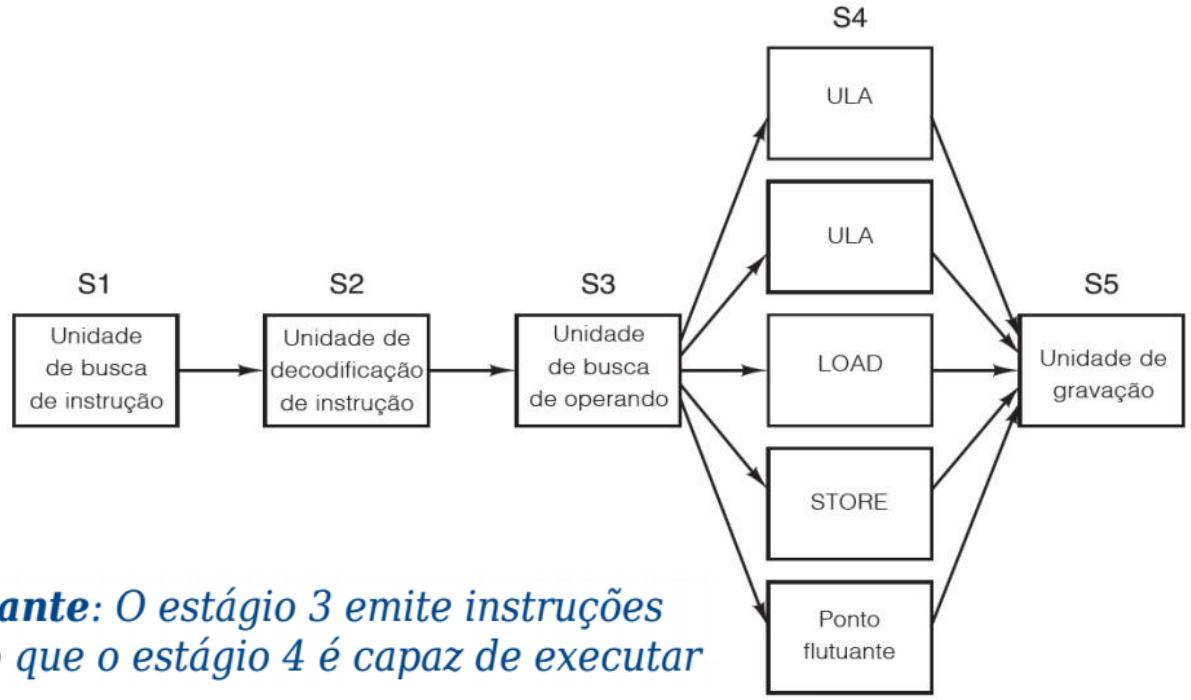
- Pipelines duplos de cinco estágios com uma unidade de busca de instrução em comum



- O compilador deve garantir que não haja dependência entre os pipelines ou conflitos devem ser detectados por hardware extra

PIPELINING (PARALELISMO, CANALIZAÇÃO)

- Duplicar o número de pipelines exigia aumentar a complexidade do hardware
- A ideia básica é ter apenas um único pipeline, mas lhe dar várias unidades funcionais:
- Superescalar



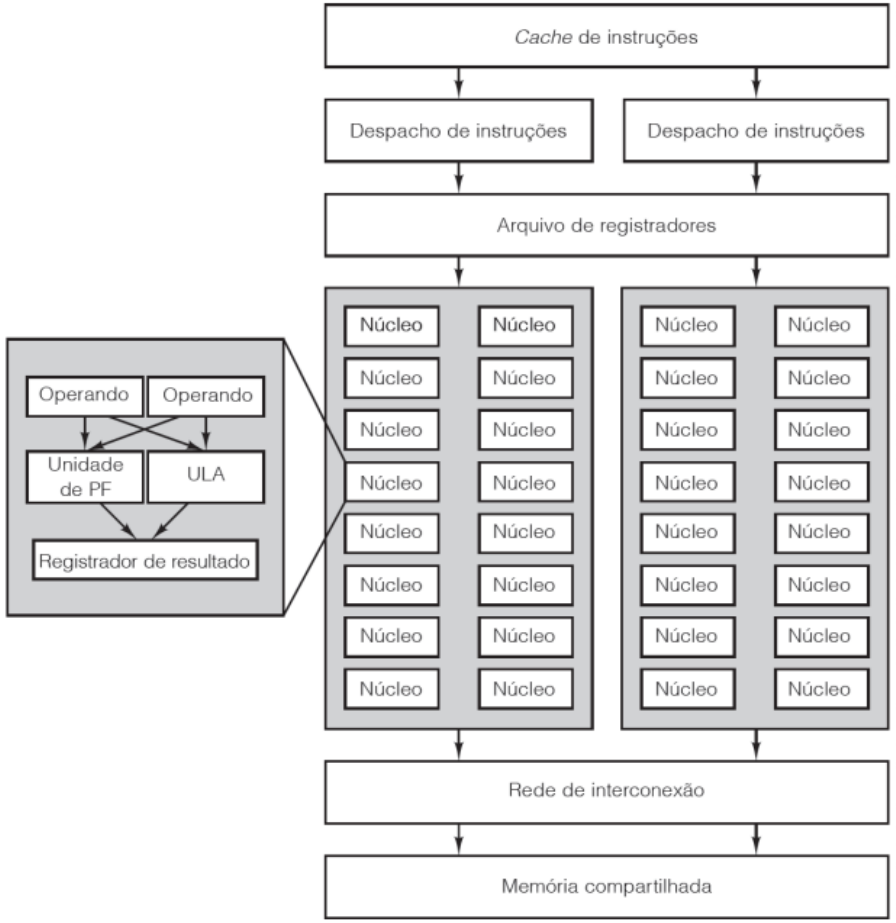
***Observação importante**: O estágio 3 emite instruções muito mais rápido do que o estágio 4 é capaz de executar*

PARALELISMO NO NÍVEL DO PROCESSADOR

- Um processador SIMD (Single Instruction-stream Multiple data-stream) consiste em um grande número de processadores idênticos que efetuam a mesma sequência de instruções sobre diferentes conjuntos de dados
  - Requer regularidade entre as instruções, como operações em matrizes
  - Ideal para processamento gráfico

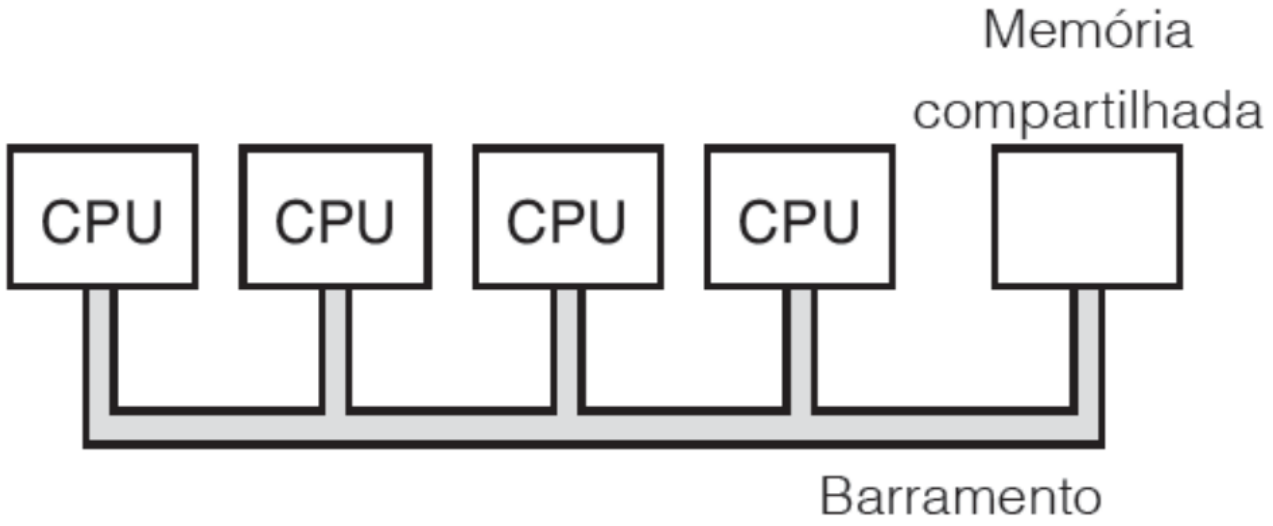
- As modernas unidades de processamento de gráficos (GPUs) contam bastante com o processamento SIMD para fornecer poder computacional maciço com poucos transistores
  - Uma menor quantidade de transistores proporciona menos geração de calor

PARALELISMO NO NÍVEL DO PROCESSADOR



PARALELISMO NO NÍVEL DO PROCESSADOR

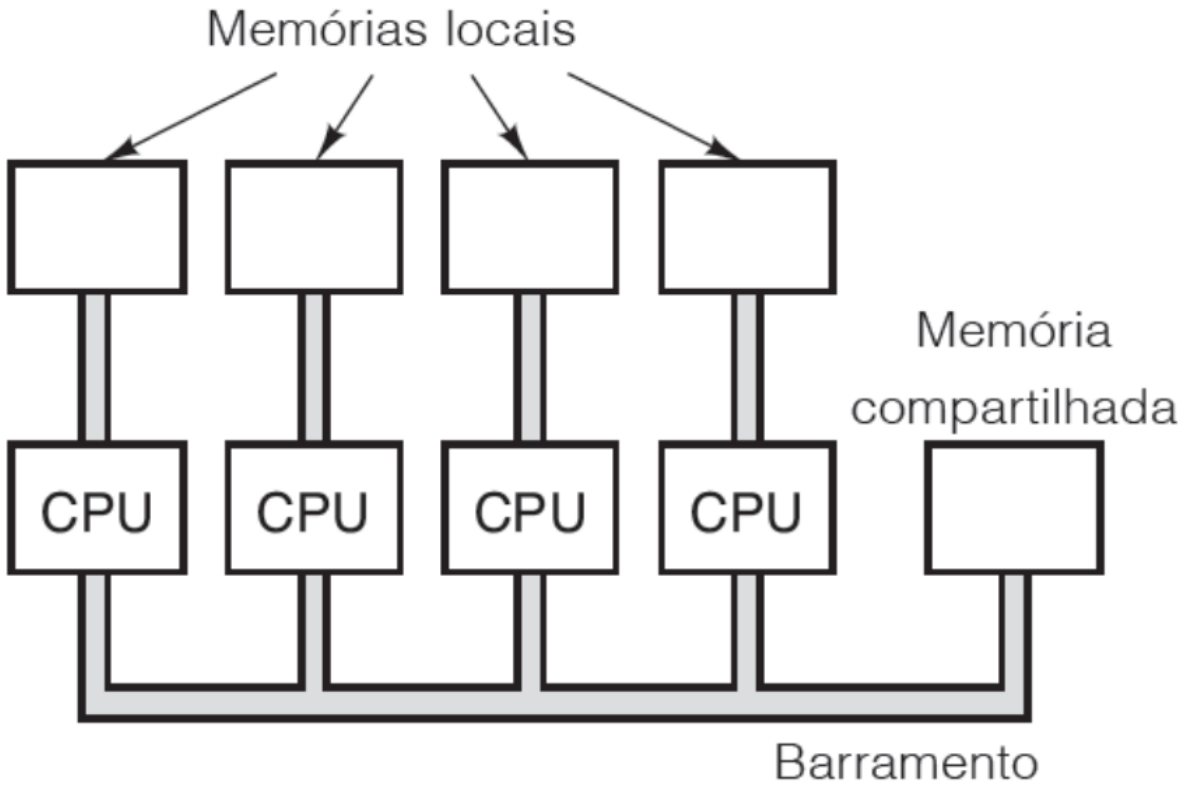
- Multiprocessador de barramento único



- Cada CPU é independente – cada uma possui sua própria unidade de controle, enquanto que processadores SIMD possuem uma única UC compartilhada entre os vários processadores

PARALELISMO NO NÍVEL DO PROCESSADOR

- Multicomputador com memórias locais



PARALELISMO NO NÍVEL DO PROCESSADOR

- Multicomputador
  - Dificuldade está em conectar todos os processadores à memória
  - Grande número de computadores conectados com sua própria memória
  - Enviam mensagem entre eles

PARALELISMO NO NÍVEL DO PROCESSADOR

- Costuma-se dizer que as CPUs de um multicomputador são fracamente acopladas, em contraste com as CPUs fortemente acopladas de um multiprocessador

- As CPUs de um multicomputador se comunicam enviando mensagens umas às outras
- Multiprocessadores são mais fáceis de programar
- Multicomputadores são mais fáceis de construir