Sistemas Operacionais

Simulador da Arquitetura de Von Neumann e Pipeline MIPS

Professor: Prof. Michel Pires da Silva

1. Introdução

A arquitetura de Von Neumann, proposta por John von Neumann na década de 1940, é a base dos computadores modernos. Esta arquitetura caracteriza-se pelo uso de uma única memória compartilhada para armazenar tanto dados quanto instruções, o que leva ao fenômeno conhecido como o *bottleneck* de Von Neumann, onde a CPU fica limitada pela velocidade de transferência entre a memória e o processador.

Para mitigar esse problema e otimizar o desempenho dos sistemas computacionais, é comum utilizar uma hierarquia de memórias. Nesse contexto, a memória *cache* — uma memória de alta velocidade, porém com capacidade limitada — atua como intermediário entre a CPU e a memória principal, armazenando temporariamente dados frequentemente acessados para reduzir o tempo de acesso.

O presente trabalho tem como objetivo a construção de um simulador básico, utilizando a linguagem C/C++, que implementa os principais componentes da arquitetura de Von Neumann. Esses componentes incluem a Unidade Central de Processamento (CPU), a memória principal, a cache, a memória secundária e os periféricos. Além disso, o simulador também irá incorporar a execução de instruções modelada com base no *pipeline* MIPS (*Microprocessor without Interlocked Pipeline Stages*), que permite a execução de múltiplas instruções de forma paralela, em diferentes ciclos de clock. Nesse caso, considera-se um ciclo de clock a movimentação de uma instrução de um estágio para outro do referido pipeline.

Por meio deste simulador, será possível observar o impacto da hierarquia de memórias, em especial o uso da cache, no desempenho geral do sistema, além de compreender como as instruções são processadas em uma arquitetura de *pipeline*. Ainda sob esse conceito, será possível trabalhar conceitos como escalonamento de processos, *deadlocks*, gestão de memória, multiprocessamento e multicomputação.

2. Objetivo

O objetivo principal deste simulador é proporcionar aos alunos uma visão detalhada e prática de como os diferentes componentes de um sistema computacional, como a CPU, a memória e os periféricos, interagem durante a execução de um programa. O simulador também permitirá a implementação de otimizações como o escalonamento de processos e a gestão de memória. Dessa forma, será possível avaliar o impacto dessas otimizações no tempo total de processamento e na eficiência global do sistema.

Além disso, o simulador permitirá que o aluno explore diferentes políticas de substituição de cache, analisando como essas políticas influenciam o desempenho do sistema em termos de acertos (*cache hits*) e falhas (*cache misses*). A integração do *pipeline* MIPS permitirá o estudo detalhado de como as instruções são processadas de maneira paralela em diferentes estágios, otimizando a utilização da CPU.

3. Componentes do Simulador

O simulador será composto pelos seguintes componentes, que serão modelados em C/C++:

- **CPU**: A unidade central de processamento será modelada com múltiplos *cores*, onde cada *core* terá seu próprio conjunto de registradores. O conjunto de instruções, a cache e o *pipeline* serão compartilhados entre os *cores*. Pode-se considerar como estrutura viável para a implementação desse componente listas e filas, bem como, vetores e/ou matrizes.

1. *Registradores*: São pequenas unidades de armazenamento na CPU que guardam dados temporários durante a execução das instruções. A forma mais prática para compor essa etapa é representá-los por uma matriz ou vetor em que cada posição detalha um registrador específico.
2. *Unidade de Controle (UC)*: Responsável por decodificar as instruções e gerar os sinais de controle necessários para coordenar as operações da CPU.
3. *Unidade Lógica Aritmética (ULA)*: Realiza as operações aritméticas e lógicas requisitadas pelas instruções, como somas, subtrações, operações lógicas (`AND`, `OR`, etc.).
4. *Contador de Programa (PC)*: Mantém o endereço da próxima instrução a ser executada pela CPU.

- *Conjunto de Instruções*: Define as operações que a CPU pode executar. As instruções são compostas por um *opcode* (código de operação) e operandos. Exemplos de instruções incluem `LOAD`, `STORE`, `ADD`, e `SUB`.
- **Pipeline**: O *pipeline* será baseado na arquitetura MIPS e modelado com os cinco estágios de execução de uma instrução:
  - **IF (Instruction Fetch)**: Busca a próxima instrução da memória e a armazena no registrador de instrução. O PC é incrementado.
  - **ID (Instruction Decode)**: Decodifica a instrução e carrega os operandos dos registradores.
  - **EX (Execute)**: A ULA executa a operação especificada pela instrução.
  - **MEM (Memory Access)**: A instrução pode requerer leitura ou escrita na memória principal.
  - **WB (Write Back)**: O resultado da operação é escrito de volta no registrador de destino.
- **Cache**: A memória *cache* será modelada como uma tabela associativa ou *hash*, compartilhada entre os *cores*. A cache usará um mapeamento direto com política de substituição FIFO (First-In, First-Out) e política de escrita *Write-back*, ou seja, os dados são escritos na memória principal apenas quando removidos da cache.
- **Memória Principal**: A memória principal será representada por um vetor, simulando a RAM. O simulador permitirá operações de leitura e escrita, simulando a comunicação entre a CPU e a memória.
- **Memória Secundária (Disco)**: Simulada por uma matriz de dados, representando um armazenamento permanente. Funções de leitura e escrita serão implementadas para gerenciar dados não voláteis.
- **Periféricos**: Serão representados por variáveis booleanas que indicam a disponibilidade de um periférico. Processos em execução podem requisitar o uso de periféricos, gerando eventos de entrada e saída (E/S).

4. Instruções Adicionais

Para auxiliar na implementação dos componentes descritos, seguem exemplos práticos de funções em C/C++ que simulam algumas das operações da CPU e da memória.

Registradores: Armazenam dados temporários utilizados durante a execução das instruções. No exemplo a seguir, são definidos 32 registradores.

```
int registradores[32]; // 32 registradores
registradores[0] = 10;  // Armazena 10 no registrador 0
registradores[1] = 20;  // Armazena 20 no registrador 1
int soma = registradores[0] + registradores[1];  // Soma
```

Unidade de Controle (UC): A unidade de controle decodifica o *opcode* e gera sinais de controle para as demais unidades da CPU.

```
void UnidadeControle(int opcode) {
    switch(opcode) {
        case 0: // ADD
            // Instruções para soma
            break;
        case 1: // SUB
            // Instruções para subtração
            break;
    }
}
```

Unidade Lógica Aritmética (ULA): Executa operações aritméticas e lógicas.

```
int ULA(int operando1, int operando2, char operacao) {
    if (operacao == '+') return operando1 + operando2;
    else if (operacao == '-') return operando1 - operando2;
}
```

Contador de Programa (PC): Mantém o endereço da próxima instrução a ser executada.

```
int PC = 0;
PC += 4;  // Incrementa PC para a próxima instrução
```

Conjunto de Instruções: Exemplos de operações que o simulador deve suportar, como `ADD` e `SUB`.

```
#RD: Registrador Destino
#R1: Registrador 1
#R2: REgistrador 2
registradores[RD] = registradores[R1] + registradores[R2];
```

5. Pipeline

Como descrito, o pipeline MIPS será implementado com os cinco estágios. A seguir, um exemplo de funções para cada estágio do pipeline:

```
void InstructionFetch() {
    instrucaoAtual = memoria[PC];
    PC += 4;  // Incrementa PC
}

void InstructionDecode() {
    // Decodifica a instrução e prepara a execução
}

void Execute() {
    // Executa a operação na ULA
}

void MemoryAccess() {
    // Leitura ou escrita na memória
}

void WriteBack() {
    // Escreve o resultado de volta nos registradores
}
```

6. Conclusão

O simulador proporcionará uma análise aprofundada dos princípios fundamentais da arquitetura de Von Neumann e da execução de instruções em pipeline, oferecendo aos usuários uma experiência prática e interativa. A implementação de uma hierarquia de memórias permitirá observar, de forma direta, os impactos das diversas estratégias de gerenciamento adotadas pelo sistema operacional para otimizar o uso desse recurso, além de evidenciar os benefícios da utilização da memória cache como intermediária entre a CPU e a memória principal. Adicionalmente, a integração de uma arquitetura multicore permitirá explorar conceitos avançados, como virtualização de *hardware* e gerenciamento de recursos, que são características essenciais de sistemas operacionais modernos. No que diz respeito à execução de processos, o simulador permitirá avaliar diferentes algoritmos de escalonamento, possibilitando a análise de aspectos como concorrência e contenção de recursos. Dessa forma, o simulador se configura como uma ferramenta educacional robusta e completa para o estudo dos sistemas computacionais contemporâneos.

Equipe: 4 alunos
Data de entrega: 29 de outubro de 2024
Forma de entrega: Link do Git
Pontos: 20 pontos