Centro Federal de Educação Tecnológica de Minas Gerais
Campus Divinópolis
Graduação em Engenharia de Computação

José Marconi de A. Júnior

Simulador Arquitetura Mips

Divinópolis
2025

Sumário

Capítulo 1

Metodologia

1.1 Instruções

O README no GitHub especifica como devem ser feitas as instruções. Para esta etapa do trabalho, foram inseridos novos arquivos de instruções (nomeados instructiosN.txt, para $N \geq 0$), enquanto é utilizado um mesmo arquivo para inicializar os registradores para todas as instruções.

Foi adotada a lógica de que cada arquivo será um processo e cada processo será uma thread. Sendo assim, cada processo/thread pode conter mais de uma linha, o que aumenta o seu tempo de execução.

1.2 Implementação

Foram criadas duas novas classes, denominadas Processos e Escalonador. Também foram realizadas modificações na classe Core, alterando a função antiga activate() para executeProcess().

A classe Processos cria os processos e realiza a leitura dos arquivos de instruções, efetuando a escrita na RAM (StructionsLoad()) e nos registradores, por meio do RegisterLoad() que chama uma função na unidade de controle responsável por acionar uma função da Pipeline.

O que é chamado de processo é o próprio PCB do mesmo, sendo composto por:

- Id: Identificador único do processo;
- State: estado de vida do processo;

- Quantum: quantum máximo que o processo pode utilizar;
- ActualInstruction: indice da instrução atual no vetor de instruções do processo;
- Regs: registradores utilizados na instrução atual;
- Files: nome do arquivo que contém as instruções sendo executadas.

O quantum utilizado foi fixado em 12 para todos os processos.

A clase contém a função execute() esponsável por executar as instruções (a primeira ação é alterar o estado do ciclo de vida para RUNNING), interagindo com a unidade de controle e a Pipeline. Também possui as funções block() e unlock() para alterar o estado do ciclo de vida do processo para BLOCK e READY, respectivamente.

O estado de vida do processo pode ser resumido como:

- READY: quando o processo está pronto para ser executado na fila;
- RUNNING: quando o processo está sendo executado;
- BLOCK: quando o quantum excede e o processo deve ser parado;
- TERMINATED: quando o processo é finalizado;

Sempre que o processo sai da execução, seu estado é verificado. Caso esteja em BLOCK, ele é colocado novamente no fim da fila; caso esteja em TERMINATED, ele é finalizado e retirado da fila.

1.2.1 Escalonador

A classe Scheduler implementa o escalonador, utilizando a política First Come First Service (FCFS), ou seja, o primeiro que entra é o primeiro a ser executado.

Primeiramente, o construtor é chamado na função main, recebendo as memórias e o vetor de Cores como referência. Ainda no construtor, é chamada a função createAndAddProcess() que cria um novo processo e o adiciona à fila de processos. Em seguida, a função schedule() é chamada.

Na função schedule(), a política é aplicada. Um mutex é utilizado para proteger a execução, e o vetor de threads a ser executado é definido. Há um laço while que roda enquanto a fila não estiver vazia ou enquanto algum Core estiver ocupado. Dentro dele,

há uma condicional que verifica se a fila não está vazia. Nesse caso, entra em um laço que percorre os Cores. Se um Core estiver disponível, uma thread é criada, chamando a função executeProcess() da classe Core, que será responsável pela execução do processo, passando as devidas referências. Logo após, o vetor de threads é iniciado com a função detach(), permitindo que as threads sejam executadas de forma independente.

1.2.2 Função executeProcess() da Classe Core

A função utiliza um mutex (coutMutex) para sincronizar o acesso à saída padrão. Isso é necessário em um ambiente multithreaded para evitar que múltiplas threads escrevam na saída padrão ao mesmo tempo, o que pode gerar mensagens misturadas e ilegíveis.

Nela, a função execute() da classe Process é chamada para realizar a execução do processo. Em seguida, o estado de vida do processo é analisado. Caso esteja em BLOCK, ele é reposicionado no fim da fila. Caso esteja em TERMINATED, é exibida uma mensagem indicando a conclusão do processo. Por fim, o Core é liberado para a próxima execução.

1.2.3 Função execute() Classe Process

A função retorna informações sobre o que está sendo executado, incluindo o ID do processo, a instrução, os registradores, o arquivo fonte e o clock ao final da execução da instrução.

O controle de quantum é feito por meio de um contador. Dentro da lógica, a execução de uma instrução pode ultrapassar o quantum máximo por alguns poucos ciclos de clock. Isso ocorre porque o controle desse clock não é realizado na Pipeline, mas apenas na instrução. Assim, o clock retornado da execução na Pipeline pode exceder ligeiramente o quantum.

De forma geral, a função controla o quantum e retorna o output, enquanto a execução das instruções é realizada com o uso da unidade de controle e da Pipeline.

Capítulo 2

Resultados

A saída foi conforme o esperado, considerando o que foi implementado:

```
Processo 1 sendo executado no core 1

[Processo 1] Executando instrução: PC=0 Opcode=0 Destino=R1 Valor1=10 Valor2=5
Arquivo fonte: data/instructions0.txt
clock = 13

--- PROCESSO BLOQUEADO, QUANTUM EXCEDIDO ---
Processo 1 bloqueado no Core 1


Processo 2 sendo executado no core 2


Processo 3 sendo executado no core 1

[Processo 2] Executando instrução: PC=0 Opcode=0 Destino=R1 Valor1=10 Valor2=5
Arquivo fonte: data/instructions1.txt
clock = 10

[Processo 2] Executando instrução: PC=4 Opcode=1 Destino=R4 Valor1=15 Valor2=5
Arquivo fonte: data/instructions1.txt
clock = 14

--- PROCESSO BLOQUEADO, QUANTUM EXCEDIDO ---
Processo 2 bloqueado no Core 2
```

Figura 1 – Exemplo de output.

É possível visualizar quando um processo foi bloqueado e em qual core isso aconteceu. Também é exibido em qual core cada processo está sendo executado. É importante lembrar que cada core possui um clock total diferente. O clock final total de cada core pode ser visualizado ao final dos últimos processos que foram executados e concluídos.

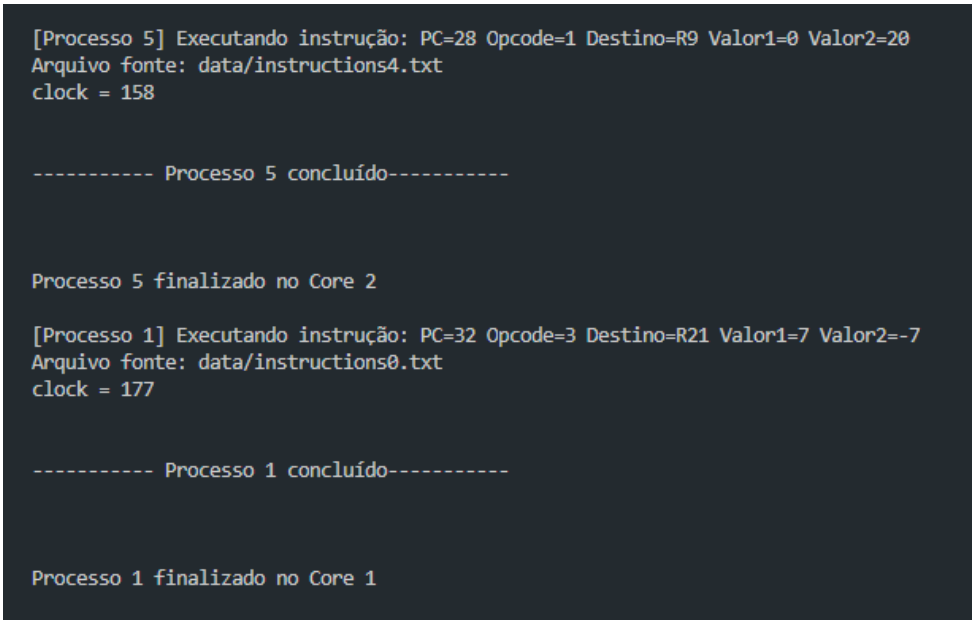

Figura 2 – Exemplo de output.

Observa-se que o core 1 realizou, ao todo, 177 ciclos de clock, enquanto o core 2 executou 158 ciclos de clock.

Por fim, é exibido o estado final da RAM e uma mensagem indicando que todos os processos foram finalizados:

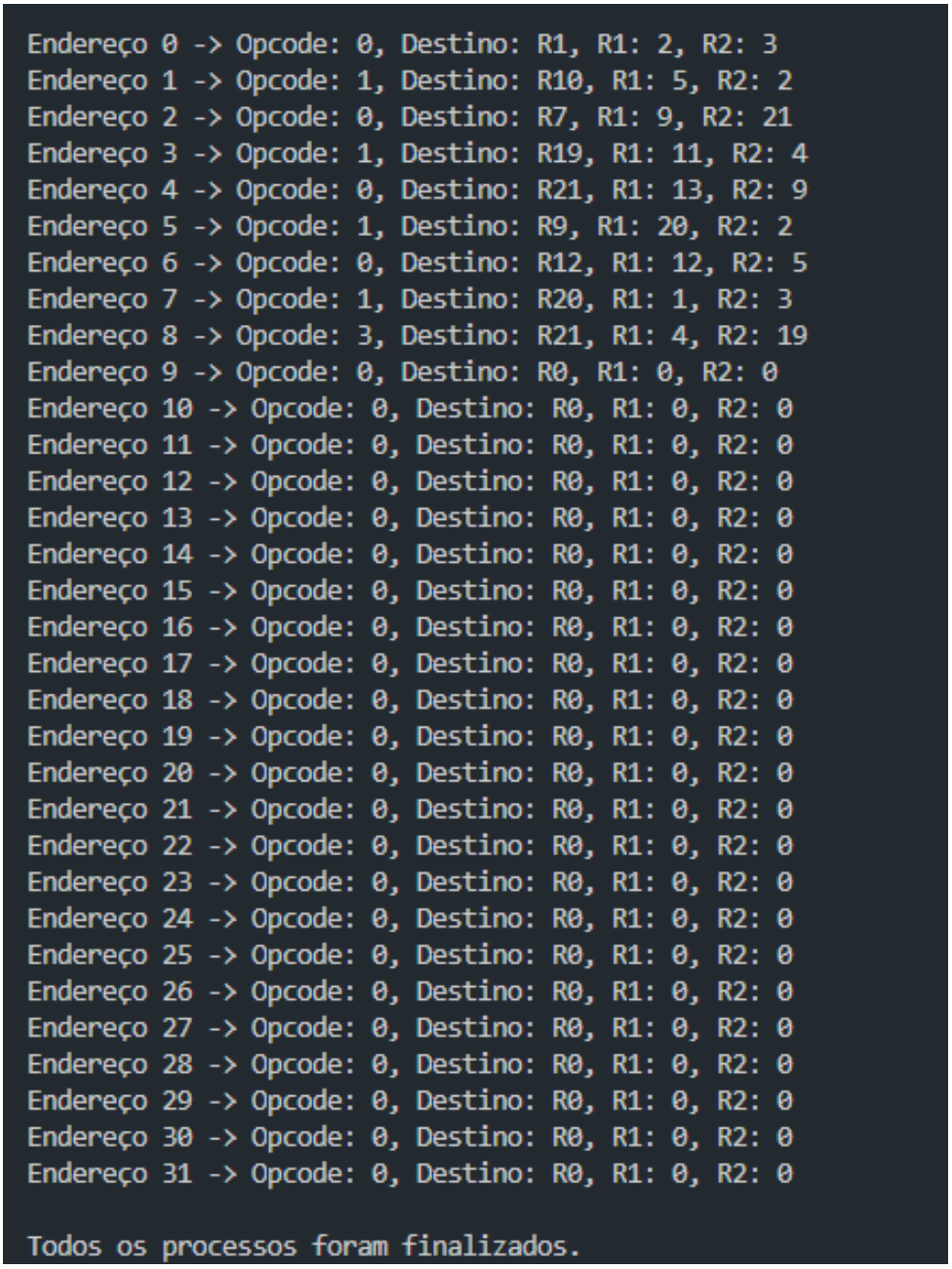

Figura 3 – Exemplo da ram ao final da execução geral.