Simulador: Arquitetura Von Neuman e Pipeline Mips

1st José Marconi de A. Júnior
Engenharia de Computação
Centro Federal de Educação Tecnológica (CEFET)
Divinópolis, MG
jmarconiadm@gmail.com

***Abstract*—**
***Index Terms*—.**

I. Introduction

O desenvolvimento de sistemas computacionais exige uma compreensão profunda tanto do hardware quanto do software que os suporta. A arquitetura von Neumann, fundamental para a computação moderna, define a base para a interação sequencial entre memória e processador, enquanto o pipeline, como implementado na arquitetura MIPS, permite a execução paralela e eficiente de instruções.

Este artigo apresenta o desenvolvimento de um simulador de sistema operacional que emula, por meio de código, uma arquitetura von Neumann com suporte a pipeline MIPS. A emulação inclui a implementação de uma CPU, abrangendo unidade de controle, opcodes, instruções e uma estrutura modular que integra RAM, Disco, Cache, pipeline e periféricos. O objetivo principal é oferecer uma plataforma para explorar, testar e compreender os princípios fundamentais dessas arquiteturas e sua relação com sistemas operacionais.

II. Fundamentação Teórica

A. Arquitetura Von Neumann

A arquitetura de Von Neumann é um modelo de computador clássico que organiza os componentes do sistema de forma a executar programas armazenados na memória (figura 1). Essa organização é a base para a maioria dos computadores modernos.

1) CPU: A CPU é considerado o ”cérebro” do sistema. Ela executa as instruções armazenadas na memória principal. É composta principalmente por:

- Contador de Programa (Program Counter): guarda o endereço da próxima instrução a ser executada, geralmente sendo um registrador especial do banco de registradores;
- Banco de Registradores(Registros): pequenas áreas de armazenamento temporário, usadas para armazenar dados intermediários e facilitar cálculos;

Identify applicable funding agency here. If none, delete this.

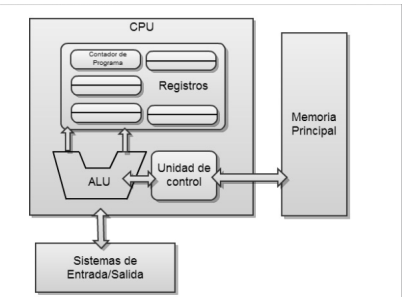

Fig. 1. Exemplo ilustrativo de uma arquitetura clássica de Von Neumann.

- ALU (Unidade Lógica e Aritmética): realiza operações matemáticas (soma, subtração, etc.) e lógicas (comparações como AND e OR, por exemplo);
- Unidade de Controle: coordena as operações do sistema. Ela decodifica as instruções da memória e controla o fluxo de dados entre os componentes da CPU, memória principal e dispositivos de entrada/saída.

2) Memória Principal (RAM): É onde os dados e instruções do programa são armazenados. Na arquitetura von Neumann, memória é unificada, ou seja, tanto os dados quanto as instruções do programa compartilham o mesmo espaço de armazenamento.

3) Sistemas de Entrada/Saída: Representam os dispositivos que permitem a interação com o sistema (teclado, monitor, discos, etc.). A CPU controla esses dispositivos por meio da Unidade de Controle.

4) Fluxo de Dados:

- Entre CPU e Memória Principal: as instruções e dados são buscados da memória para serem processados pela CPU. Após o processamento, os resultados podem ser armazenados de volta na memória;
- Entre CPU e Dispositivos de Entrada/Saída: A CPU envia comandos para os dispositivos e recebe dados deles.

B. Pipeline Mips

A pipeline MIPS (figura 2) é uma técnica usada em arquiteturas baseadas no conjunto de instruções MIPS (Micro-

processor without Interlocked Pipeline Stages) para melhorar o desempenho do processador. Essa técnica divide a execução de uma instrução em várias etapas, permitindo que várias instruções sejam processadas simultaneamente em diferentes estágios da CPU.

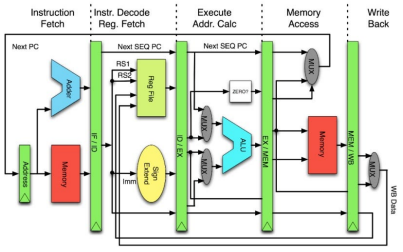

Fig. 2. Exemplo ilustrativo da Pipeline Mips e seus estágios.

1) Estágios da Pipeline:

- IF (Instruction Fetch - Busca da Instrução):
  - A instrução é buscada da memória usando o endereço armazenado no contador de programa (PC);
  - O PC é incrementado para apontar para a próxima instrução.
- D (Instruction Decode/Register Fetch - Decodificação da Instrução):
  - A instrução é decodificada;
  - Os operandos são lidos dos registradores (se necessário);
  - Determina-se o tipo de operação (aritmética, lógica, memória, etc.).
- EX (Execute - Execução):
  - A ALU realiza a operação especificada pela instrução;
  - Exemplo: Soma, subtração, ou cálculo do endereço efetivo (para instruções de memória).
- MEM (Memory Access - Acesso à Memória):
  - Instruções de leitura/escrita acessam a memória;
  - Outras instruções ignoram este estágio.
- WB (Write Back - Escrita no Registrador):
  - O resultado da operação é escrito de volta em um registrador.

A pipeline MIPS permite que múltiplas instruções sejam executadas ao mesmo tempo. Enquanto uma instrução está sendo buscada (IF), outra está sendo decodificada (ID), outra está na execução (EX), e assim por diante. Isso resulta em um aumento significativo na taxa de transferência de instruções.

III. Metodologia

O objetivo inicial foi integrar os conceitos fundamentais de ambas as arquiteturas para desenvolver um emulador com uma estrutura mais robusta. Durante o desenvolvimento, foram incorporadas modificações em cada etapa para aproximar o sistema operacional de um ambiente real, incluindo conceitos de multicore, multithreading e a implementação de políticas de escalonamento.

A. Modificações

1) Multicore: O conceito de ”multicore” refere-se a um processador que contém dois ou mais núcleos de processamento. Cada núcleo pode executar instruções de maneira independente, funcionando como se fosse um processador separado. Essa arquitetura permite a execução simultânea de múltiplas tarefas ou threads, aumentando significativamente o desempenho e a eficiência do sistema (figura 3).

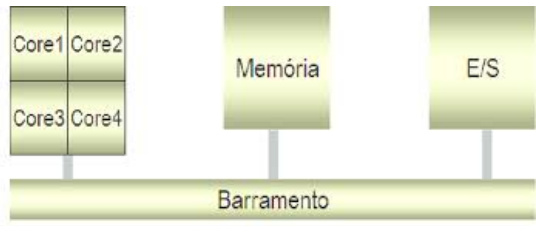

Fig. 3. Exemplo ilustrativo básico de uma estrutura multicore.

No contexto do sistema implementado, cada núcleo (Core) terá sua própria CPU, juntamente com todos os componentes necessários, e um clock exclusivo. A memória principal (RAM), por outro lado, será compartilhada entre todos os núcleos em operação.

Com essa estrutura, o objetivo é permitir que os processos concorram pelos núcleos, disputando os recursos para sua execução.

2) Multithreading: O conceito de multithreading refere-se à capacidade de um processador ou sistema operacional de executar múltiplas threads dentro de um único processo de forma concorrente. Uma thread é a menor unidade de execução de um programa, contendo seu próprio conjunto de instruções, registros e pilha de execução.

Ao adotar o multithreading, um programa pode dividir seu trabalho em várias threads, permitindo que elas sejam executadas simultaneamente em sistemas com múltiplos núcleos.

Dessa forma, como o sistema foi projetado para suportar múltiplos núcleos, é possível distribuir as threads entre os núcleos disponíveis, aproveitando de maneira mais eficiente os recursos disponíveis e aumentando o desempenho na execução de tarefas paralelizadas.

Para implementar o multithreading, foi utilizada a biblioteca ¡thread¿ do C++, que oferece as funcionalidades necessárias para essa aplicação.

3) Quantum e Clock:

- Clock: é a unidade de ”tempo” utilizada pelo sistema. Cada ciclo de clock é considerado uma unidade de tempo, ou seja, cada ciclo do clock é um ”Clock”. Cada etapa do pipeline usa um ciclo de clock, e essa medida será utilizada para avaliar a eficiência de cada escalonador no sistema, além do ”tempo” total gasto em cada Core utilizado;
- Quantum: é o número de ciclos de clock alocados para a execução de cada processo. O quantum fixo associado a cada processo determina o número máximo de ciclos de clock que um processo pode usar antes de ser bloqueado.

Também existe o ”quantum máximo”, que é o número total de ciclos de clock usados por um processo para completar sua execução.

O quantum atribuído a cada processo é definido pelo somatório dos pesos de suas instruções, como apresentado na Tabela 1. Esses pesos representam, de forma aproximada, o número de ciclos de clock necessários para a execução de cada instrução. Embora o peso não corresponda exatamente ao total de ciclos utilizados por cada instrução, ele reflete um valor representativo, quanto maior o número de ciclos maior o peso atribuído.

TABLE I
REFERENTE AOS PESOS ASSOCIADOS DE ACORDO COM CADA INSTRUÇÃO.

Instrução	Peso
ADD	3
SUB	3
AND	3
OR	3
STORE	2
LOAD	2
ENQ	2
IF igual	2

Ao avaliar a prioridade dos processos, observa-se que, quanto menor a soma dos pesos, menor o quantum atribuído ao processo e maior o número de ”bilhetes de loteria”, conforme ilustrado na Tabela 2.

TABLE II
REFERENTE AOS QUANTUS E BILHETES ASSOSCIADOS

Somatório	Quantum	Bilhetes
$0 < Soma \leq 10$	10	5
$11 \geq Soma \leq 20$	15	3
$Soma \geq 21$	20	1

Portanto, o sistema de prioridade adotado segue a lógica de que processos com menores quantums possuem maior prioridade de execução.

4) Escalonamento: O escalonamento de threads é o processo pelo qual o sistema operacional decide qual thread deve ser executada em um dado momento. O objetivo do escalonamento é gerenciar a execução concorrente de várias threads de forma eficiente, garantindo que os recursos do processador sejam utilizados de maneira otimizada.

Existem várias estratégias de escalonamento de threads, para diferentes tipos de sistema. Como a implemnetação em questão, é um sistema em lote, a principio foi utilizado um escalonamento básico de First Come First Service(FCFS) com preempção. Posteriormente foi implementado outras duas outras formas de escalonamento para realizar testes comparativos, foram eles Small Job First(SJF) e Lottery(loteria).

- FCFS: as threads são executadas na ordem de chegada, a partir de uma fila, e cada uma recebe um quantum máximo fixo, determinado pelas instruções que há nesse arquivo, cada instrução tem um peso diferente, logo, o quantum é definido em cima da prioridade total. Quando o quantum é excedido e o processo não é concluído, a thread é bloqueada e o processo é colocado novamente no final da fila. Assim que o processo é concluído, ele é removido da fila;
- SJF: a implementação é semelhante à do FCFS, porém, em vez de uma fila comum, foi utilizada uma fila de prioridade, na qual a prioridade de cada processo está relacionada ao quantum fixo que ele recebe. Quanto menor o quantum, maior a prioridade do processo, pois indica que se trata de um processo de curta duração. Se o quantum for excedido e o processo não for concluído, ele é bloqueado e recolocado na fila. Caso o processo tenha uma prioridade maior, a fila é reorganizada, colocando os processos de maior prioridade à frente para serem executados primeiro;
- Lottery: no escalonamento por lotaria, cada processo recebe um número de bilhetes proporcional à sua prioridade, que, por sua vez, está relacionada ao quantum. Quanto menor o quantum, maior o número de bilhetes atribuídos ao processo. Em seguida, é realizado um sorteio aleatório para determinar qual processo será executado, levando em consideração o peso dos bilhetes. Após a execução, os processos que permanecem na fila recebem mais bilhetes. Quando um processo é bloqueado, ele é recolocado na fila e recebe um novo número de bilhetes, continuando o ciclo até a execução de todos os processos,

IV. RESULTADOS

A saída no terminal exibe as principais informações sobre cada instrução executada por cada processo, (PCB). Além disso, são exibidas mensagens detalhando quando um processo é bloqueado, indicando qual processo foi afetado e em qual Core isso ocorreu. Também são apresentadas notificações informando quando um processo é finalizado, juntamente com o Core responsável pela execução, como o exemplo a seguir:

```
[Processo 1] Executando instrução:
 PC=4
 Opcode=1
 Destino=R10 Valor1=0 Valor2=10
State:1
Arquivo fonte: data/instructions0.txt
Quantum total utilizado: 20
clock = 20

Processo 1 bloqueado no Core 1
```

A saída permite visualizar diversas informações detalhadas sobre a execução dos processos:

- Processo em execução (linha 1): Indica qual processo está sendo executado no momento.
- Endereço da instrução (linha 2): Exibe o endereço da instrução que está em execução.
- Instrução (linha 3): Mostra o opcode correspondente à instrução.

- Registradores utilizados (linha 4): Apresenta os registradores envolvidos, incluindo o registrador de destino e os valores dos registradores utilizados.
- Estado do processo (linha 6): Indica o estado de vida do processo (1 = RUNNING).
- Arquivo fonte (linha 7): Exibe o arquivo fonte ao qual a instrução pertence.
- Quantum utilizado (linha 8): Mostra o total de quantum consumido pelo processo até o momento.
- Clock total do Core (linha 9): Informa o total de ciclos de clock já acumulados pelo Core.
- Mensagem de bloqueio (última linha): Indica que o processo foi bloqueado no Core 1.

A. Comparações

Todos os processos foram concluídos com sucesso utilizando ambos os núcleos. As tabelas a seguir apresentam o quantum máximo de ciclos de clock permitidos para cada processo, o total de ciclos efetivamente utilizados por cada núcleo e o tempo de execução correspondente: Tabela 3 para o escalonamento FCFS, Tabela 4 para SJF e Tabela 5 para o escalonamento de Loteria. Observa-se que a soma total dos ciclos de clock dos processos deve ser igual à soma total dos ciclos de ambos os núcleos utilizados. Vale ressaltar que os testes foram realizados separadamente, mas é possível executar todos os processos simultaneamente no código. No entanto, como os núcleos e o quantum máximo não são resetados, ao rodar todos os processos juntos, os valores serão a soma total de todos os escalonadores combinados.

TABLE III
CICLOS DE CLOCK TOTAIS E TEMPO DE EXECUÇÃO - FCFS

Processo / Core	Clock
P1	94
P2	62
P3	18
P4	33
P5	67
P6	22
Total - P	296
Core 1	149
Core 2	147
Total - Cores	296
Tempo de execução	0.09297 s

TABLE IV
CICLOS DE CLOCK TOTAIS E TEMPO DE EXECUÇÃO - SJF

Processo / Core	Clock
P1	74
P2	62
P3	18
P4	33
P5	67
P6	22
Total - P	276
Core 1	142
Core 2	134
Total - Cores	276
Tempo de execução	0.082987 s

TABLE V
CICLOS DE CLOCK TOTAIS E TEMPO DE EXECUÇÃO - LOTERIA

Processo / Core	Clock
P1	54
P2	62
P3	18
P4	33
P5	67
P6	22
Total - P	256
Core 1	119
Core 2	137
Total - Cores	256
Tempo de execução	0.0143661 s

B. Conclusões

A partir dos resultaddos optidos, podemos visualizar alguns resultados importantes com relação ao que foi proposto:

- Em todos os algoritmos de escalonamento (FCFS, SJF e Loteria), os processos foram distribuídos entre os dois núcleos (Cores), garantindo o uso paralelo e eficiente dos recursos do sistema.
- Nos algoritmos FCFS e SJF, a carga de trabalho está bem equilibrada entre os dois núcleos, com uma diferença muito pequena no total de ciclos executados por cada núcleo. A diferença é inferior a 1% no caso do FCFS e de apenas 2,9% no caso do SJF. Já no algoritmo de Loteria, observa-se uma diferença maior, de 7,04%, o que é esperado devido à natureza probabilística do escalonamento.
- A partir dos valores de ciclos totais e tempo de execução, é possível analisar as relações entre os ciclos totais, o tempo de execução e a vazão dos processos:
  1) Considerando o total de ciclos, observa-se uma diferença de 6,2% entre FCFS e SJF, e de 13,5% entre Loteria e FCFS. Já a diferença entre Loteria e SJF é de 6,7%.
  2) Em relação ao tempo de execução, a diferença entre FCFS e SJF é de 9,8%, enquanto que entre FCFS e Loteria, a diferença é de 84,6%. A diferença entre Loteria e SJF é de 73,8%
  3) Com base nas métricas anteriores, podemos estimar a vazão de cada escalonador. Observa-se uma diferença de 1,9% a mais de vazão no SJF em comparação ao FCFS, enquanto as diferenças de vazão entre Loteria e os outros escalonadores são bastante significativas: 84,6% a mais em relação ao FCFS e 82,7% a mais em relação ao SJF.

Os resultados dos testes com os algoritmos de escalonamento FCFS, SJF e Loteria mostram diferenças significativas no desempenho, tanto em termos de tempo de execução quanto de vazão dos processos. O algoritmo FCFS, embora simples e intuitivo, apresenta o maior tempo de execução e a menor vazão, indicando que não é a escolha mais eficiente para a distribuição de carga de trabalho. O SJF, por outro lado, apresentou um desempenho mais equilibrado, com uma

diferença menor em relação ao FCFS em termos de tempo de execução e uma leve melhoria na vazão de processos.

O algoritmo de Loteria, devido à sua natureza probabilística, demonstrou a maior vazão de processos e o menor tempo de execução, superando significativamente os outros dois algoritmos. No entanto, essa superioridade pode variar em situações diferentes, já que o desempenho do algoritmo de Loteria depende da aleatoriedade na alocação dos processos, podendo apresentar resultados inconsistentes em alguns casos.

De modo geral, os resultados sugerem que, para cenários em que a eficiência e a rapidez na execução são cruciais, o escalonador baseado em Loteria pode ser mais vantajoso, embora sua performance dependa de fatores aleatórios. O SJF oferece um bom equilíbrio entre desempenho e previsibilidade, enquanto o FCFS pode ser considerado menos eficiente, especialmente em sistemas com uma carga de trabalho mais variável.

REFERENCES

[1] W. Stallings, "Arquitetura e Organização de Computadores," 8th ed. Always Learning, Nov. 2010.

[2] A.S.Tanenbaum and T.Austin, Structure Computer Organization, 6th ed., Pearson, 2013

[3] A. José Marconi, "Simulador Mips: Sistemas Operacionais: Simulador da Arquitetura de Von Neumann e Pipeline MIPS", Disponível em: https://github.com/josemarconi/Simulador-Mips