Minix SO

1st Felipe Parreiras Dias
Departamento de Informática Gestão e Design
CEFET-MG
Divinópolis, MG
felipeparreiras86@gmail.com

2nd Rafael Guimarães Gontijo
Departamento de Informática Gestão e Design
CEFET-MG
Divinópolis, MG
rafaelg000@gmail.com

***Resumo*—Este trabalho aborda a implementação de uma máquina virtual baseada na arquitetura MIPS com pipeline de cinco estágios, com o objetivo de explorar aspectos fundamentais de sistemas operacionais, como escalonamento de processos e gerência de memória. Foi ainda adicionado suporte a multicore, de forma a executar processos paralelamente, escolhidos com base no algoritmo de escalonamento primeiro a chegar, primeira a ser servido. A máquina virtual demonstrou eficiência ao executar programas exemplo, validando a funcionalidade do pipeline e a interação dos processos no sistema.**

***Palavras-chave*—máquina virtual, sistemas operacionais, MIPS, multicore, escalonamento**

I. Introdução

Um computador é formado por diversas partes de *hardware* e, com isso, torna-se um sistema muito complexo para realizar o desenvolvimento e gerenciamento destas partes. Por isso, tem-se o sistema operacional (SO), que tem como função gerenciar os recursos do computador e abstrair conceitos complicados relacionados ao *hardware*, com o objetivo que oferecer um ambiente melhor para os desenvolvedores.

Dentro do SO, são realizados trabalhos relacionados ao escalonamento de processos, gerência de memória, dentre outros. Para entender melhor o funcionamento de um SO, este trabalho tem por objetivo a construção de uma máquina virtual que apresenta estes e outros aspectos importantes relacionados a SO. Com isso, espera-se adquirir uma melhor compreensão do objeto aqui estudado.

Par implementar a máquina virtual, utilizou-se a arquitetura MIPS com um pipeline de cinco estágios: *Instruction Fetch* - IF; *Instruction Decode* - ID; *Execute* - EX; *Memory Access* - *MEM*; *Write Back* - WB. O MIPS é uma arquitetura ideal porque é um arquitetura real enviada em milhões de produtos anualmente, mas é simplificada e fácil de aprender [1].

O desenvolvimento da máquina virtual foi dividido em diversas etapas, onde, a cada passo, se implementa uma função importante de um SO. Assim, é possível focar individualmente nas tarefas executadas por um sistema operacional, garantindo um melhor entendimento de um SO como um todo. Deste modo, o artigo expõe as etapas do desenvolvimento e discute os principais conceitos envolvidos, buscando conectar a teoria à prática.

II. Metodologia

A. Simulador da Arquitetura de Von Neumann e Pipeline MIPS

A arquitetura MIPS é uma arquitetura do microprocessador, definido por seus registros e linguagem *assembly* conjunto de instruções, e simples o suficiente para entender claramente e construir. A linguagem *assembly* é um representação de linguagem nativa do computador legível para humanos. Cada instrução especifica a operação a ser realizada, assim como os operandos necessários para realizar a instrução [1].

O conjunto de instrução MIPS é, de modo geral, executada rapidamente. O número de instruções é pequeno, para que o *hardware* necessário para decodificar as instruções seja simples e rápido. Algumas operações mais complexas são realizadas utilizando múltiplas operações simples. Assim, o MIPS é uma arquitetura *reduced instruction set computer* (RISC).

As instruções precisam acessar os operandos rapidamente para que possam ser executadas no menor tempo possível. Porém, operandos armazenados na memória primária gastam muito tempo para serem acessados. Portanto, a maioria das arquiteturas especifica um pequeno número de registradores que armazenam dados operandos comumente usados. A arquitetura MIPS usa 32 registros, chamados de conjunto de registros ou *register file*. Quanto menos registros, mais rápido eles podem ser acessado.

Se os registradores fossem o único espaço de armazenamento para operandos, os programas estariam limitados a apenas 32 variáveis. No entanto, dados também pode ser armazenado na memória. Quando comparado com o arquivo de registro, a memória tem muitos locais de dados, mas acessá-los leva mais tempo tempo. Considerando que o *register file* é pequeno e rápido, a memória é grande e lenta. Por isso, as variáveis mais utilizadas durante a execução do programa são mantidas em registros. Usando uma combinação de memória e registradores, um programa pode acessar um grande quantidade de dados com bastante rapidez.

O pipeline do MIPS (figura 1) permite executar várias instruções simultaneamente, melhorando significativamente o rendimento do processador. O *pipelining* deve adicionar lógica para lidar com as instruções em execução simultaneamente em seus estágios. A complexidade da lógica e os registros

adicionados valem a pena, visto que todos os processadores comerciais de alto desempenho usam pipeline hoje.

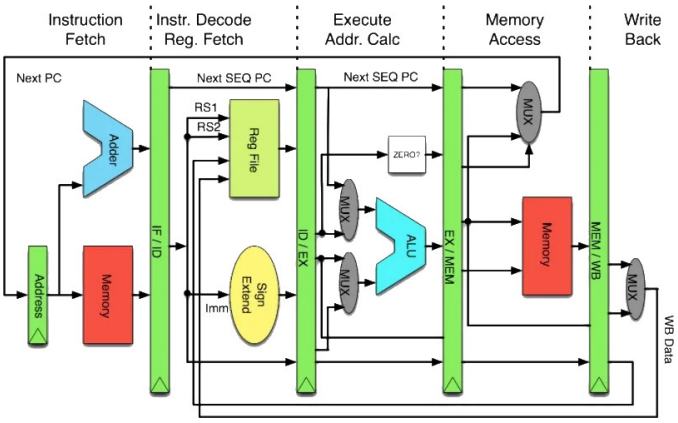

Figura 1. MIPS datapath

O *pipelining* é uma maneira poderosa de melhorar o rendimento de um sistema digital. Projetamos um processador em pipeline subdividindo o processador de ciclo único em cinco estágios de pipeline. Por isso, cinco instruções podem ser executadas simultaneamente, uma em cada estágio. Porque cada estágio tem apenas um quinto de toda a lógica, a frequência do clock é quase cinco vezes mais rápido. Portanto, a latência de cada instrução é idealmente inalterado, mas o rendimento é idealmente cinco vezes melhor.

Os cinco estágios do pipeline são chamados de *Fetch*, *Decode*, *Execute*, *Memory Access* e *Writeback*. No estágio *Fetch*, o processador lê a instrução da memória de instruções. Na fase de *Decode*, o processador lê os operandos fonte do arquivo de registro e decodifica a instrução para produzir os sinais de controle. No estágio *Execute*, o processador realiza um cálculo com a unidade lógica e aritmética (ULA). No Estágio de *Memory Access*, o processador lê ou grava dados na memória. Finalmente, no Estágio de *Writeback*, o processador grava o resultado no arquivo de registro, quando aplicável.

Vale destacar que, a fim de simplificar a parte de arquitetura, que não é o foco do trabalho, mas sim o SO, foi implementada uma versão mais simples da arquitetura MIPS. O conjunto de instruções é bem pequeno, de modo a realizar apenas as instruções mais básicas para verificar o funcionamento dos processos dentro do SO.

B. Arquitetura Multicore e Suporte a Preempção

Até um certo momento, os fabricantes de processadores ampliavam o poder de processamento apenas aumentando o número de transistores e a frequência dos processadores. Porém, tal solução tornava a tarefa de gerenciar a energia e o resfriamento dos processadores cada vez mais difícil [2].

Com isso, surgiu a ideia de diminuir a frequência dos processadores e aumentar o número destes em cada processador, gerando ganhos de processamento sem aumentar significativamente a geração de calor e o consumo de energia.

Utilizando threads, foi possível implementar os núcleos do processador da máquina virtual. É possível modificar o número de núcleos, para observar como o SO se comporta com um ou múltiplos núcleos.

Um computador multiprogramado possui, muitas das vezes, possui múltiplos processos rodando na CPU ao mesmo tempo. Se houver apenas uma CPU disponível, o SO deverá realizar uma escolha para determinar qual processo deve ser executado em seguida [3].

A parte do SO que faz esta escolha é chamada de escalonador, e o algoritmo que este utilizada é denominado algoritmo de escalonamento.

Dadas as diversas situações de diferentes áreas de aplicação, existem três ambientes para executar o escalonamento: em lote, interativo e de tempo real.

Neste trabalho, foi aplicado o escalonamento em lote. Em sistemas de lote, não há usuários esperando altas velocidades em seus terminais. Por isso, é aceitável o uso de algoritmos não-preemptivos ou algoritmos preemptivos com longos períodos de tempo para cada processo.

O escalonador ainda trabalha de forma preemptiva, ou seja, escolhe um processo e o deixa executar por no máximo um tempo fixado, denominado quantum. Se esse processo ainda estiver executando quando o quantum expirar, ele é suspenso e o escalonador escolhe outro processo para ser executado.

O algoritmo de escalonamento utilizado é denominado primeiro a chegar, primeira a ser servido (*first come, first service*). Ele é um fila simples, onde o próximo processo na fila é executado.

III. Resultados

Como exemplo para demonstrar os resultados, foi executado o programa a seguir carrega o número 10 no registrador $t3, o decrementa e verifica se ele é maior que zero; se for verdade, ele volta para a linha dois, criando um loop de 10 repetições no programa.

```
1  li   $t3 10
2  dec  $t3
3  bgtz $t3 2
4  halt
```

Após execução na máquina virtual Minix, foi gerada a seguinte saída, demonstrando as instruções percorrendo a pipeline.

```
CLOCK: 1

GENERAL-PURPOSE REGISTERS
$zero: 0    $at: 101    $v0: 102    $v1: 103
$a0: 104    $a1: 105    $a2: 106    $a3: 107
$t0: 108    $t1: 109    $t2: 110    $t3: 111
$t4: 112    $t5: 113    $t6: 114    $t7: 115
$s0: 116    $s1: 117    $s2: 118    $s3: 119
$s4: 120    $s5: 121    $s6: 122    $s7: 123
$t8: 124    $t9: 125    $k0: 126    $k1: 127
$gp: 128    $sp: 129    $fp: 130    $ra: 131

PIPELINE
```

```
IF:  add  $t3  $t1  $t2
ID:  nop
EX:  nop
MEM: nop
WB:  nop

MEMORY DATA
empty
============================================================
[...]
```

IV. Conclusões

Com isso, é possível concluir que a máquina virtual Minix cumpriu os seus requisitos, de forma a implementar um mini conjunto de instruções baseado na arquitetura MIPS. Assim, foi possível realizar o escalonamento na máquina multicore, de modo a fazer o chaveamento de processos baseados no tempo de execução na CPU.

Referências

[1] HARRIS, Sarah; HARRIS, David. Digital design and computer architecture. Morgan Kaufmann, 2015.

[2] MA, Josué. Multicore. Instituto de Computação - Unicamp. Campinas, Brasil. 2016.

[3] TANENBAUM, Andrew S.; WOODHULL, Albert S. Sistemas Operacionais: Projetjos e Implementaçao. Bookman Editora, 2009.