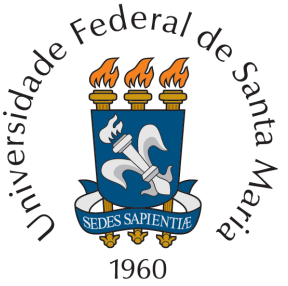

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA

CIÊNCIA DA COMPUTAÇÃO

LUCAS BOTH STEINMETZ RIBEIRO
MURILO HESSE BLOCK

SIMULADOR DE INSTRUÇÕES MIPS

SANTA MARIA/RS
2024

INTRODUÇÃO

O Assembly é uma linguagem de programação de baixo nível, característica que permite uma interação direta com o hardware do computador. Diferente das linguagens de alto nível, que abstraem muitas operações internas, o Assembly proporciona um controle detalhado sobre as operações realizadas pelo processador, permitindo ao programador manipular diretamente registradores, memória e outros componentes fundamentais da arquitetura do computador. Isso faz com que o Assembly seja uma ferramenta poderosa para tarefas que exigem desempenho otimizado e uso eficiente dos recursos do hardware.

Neste contexto, desenvolveu-se para o trabalho um programa em Assembly para a arquitetura MIPS, muito estudada e utilizada em contextos educacionais e de sistemas embarcados. O programa criado simula um conjunto de instruções, lendo um arquivo binário, decodificando suas instruções e executando-as. A tarefa escolhida para demonstrar essa funcionalidade é o cálculo do fatorial de um número, uma operação matemática simples, mas que envolve repetição e manipulação de dados, ilustrando bem o funcionamento das instruções em baixo nível.

Dessa forma, a escolha da linguagem Assembly e da arquitetura MIPS, bem como da tarefa específica do cálculo do fatorial, visa aprofundar o conhecimento em programação de baixo nível e na arquitetura de computadores. Outrossim, o trabalho demonstra a importância da manipulação direta de bits e bytes e como essa habilidade pode ser crucial para otimização e controle detalhado de sistemas. Por fim, o projeto reforça a compreensão dos fundamentos da interação entre software e hardware.

1. Objetivos

O objetivo primário deste trabalho é desenvolver um programa em Assembly MIPS que seja capaz de ler, decodificar e executar instruções a partir de um arquivo binário. A escolha do Assembly MIPS deve-se à sua relevância tanto em ambientes educacionais quanto em sistemas embarcados, onde a eficiência e o controle são cruciais. O programa terá como tarefa específica o cálculo do fatorial de um número, uma operação que, apesar de simples, oferece uma rica oportunidade de explorar conceitos fundamentais de programação de baixo nível e manipulação de dados.

Especificamente, este projeto busca aprofundar o entendimento dos conceitos de leitura e manipulação de arquivos binários. Isso envolve não apenas a capacidade de abrir e ler arquivos, mas também a interpretação dos dados binários neles contidos. A decodificação das instruções requer uma análise cuidadosa do formato binário e uma tradução precisa para operações que o processador MIPS possa executar. Assim, o projeto explora a implementação da lógica necessária para essa decodificação, um passo crucial para a execução correta das instruções.

Ao final, espera-se que o programa seja capaz de realizar a leitura correta dos arquivos, interpretar suas instruções e executar as operações conforme previsto. Para validar a funcionalidade do programa, serão realizados testes comparando os resultados obtidos com os valores esperados. O sucesso desses testes demonstrará a eficácia do programa e confirmará que os objetivos foram atingidos. Além disso, o projeto servirá como base para futuras explorações e aprimoramentos na área de programação em Assembly e simulação de instruções em arquiteturas específicas como a MIPS.

2. Revisão bibliográfica

A linguagem Assembly e a arquitetura MIPS desempenham papéis fundamentais na compreensão dos conceitos de programação de baixo nível e da operação do hardware de computadores. Nesse sentido, esta revisão, busca estabelecer uma base teórica sólida que sustente a implementação e validação do simulador de instruções MIPS desenvolvido neste projeto.

2.1 Introdução à Linguagem Assembly

A linguagem Assembly é conhecida por ser uma linguagem de programação de baixo nível, que permite uma interação direta com o hardware do computador. Ao contrário das linguagens de alto nível, que abstraem a maioria dos detalhes do hardware, a programação em Assembly proporciona um controle detalhado sobre as operações realizadas pelo processador. De acordo com Stallings (2018), a linguagem Assembly é crucial para entender os princípios fundamentais da arquitetura de computadores e da operação do hardware.

2.2 Arquitetura MIPS

A arquitetura MIPS (Microprocessor without Interlocked Pipeline Stages) é uma das arquiteturas RISC (Reduced Instruction Set Computing) mais estudadas e implementadas em ambientes acadêmicos e industriais. Patterson e Hennessy (2013) descrevem a arquitetura MIPS como uma das mais influentes na evolução dos processadores RISC. A simplicidade e a eficiência da MIPS a tornam uma excelente escolha para o ensino de conceitos de arquitetura de computadores e programação em Assembly.

2.3 Simulação de Processadores

Ferramentas de simulação, como o MARS (MIPS Assembler and Runtime Simulator), desempenham um papel vital no ensino e na pesquisa de arquitetura de computadores. De acordo com MIPS Technologies (2003), o MARS permite a simulação detalhada de programas escritos em Assembly MIPS, facilitando a visualização e a depuração das instruções e da memória. O uso do MARS em ambientes acadêmicos tem se mostrado eficaz para o ensino prático de conceitos teóricos complexos.

Referências:

- Patterson, D. A., & Hennessy, J. L. (2013). Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann.
- Stallings, W. (2018). Computer Organization and Architecture. Pearson.
- MIPS Technologies. (2003). MIPS32 Architecture for Programmers Volume I: Introduction to the MIPS32 Architecture. MIPS Technologies.

3. Metodologia

O desenvolvimento do simulador de instruções MIPS foi realizado em várias etapas, envolvendo a leitura do arquivo binário, a implementação de uma memória virtual, a criação de máscaras de bits para a decodificação das instruções e a execução das mesmas. A seguir, detalhamos cada uma dessas etapas:

3.1 Leitura do arquivo binário e do arquivo de dados

A primeira etapa no desenvolvimento do simulador foi a leitura dos arquivos. O arquivo binário contém as instruções codificadas que o simulador deve interpretar e executar. Em Assembly, foram implementadas rotinas para abrir o arquivo e ler seu conteúdo, armazenando os dados binários em uma área de memória designada. Esse processo é crucial, pois fornece a base para todas as operações subsequentes do simulador.

O arquivo de dados contém as *strings* inicializadas no segmento "*.data*", e é essencial para imprimir os resultados da execução do programa, de forma que garanta o correto comportamento do mesmo.

3.2 Implementação da memória virtual

Com os dados armazenados, a próxima etapa foi a implementação de uma memória virtual. Esta memória simula a memória física do processador, permitindo o armazenamento e a manipulação dos dados necessários para a execução das instruções. A memória virtual foi estruturada para incluir áreas específicas para registradores, valores intermediários e resultados finais.

Além da memória virtual principal, foi implementada uma memória exclusiva para o segmento "*.data"*. Este segmento é comumente utilizado em programas Assembly para armazenar dados inicializados, como variáveis globais e constantes. A memória "*.data"* virtual implementada no simulador permitiu o armazenamento e o acesso a esses dados de maneira organizada e eficiente.

3.3 Decodificação de Instruções

A decodificação das instruções é um passo crítico no funcionamento do simulador. Cada instrução MIPS é composta por vários campos, incluindo o opcode, registradores, valores imediatos, *functs* e *shamt*. Para identificar e separar esses campos, foram utilizadas máscaras de bits. Primeiramente, uma máscara de bits foi aplicada para extrair os 6 primeiros bits da instrução, que representam o opcode. O opcode determina o tipo de instrução (R, I ou J) e guia a execução subsequente.

Após identificar o opcode, novas máscaras de bits foram aplicadas para extrair os componentes específicos das instruções do tipo R, como os registradores fonte (rs e rt), o registrador de destino (rd), o *shift amount* (*shamt*) e a função (*funct*). Cada um desses componentes é crucial para a execução correta das instruções.

3.4 Manipulação de Registradores

Para muitas instruções do tipo R e I, é necessário manipular e armazenar valores em registradores específicos. Isso envolve identificar os registradores corretos e calcular os deslocamentos necessários para acessar suas posições na memória virtual.

3.5 Execução das Instruções

Com os componentes das instruções devidamente extraídos e os registradores identificados, a próxima etapa é a execução das instruções. Esta fase envolve realizar as operações especificadas pelas instruções, que podem incluir operações aritméticas, lógicas, de desvio e de memória. Os resultados dessas operações são então armazenados na memória virtual ou nos registradores, conforme necessário.

4. Experimento

Para validar o funcionamento do simulador de instruções MIPS desenvolvido, um experimento utilizando o editor MARS. O MARS é uma ferramenta amplamente utilizada para simular programas escritos em Assembly MIPS, permitindo a visualização e depuração detalhada das instruções e da memória.

O experimento consistiu em utilizar o arquivo em binário fornecido pelo professor, que contém as instruções necessárias para calcular corretamente o fatorial do número 5. Após

compilar o código e executá-lo, comparou-se a saída do terminal com a saída do terminal do código do arquivo “trabalh0_01-2024_1.asm”, também fornecido pelo professor. Este arquivo contém a implementação correta do cálculo do fatorial em Assembly MIPS, servindo como referência para a verificação dos resultados.

5. Resultados e discussão

A implementação do simulador de instruções MIPS utilizando o editor MARS demonstrou resultados satisfatórios e proporcionou uma compreensão profunda dos conceitos de programação em Assembly e da arquitetura MIPS. Durante o desenvolvimento e execução do experimento, diversos aspectos importantes foram observados e analisados, permitindo uma discussão detalhada sobre o processo e os resultados obtidos.

Ao executar o simulador com o arquivo binário fornecido, observou-se que todas as instruções foram corretamente decodificadas e executadas. O simulador foi capaz de identificar os opcodes, registradores e outros componentes das instruções, realizando as operações necessárias conforme especificado. O cálculo do fatorial do número 5, que foi utilizado como caso de teste, foi concluído com sucesso, apresentando o valor correto de 120. No terminal, a combinação de *strings* do “.data” com o resultado da operação do fatorial foi corretamente impressa. A comparação da saída do terminal do simulador com a saída do terminal do código de referência fornecido pelo professor confirmou a precisão do simulador.

Com isso, o experimento proporcionou *insights* valiosos sobre a importância da organização e do planejamento no desenvolvimento de simuladores de instruções. A implementação de uma memória virtual detalhada e de uma memória “*.data*” virtual mostrou-se crucial para a simulação precisa das operações do processador MIPS. O sucesso na decodificação e execução das instruções binárias reforçou a compreensão dos conceitos fundamentais de arquitetura de computadores e programação de baixo nível.

6. Conclusões e perspectivas

Em primeira instância, o desenvolvimento e a validação do simulador de instruções MIPS utilizando o editor MARS foram bem-sucedidos, resultando em uma ferramenta funcional e precisa para a execução de programas em Assembly. Durante o experimento, o simulador demonstrou sua capacidade de decodificar e executar corretamente as instruções binárias, com uma precisão confirmada pela comparação com o código de referência fornecido pelo professor.

O uso do MARS foi crucial para o sucesso do projeto, permitindo a depuração detalhada e a visualização em tempo real das operações do simulador. As ferramentas de depuração e visualização de memória do MARS ajudaram a monitorar o comportamento do simulador, garantindo a identificação e correção de erros de forma eficaz. A implementação de uma memória virtual detalhada e de uma memória “*.data*” virtual mostrou-se essencial para a simulação precisa das operações do processador MIPS.

Com base nos resultados obtidos, várias perspectivas podem ser exploradas para aprimorar e expandir o projeto. Uma direção possível é a implementação de suporte para um conjunto mais

amplo de instruções MIPS, incluindo operações mais complexas e instruções de controle de fluxo mais avançadas. Isso permitiria ao simulador lidar com programas mais complexos e diversificados.

Além disso, a expansão do simulador para suportar outras arquiteturas de processadores poderia tornar a ferramenta ainda mais útil e versátil, permitindo a comparação e o estudo de diferentes arquiteturas de computação.

Em conclusão, o simulador de instruções MIPS desenvolvido mostrou-se uma ferramenta eficaz e precisa, com um potencial significativo para aprimoramentos e expansões futuras. As perspectivas exploradas sugerem diversas oportunidades para o desenvolvimento contínuo e a aplicação da ferramenta em contextos educacionais e de pesquisa, contribuindo para uma compreensão aprofundada da arquitetura de computadores e da programação de baixo nível.