Documentação do Projeto PyMips

Guilherme Ricardo Beira

11 de dezembro de 2023

Apresentação do Projeto

O PyMips é um simulador de processador MIPS desenvolvido como parte do projeto da disciplina de Arquitetura e Organização de Computadores. Este projeto visa proporcionar uma experiência prática e interativa para os estudantes explorarem a arquitetura MIPS, uma arquitetura de conjunto de instruções (ISA) amplamente utilizada em cursos de arquitetura de computadores.

Motivação

Inserido no contexto acadêmico, o PyMips foi concebido para atender às demandas de aprendizado da arquitetura MIPS. Ele oferece aos usuários a oportunidade de aprofundar seus conhecimentos em linguagem assembly e arquitetura de computadores, proporcionando uma plataforma de simulação onde podem carregar, executar e analisar o código assembly MIPS passo a passo.

Objetivos

O principal objetivo deste projeto é desenvolver um simulador robusto que permita aos usuários:

- Carregar código assembly MIPS.
- Executar o código de maneira passo a passo.
- Visualizar os efeitos de cada instrução nos registradores, memória e outros elementos do processador MIPS.

- Exercitar e consolidar conhecimentos adquiridos sobre linguagem assembly.
- Reforçar o entendimento sobre decisões de projeto e funcionamento de hardware.
- Praticar conceitos de integração software e hardware por meio de simulação.

Ao seguir esta documentação, você será guiado através das diversas funcionalidades do PyMips, desde a instalação até a execução de seus primeiros programas MIPS. Vamos começar explorando os recursos e benefícios que o PyMips oferece para o aprendizado prático da arquitetura MIPS.

1 Arquitetura

A arquitetura do PyMips é projetada para oferecer uma simulação completa e interativa do processador MIPS. Os principais componentes do projeto incluem:

1.1 Interface Gráfica do Usuário (GUI)

A GUI é responsável por fornecer uma interface amigável para os usuários interagirem com o simulador. Ela exibe informações cruciais, como registradores, memória e o código MIPS em execução. A GUI também permite a interação do usuário para carregar programas, executar passo a passo e visualizar os resultados.

1.2 Tradutor de Instruções

O Tradutor de Instruções converte o código assembly MIPS carregado pelo usuário em instruções binárias compreendidas pelo simulador. Ele utiliza tabelas de mapeamento para traduzir cada instrução, considerando os registradores envolvidos e os formatos específicos.

1.3 Simulador MIPS

O Simulador MIPS é o núcleo do PyMips. Ele executa as instruções binárias traduzidas pelo Tradutor de Instruções, atualiza o estado dos registradores e da memória, e fornece informações à GUI para exibição. O simulador também gerencia as operações de controle de fluxo, como desvios e chamadas de sistema.

1.4 Memória

A Memória armazena tanto o código quanto os dados utilizados durante a execução do programa MIPS. Ela é organizada em seções para instruções e dados, permitindo o acesso eficiente durante a execução.

1.5 Registradores

Os Registradores representam o estado interno do processador MIPS. O simulador mantém uma tabela de registradores que é atualizada durante a execução das instruções. Os registradores incluem os registradores de propósito geral, o contador de programa (PC) e outros registradores especiais.

1.6 Console de Saída

O Console de Saída exibe mensagens de saída, como resultados de operações, erros e mensagens de depuração. Ele fornece uma maneira adicional de comunicação com o usuário, além da GUI.

1.7 Controlador Principal

O Controlador Principal coordena a interação entre os diferentes componentes do PyMips. Ele inicia o simulador, gerencia a comunicação entre a GUI e o simulador, e coordena eventos importantes, como carregamento de programas e interrupções de usuário.

1.8 Fluxo de Interconexão

O fluxo de interconexão entre esses componentes é essencial para garantir uma simulação suave. A GUI interage com o Tradutor de Instruções para carregar programas, o Tradutor de Instruções envia as instruções traduzidas ao Simulador MIPS, e o Simulador MIPS atualiza o estado, que é refletido na GUI.

Essa arquitetura modular facilita a manutenção, expansão e compreensão do PyMips, tornando-o uma ferramenta valiosa para o aprendizado e experimentação com a arquitetura MIPS.

2 Algoritmos

O PyMips implementa diversos algoritmos para simular a execução de código assembly MIPS, proporcionando uma experiência educacional rica. Abaixo,

detalhamos alguns dos principais algoritmos empregados:

2.1 Tradução de Instruções

A tradução de instruções é um processo essencial no PyMips, onde o código assembly MIPS é convertido em representações binárias compreensíveis pelo processador. Os algoritmos de tradução variam de acordo com o tipo de instrução (R, I, J) e incluem lógica para lidar com diferentes operandos.

2.2 Manipulação de Registradores e Memória

Os algoritmos de manipulação de registradores e memória são responsáveis por atualizar e acessar os valores armazenados nos registradores e na memória do processador. Essas operações são fundamentais para simular o estado dinâmico do processador durante a execução do código.

2.3 Controle de Desvio

Para instruções de controle de fluxo, como branches e jumps, o PyMips utiliza algoritmos de controle de desvio. Esses algoritmos determinam o próximo endereço de execução com base nas condições especificadas pelas instruções de desvio.

Esses são apenas alguns exemplos dos algoritmos implementados no PyMips para fornecer uma simulação eficiente e educativa da arquitetura MIPS.

3 Estruturas de Dados

O PyMips utiliza diversas estruturas de dados para organizar e manipular informações durante a execução do simulador. A seguir, detalhamos algumas das principais estruturas de dados empregadas:

3.1 Classes e Objetos

O PyMips é organizado em um conjunto de classes que representam entidades essenciais no simulador. Algumas dessas classes incluem:

- `Registrador`: Representa um registrador do processador MIPS e contém métodos para manipulação.
- `Memoria`: Modela a memória do processador e oferece métodos para leitura e escrita.

- `Console`: Essa classe foi criada para associar funções ao console do programa de forma mais simples e eficaz.

3.2 Tokens e Análise Lexical

Na análise lexical, o PyMips utiliza tokens para representar unidades léxicas no código assembly MIPS. Cada token contém informações sobre o tipo, o valor e a linha associada. A análise lexical do código fonte é realizada por meio de estruturas de dados que lidam com a identificação e categorização dos tokens.

Essas estruturas de dados são cruciais para garantir uma análise eficiente do código fonte e para preparar as informações necessárias para a tradução e execução subsequente.

3.3 Outras Estruturas Internas

Além das estruturas mencionadas, o PyMips pode fazer uso de outras estruturas internas para otimizar operações específicas ou armazenar informações temporárias durante a execução.

3.4 Validação e Interpretação do Código MIPS

A função `validar_codigo` desempenha um papel crucial no processo de interpretação e validação do código assembly MIPS no simulador PyMips. Abaixo, apresentamos um resumo das principais funcionalidades e lógica da função.

3.4.1 Resumo da Função `validar_codigo`

A função `validar_codigo` é responsável por validar e interpretar um conjunto de tokens que representam um programa assembly MIPS. O código percorre os tokens agrupados por linha, realizando as seguintes verificações:

1. **Registradores:** Verifica a validade dos registradores utilizados no código, destacando qualquer registrador inválido.

2. **Diretivas Data e Text:** Assegura a presença das diretivas `.data` e `.text` no código, gerando mensagens de aviso se ausentes.

3. **Instruções:** Avalia a validade das instruções MIPS, identificando instruções inválidas e gerando mensagens informativas.

4. **Labels:** Registra a presença de labels no código, associando-as aos endereços correspondentes.

5. **Strings:** Gerencia declarações de strings, escrevendo-as na memória e atualizando os registradores apropriados.

6. **Execução Passo a Passo:** Chama a função `executa_linha_linha` para executar o código linha por linha.

A função visa fornecer uma validação abrangente do código MIPS, identificando potenciais erros e preparando o ambiente para a execução passo a passo no simulador PyMips.

3.4.2 Uso

Esta função desempenha um papel crucial no processo de preparação e validação do código MIPS antes da execução no simulador. Seu objetivo é garantir que o código esteja estruturalmente correto e em conformidade com as regras da arquitetura MIPS, promovendo uma execução suave e precisa no ambiente simulado.

3.5 Execução Passo a Passo do Código MIPS

A função `executa_linha_linha` desempenha um papel fundamental na execução passo a passo do código MIPS no simulador PyMips. Abaixo, apresentamos um resumo das principais funcionalidades e lógica da função.

3.5.1 Resumo da Função `executa_linha_linha`

A função `executa_linha_linha` é responsável por executar o código MIPS, linha por linha, no simulador PyMips. Aqui estão as principais verificações e ações realizadas durante a execução:

1. **Tipos de Instruções:** A função categoriza as instruções em tipos, como lógico-aritméticas, de controle e de acesso a dados.

2. **Atualização do Contador de Programa (PC):** O contador de programa é incrementado em 4, refletindo o avanço para a próxima instrução.

3. **Tradução para Binário:** A instrução é traduzida para sua representação binária e exibida na interface gráfica de tradução.

4. **Execução de Instruções:** A função chama módulos específicos para executar instruções lógico-aritméticas, de controle ou de acesso a dados, dependendo do tipo identificado.

5. **Atualização da GUI:** Os registradores são atualizados na interface gráfica após a execução da instrução.

6. **Syscall:** Se a instrução atual for uma syscall, a função `syscall` é chamada para tratar chamadas de sistema.

A função `executa_linha_linha` desempenha um papel crucial na simulação da execução de código MIPS, fornecendo uma visão detalhada das operações realizadas a cada passo.

3.5.2 Uso na Documentação

Este trecho de código é parte integrante do mecanismo de execução do simulador PyMips. Sua implementação é essencial para garantir uma execução precisa e passo a passo do código MIPS carregado no ambiente do simulador. O entendimento desta função é crucial para usuários que desejam explorar e compreender o comportamento interno do simulador.

4 Execução de Instruções

O processo de execução de instruções no PyMips é coordenado pelo módulo `execution` e envolve a interpretação e execução de diferentes tipos de instruções.

4.1 Instruções Lógico-Aritméticas

A execução das instruções lógico-aritméticas é tratada pela função `instrucao_logicoaritmetica`. Esta função recebe os tokens da linha que contêm a instrução e os registradores relevantes, realizando a operação especificada pela instrução.

Listing 1: Função `instrucao_logicoaritmetica`

```
import tkinter.simpledialog

def instrucao_logicoaritmetica(tokens_da_linha, registradores_widget):
    ...
    try:
        imediato = int(registrador_operando2)
        if instrucao == "addi":
```

```
            registradores_widget.registradores[registrador_destino] =
        ...
    except:
        if instrucao == "add":
            registradores_widget.registradores[registrador_destino] =
        ...
```

4.2 Instruções de Desvio

As instruções de desvio são processadas pela função `instrucao_desvio`. Esta função determina o próximo índice de linha a ser executado com base no tipo de instrução de desvio.

Listing 2: Função `instrucao_desvio`

```
def instrucao_desvio(tokens_da_linha, registradores_widget, linha_atual
    ...
    if tipo == "j":
        return registradores_widget.endereco_label[destino_label]
    elif tipo == "jal":
        ...
    elif tipo == "jr":
        return registradores_widget.registradores["ra"]
    elif tipo in {"beq", "bne", "bgt", "bge", "blt"}:
        ...
    return linha_atual_index
```

4.3 Instruções de Acesso à Memória

A execução das instruções de acesso à memória é gerenciada pela função `instrucao_acesso`, que manipula operações como carga e armazenamento na memória.

Listing 3: Função `instrucao_acesso`

```
def instrucao_acesso(tokens_da_linha, registradores_widget, memoria):
    ...
    if instrucao == "li":
        ...
    elif instrucao == "lw":
        ...
    elif instrucao == "sw":
```

```
    ...
elif instrucao == "la":
    ...
elif instrucao == "move":
    ...
```

4.4 Syscalls

O tratamento de syscalls é implementado na função `syscall`, que executa a operação apropriada com base no valor do registrador `v0`.

Listing 4: Função `syscall`

```
def syscall(registradores_widget, console, memoria):
    ...
    if registradores_widget.registradores['v0'] == 1:
        console.write(str(registradores_widget.registradores['a0']) +
    elif registradores_widget.registradores['v0'] == 4:
        ...
    elif registradores_widget.registradores['v0 '] == ...
```

5 Guia de Uso

O PyMips oferece uma plataforma interativa para explorar a arquitetura MIPS e praticar conceitos de linguagem assembly. Siga as etapas abaixo para começar a usar a ferramenta.

5.1 Requisitos

Antes de começar, certifique-se de ter atendido aos seguintes requisitos:

- Python instalado (versão 3.11.X ou superior).
- Bibliotecas necessárias (listadas no arquivo de requisitos).

5.2 Instalação

Para instalar o PyMips, siga estas etapas:

1. Clone o repositório do PyMips do GitHub:
   `git clone https://github.com/GuiRBeira/PyMips`

2. Navegue até o diretório do PyMips:
   `cd pymips`

3. Instale as dependências:
   `pip install -r requirements.txt`

5.3 Executando o PyMips

Para iniciar o PyMips, execute o seguinte comando:

```
python pymips.py
```

Isso abrirá a interface gráfica do PyMips.

5.4 Carregando Código MIPS

Para carregar um programa MIPS, siga estas etapas:

1. Clique no botão "Open"na interface.

2. Selecione o arquivo de código assembly MIPS que deseja executar.

3. O código será carregado na memória do simulador.

5.5 Visualização de Resultados

Ao final da execução, examine os resultados na interface gráfica. Isso inclui o estado final dos registradores e qualquer saída gerada pelo programa.