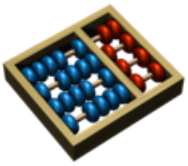
Universidade Estadual de Campinas
Instituto de Computação
MO601 – Arquitetura de Computadores II
Prof. Rodolfo Jardim de Azevedo



Projeto 2

Um simulador simples do processador RISC-V

Rubens de Castro Pereira
RA: 217146

Campinas - SP
3 de maio de 2023

Índice

1 Introdução **2**

2 Descrição geral **2**

3 Ambiente de Desenvolvimento **2**

4 Algoritmo de Simulação **3**

5 Testes **3**

6 Considerações sobre o aprendizado nesse projeto **4**

1 Introdução

Este relatório apresenta o projeto 2 da disciplina Arquitetura de Computadores II (MO601) ministrada pelo Prof. Rodoldo Jardim de Azevedo, cujo propósito é implementar um simulador básico do processador RISC-V RV32IM considerando o conjunto das instruçãos básicas de inteiros de 32 bits e, adicionalmente, as instruções de multiplicação e divisão inteiras de 32 bits. A seção 2 apresenta a descrição geral do projeto, a seção 3 relaciona as tecnologias utilizadas no ambiente de desenvolvimento, a seção 4 apresenta o algoritmo de simulação do processador, a seção 5 detalha como os testes foram realizados e a seção 6 apresenta as considerações sobre o aprendizado neste projeto.

2 Descrição geral

O objetivo deste projeto é implementar um simulador do processador RISC-V RV32IM que executa o conjunto das instruçãos básicas de inteiros, as instruções de multiplicação e de divisão inteiras, em uma arquitetura de 32 bits.

O projeto utiliza como entrada os programas codificados na linguagem C disponibilizados no repositório ACStone (*https://github.com/rjazevedo/ACStone*), que possuem funcionalidades e utilidades reduzidas, contudo são excelentes artefatos para uso em simuladores de processadores. Nesse repositório foram disponibilizados 77 programas fonte C de testes os quais contemplam todas as instruções implementadas neste simulador. O conjunto de instruções RISC-V é composto de 47 instruções básicas (*RV32I Base Instruction Set*) e 8 instruções de multplicaçao e divisão (*RV32M Standard Extension*), totalizando 55 instruções.

Os programas fonte foram compilados utilizando-se a arquitetura 32 bits, em seguida montados em formato assembler (*linked*) e utilizados como entrada no simulador do processador RISC-V.

O simulador implementa um modelo de processador contendo as estruturas de dados básicas com destaque à memória RAM com tamanho suficiente para acomodar as instruções e estuturas de dados, 32 registradores e contador de programa (PC - *Program Counter*). Cada programa assembler foi processado separadamente produzindo um arquivo de saída com o log da execução conforme formato especificado na definição desse projeto (padrão ABI). Cada arquivo de log é a evidencia da simulação de um programa fonte/assembler.

3 Ambiente de Desenvolvimento

O ambiente de desenvolvimento do simulador está estruturado com as seguintes tecnologias:

- Sistema operacional: Windows 10
- Ambiente integrado de Desenvolvimento (IDE): Visual Studio Code (VS Code) version 1.76.2
- Linguagem de programação: Python versão 3.9.5
- Pacotes Python: OS

- Ferramentas para compilação e montagem dos programas fonte C - *Embecosm - Tool Chain Downloads*
  - Web site: https://www.embecosm.com/resources/tool-chain-downloads/
  - Pacote selecionado para instalação do compilador RISC-V 32 bits:
    * Windows 10 - 32-bits GCC (.zip)
    * https://buildbot.embecosm.com/job/riscv32-gcc-win64/169/artifact/riscv32-embecosm-gcc-win64-20230402.zip

4 Algoritmo de Simulação

O algoritmo do simulador do processador RISC-V RV32IM é apresentado em *algorithm 1*.

Algorithm 1: Algoritmo do simulador do processador RISC-V RV32IM.

Entrada: *programa assembler (exemplo: 000.main.asm)*

Saída: *arquivo de saída (log) do programa simulado (exemplo: 000.main.log)*

```
1  ExecutarPrograma(programa assembler, PC) begin
2      while não alcançar a instrução ebreak do
3          ler instrução indicada pelo PC
4          decodificar instrução conforme o tipo específico (R/I/S/B/U/J)
5          executar a instrução de acordo com sua implementação
6          gerar linha de log de saída da instrução executada
7      end
8  end
9  /* Programa principal que gerencia a simulação do processador RISC-V
     RV32IM na execução de programas.  */
10 foreach programa assembler do
11     inicializar processador RISC-V criando memória RAM, registradores e PC
12     ler programa assembler
13     armazenar programa lido na memória RAM
14     inicializar contador do programa (PC) com o endereço de memória da primeira
         instrução que será executada
15     ExecutarPrograma(programa assembler, PC)
16     gravar arquivo com o log das instruções executadas
17 end
```

5 Testes

A execução dos testes compreende duas etapas: 1) geração dos programas em formato assembler (*linked*); e 2) simulação da execução dos programas assembler de acordo com o processador RISC-V

RV32IM.

A etapa 1 é realizada a partir dos programas previamente compilados compilados (extensão ”riscv”) e fornecidos pelo professor, os quais estão localizados em *https://drive.google.com/drive/u/2/folders /1ei8E-qk2dwQvCWJv8ovGJiIdLjw7yVfP*. Devido a isso, a etapa de compilação já foi sanada previamente bastando criar os programas assembler. Para isso, uUm arquivo em lote *dump-programas-compilados.bat* deve ser executado para que o programa utilitário *riscv32-unknown-elf-objdump.exe* crie os programas em assembler (extensão ”asm”) a partir dos programas compilados (extensão ”riscv”).

A etapa 2 realiza a simulação do processador RISC-V executando cada programa assembler e gerando um arquivo de log (extensão ”log”) para cada programa simulado, a fim de evidenciar a sua execução.

Os arquivos foram organizados em uma pasta de testes específica denominada *test* e localizada dentro da pasta do projeto (*RCP-MO601-Project-02*). A pasta *test* contém todos os programas fonte em C (extensão ”c”), os programas compilados previamente fornecidos pelo professor e os programas assembler gerados na etapa 1. Para facilitar a visualização do resultado do simulador, foi criada a pasta *log* dentro da pasta *test* a fim de acomodar os arquivos de saída da simulação.

A execução de simulação dos programas assembler apresenta no terminal ou console a identificação do programa e a contagem total dos ciclos de execução das instruções.

6 Considerações sobre o aprendizado nesse projeto

Neste projeto foi possível compreender de maneira detalhada como os processadores executam os programas desenvolvidos em linguagem de alto nível a partir das instruções geradas pelo compilador. Quando se cria software e sistemas de informação, essa camada do software localizada muito próxima do hardware se torna completamente oculta aos desenvolvedores e é exatamente nesse contexto que o simulador do processador atua, avaliando (decodificando) e executando cada instrução oriunda de processos anteriores de compilação e montagem (*linked*).

Conhecer em detalhes uma Arquitetura do Conjunto de Instruções (*Instruction Set Architecture - ISA*) RISC-V foi bem interessante, demonstrando o quão simples pode ser uma instrução executada por um processador e ao mesmo tempo poderosa dando suporte às linguagens de programação de alto nível.

As convenções utilizadas na especificação das instruções RISC-V foi provavelmente o maior desafio encontrado no desenvolvimento deste projeto. A documentação da implementação de determinadas instruções presume conhecimentos prévios e, por isso, a dificuldade no entendimento, tendo como resultado algumas simulações de programas que não ocorreram.

Por fim, destaco que os resultados alcançados nesse projeto mesmo parciais abriram novas frentes de conhecimento, mas continuo com o propósito de concluir na totalidade a simulação de todos os programas testes.

Referências

Andrew Waterman1, and Krste Asanovi, The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2, 2017. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf.

Msyksphinz. RISC-V Instruction Set Specifications, 2019. Acessado em abril de 2023. Disponível em https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html.

RISC-V Instruction Encoder/Decoder. Acessado em abril de 2023. Disponível em https://luplab.gitlab.io/rvcodecjs