RISC-V: A Didactic Platform
Report

Jules PERRIN, Professor: Theo KLUTER

June 12, 2023

Contents

1 Introduction

1.1 Motivation

This project, RISC-V: A Didactic Platform, embarks on the pursuit of simplifying complex computer architecture concepts, targeting an academic audience who is striving to comprehend the intricate world of processor design. It specifically focuses on the creation of a RV32I processor, based on the RISC-V instruction set architecture.

RISC-V, an open standard instruction set architecture (ISA), has increasingly gained popularity due to its simplified design and flexibility for customization. While its usage has been witnessed across a multitude of applications, its potential as an educational tool remains largely unexplored. The objective of this project is to bridge this gap by providing a detailed, comprehensible, and implementable design of an RV32IM processor, leveraging the RISC-V architecture.

This project not only contributes to the existing body of knowledge around RISC-V and processor design but also aims to democratize access to information on complex computing architectures. Through this platform, users will not only learn the theory behind processor design but will also gain valuable hands-on experience with the actual implementation, enabling them to better understand the interplay between hardware and software in a computing system.

In an era where the understanding of computer architecture is pivotal for both hardware and software development, a project like RISC-V: A Didactic Platform can serve as a crucial resource. By harnessing the power of the RISC-V architecture and presenting it through a user-friendly and accessible platform, we hope to elevate the understanding of processor design to new heights.

1.2 Project Scope

Here are the different main focus points of the project:

- Openess: It is an important point for the project to use as much as possible open source software and also for the project to be as open as possible such that people can work on it with the less possible restrictions.
- Comprehensive: Since the project needs to be open source and be used by academic people to understand but also add things on top of it, the project needs to focus on being as comprehensive as possible.
- Simplicity: This point is linked a bit with the last one, but the project doesn't aim as performance first, but more as ease of understanding, so if a solution can be easier but degrade a bit the performance, it should be taken except if it adds an educational value to do it

1.3 Project Objectives

Overall, the project aims to give a very basic implementation of an RV32IM processor such that it can be used as an academic tool to teach and explain how a RISC processor works and how to build one using HDL languages such as Verilog. Here are the main objectives that has been set for the project:

- First create an unpipelined version of the RV32IM processor as a proof of concept
- Pipeline the existing unpipelined version of the processor
- Fix the different issues caused by the pipelining of the processor, mainly the data dependency but also the branch prediction problem by either stall or implementing some forwarding paths and branch predictor and mechanisms to make everything work correctly

- The simulation should works on Icarus Verilog to make the whole process more open instead of relying on proprietary software such as Questa Modelsim.
- Add extensive testing of the different main modules to make the processor as reliable as possible.
- Create a CI pipeline such that it runs automatically the different tests and checks at each commit to make sure that the code is always working.
- Add an easy way for users to create programs and load them into the ROM memory.
- Document everything such that users can easily understand how things work and can work on it easily.

2 Background Research and Knowledge

2.1 RISC-V

RISC-V is an open-source instruction set architecture (ISA) based on the reduced instruction set computer (RISC) principles. It is a free and open ISA enabling a new era of processor innovation through open standard collaboration.

Most of the research I've done on it to understand how it works and how to implement it has been done on the official website of the RISC-V foundation [1, p. 9-25] and of course more precisely the chapter about the RV32I base integer instruction set. Inside it describes how the different instructions are encoded with the 6 different formats [1, p. 104-105] that are used to encode the different instructions but also how the different instructions should behave using a mix of the manual but also a reference card [2, p. 81-83].

2.2 Architecture

2.2.1 Unpipeline

The unpipelined version of the CPU is mostly inspired by the one we did during the Comparch 1-2 courses at EPFL since NiosII and RISC-V are very similar. The main difference is that the RISC-V is a 32 bits architecture while NiosII is a 16 bits architecture and also that RISC-V has a lot more instructions than NiosII with small differences in how they work.

2.2.2 Pipeline

The pipelined version is inspired also mostly by what we did during the Comparch 2 course at EPFL but since we didn't implement the branch predictor and the forwarding paths, I had to imagine and think about most of the implementation myself. I've only done a couple of research to see if I could find some examples of such implementation mostly about forwarding paths since I wasn't sure how to implement it correctly at first glance. I've used mainly this pdf [3] to reassure myself that I was doing it correctly. The branch predictor part has an algorithm I haven't invented called a two-level adaptive branch predictor [4]. It uses a 2 bits saturating counter but also knows patterns such that the predictions apply locally to the branch. It is a very simple algorithm but it works well enough for this project.

2.3 Verilog

Not knowing anything about Verilog since at EPFL we used VHDL instead, I had to learn it from scratch. For that, I've used a website given by the professor [5] that explains the basics of Verilog and how to use it. From here I've started creating small modules and searching for examples on the internet to see how other people were doing it and also when needed I've used different forums to find answers to my questions and when needed consulted the official Verilog documentation [6].

2.4 Simulation Tools

During the project, I mainly used a close source simulator that we were using at EPFL and developed by Intel called Questa Modelsim. It is a great tool combining compiling and seeing the waveforms of the signals in the same window. It is also very easy to use and has a lot of features that I haven't used during the project. The only downside is that it is not free and it is quite buggy sometimes and you need to activate an option called `full visibilty` to have access to all the signals in the waveform window. At the end of the project, since I had to implement the CI, I needed a free simulator with a command line interface and I've used Icarus Verilog. It is a free simulator that is not too hard to use and has a command-line interface. The only downsides I've found while using it are that there is not a lot of documentation [7] and it is a bit less user-friendly than Questa Modelsim. You have to use a different tool to see the waveforms like GTKWave and you don't have access to the array of signals easily like in Questa Modelsim. Also, one weird issue I've encountered is that the simulation gives different results than Questa Modelsim because the full visibility option does things differently than Icarus Verilog, but I've luckily found the issue even though to this day I don't know why it was doing that.

2.5 RISC-V Toolchain

2.5.1 Developping using Assembly Code

It is not that easy to find a way to compile a RISC-V RV32I assembly code into a hexadecimal file that can be read by the ROM module. So for that, I've found two projects that helped me develop simple programs for testing purposes. The first one is a Python script that converts pseudo instructions into hexadecimal instructions [8]. The second one is a website that converts assembly code into hexadecimal instructions [9]. One last tool I've used to reverse the hexadecimal instructions into assembly code is a website [10] but also used it to modify one instruction at a time to see what it does.

2.5.2 RISC-V GNU Compiler Toolchain

The RISC-V GNU Compiler Toolchain is a set of tools that allows you to compile C/C++ code into RISC-V assembly code. It is composed of a compiler, an assembler, a linker and a debugger. To use it you need to compile it from the source code which was not that easy to do because the documentation for such a small architecture set is not that great and since we're developing bare metal programs, it is even harder to find information about it. So I've used the official documentation [11] and also a tutorial I've found on the internet [12] to compile it and finally use it.

3 Design

References

[1] Andrew Waterman and Krste Asanović. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*. 2017. URL: `https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf` (visited on 06/11/2023).

[2] John Winans. *RISC-V Assembly Language Programming*. 2022. URL: `https://github.com/johnwinans/rvalp/releases/download/v0.18.1/rvalp.pdf` (visited on 06/11/2023).

[3] University of Washington. *Forwarding*. URL: `https://courses.cs.washington.edu/courses/cse378/07au/lectures/L12-Forwarding.pdf` (visited on 06/11/2023).

[4] Tse-Yu Yeh and Yale N. Patt. *Two-Level Adaptive Training Branch Prediction*. 1991. URL: `https://www.inf.pucrs.br/~calazans/graduate/SDAC/saltos.pdf` (visited on 06/11/2023).

[5] Chipverify. *Verilog Tutorial*. URL: `https://www.chipverify.com/verilog/verilog-tutorial` (visited on 06/11/2023).

[6] IEEE Computer Society. *IEEE Standard for Verilog Hardware Description Language*. 2005. URL: `http://staff.ustc.edu.cn/~songch/download/IEEE.1364-2005.pdf` (visited on 06/11/2023).

[7] Stephen Williams. *Getting Started With Icarus Verilog*. 2023. URL: `https://steveicarus.github.io/iverilog/usage/getting_started.html` (visited on 06/11/2023).

[8] Don Dennis. *RISCV-RV32I-Assembler*. Apr. 15, 2019. URL: `https://github.com/metastableB/RISCV-RV32I-Assembler` (visited on 06/11/2023).

[9] Lucas Teske. *RISC-V Online Assembler*. URL: `https://riscvasm.lucasteske.dev/#` (visited on 06/11/2023).

[10] Davis LupLab @ University of California. *RISC-V Instruction Encoder/Decoder*. 2023. URL: `https://luplab.gitlab.io/rvcodecjs/` (visited on 06/11/2023).

[11] RISC-V Foundation. *RISC-V GNU Compiler Toolchain*. URL: `https://github.com/riscv-collab/riscv-gnu-toolchain`.

[12] Vivonomicon. *Bare-metal RISC-V Development with the GD32VF103CB*. Feb. 11, 2019. URL: `https://vivonomicon.com/2020/02/11/bare-metal-risc-v-development-with-the-gd32vf103cb/`.