Bachelor Thesis

Benchmark of RISC-V in BTOR2

Jan Krister Möller

Examiner: Dr. Mathias Fleury

University of Freiburg

Faculty of Engineering

Department of Computer Science

Chair of Computer Architecture

August 18, 2025

Writing Period

24. 06. 2025 – 24. 09. 2025

Examiner

Dr. Mathias Fleury

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

foo bar [1] [2] [3]

Contents

List of Figures

List of Tables

List of Algorithms

1 Motivation

This is a template for an undergraduate or master's thesis. The first sections are concerned with the template itself. If this is your first thesis, consider reading.

2 RISC-V

2.1 Overview

RISC-V is an open source instruction set architecture first published in May 2011 by A. Waterman et al. [4]. As contained in the name, it is based on the RISC design philosophy. **(TODO: Explain RISK (compare wiki))** Since 2015 the development of RISC-V is coordinated by the RISC-V International Association, a non-profit corporation based in Switzerland since 2020 [5]. Its goals are among others an *open* ISA that is freely available to all, a *real* ISA suitable for native hardware implementation and an ISA separated into a *small* base integer ISA usable by itself e.g. for educational purpose and optional standard extensions to support general purpose software development [1](Chapter 1).

It currently contains four base ISAs, namely RV32I, RV64I, RV32E and RV64E, which may be extended with one or more of the 47 ratified extension ISAs [1] (Preface).

(EXTEND: Hier brauchts vlt noch was) **(TODO: little endian erwähnen?)**

For my work, I will focus on a subset of the RV64I ISA.

2.2 The RV64I ISA

RV64I is not complex, but its structure is relevant to understand my later work. So I explain all elements relevant for my thesis.

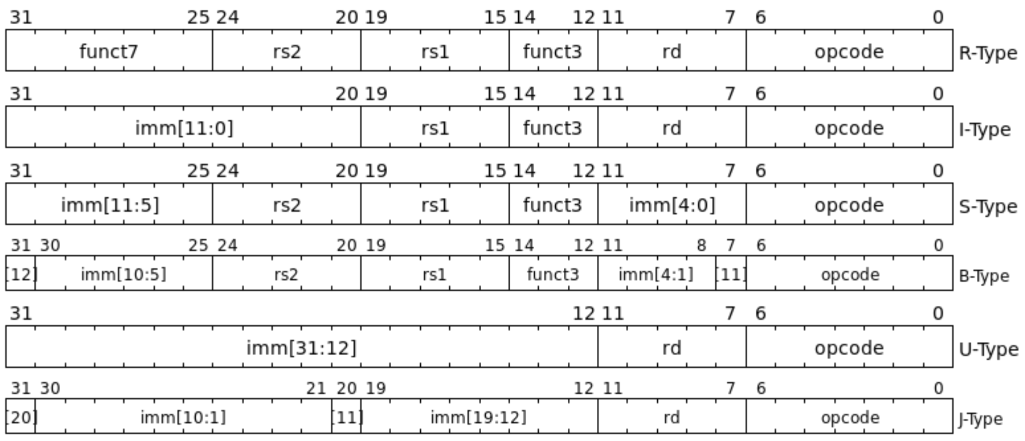

Figure 1: RV64I encoding formats, used in [1](Chapter 2.3) **(TODO: Kopie richtig angeben)**

RV64I has 32 64bit registers called $x0$-$x31$, where $x0$ is hardwired to 0 on all bits. Registers $x1$-$x31$ are general purpose and are interpreted by various instructions as a collection of booleans, two's complement signed binary integers or unsigned integers. Additionally, there is a register called *pc* acting as program counter and holding the address of the current instruction [1](Chapters 4.1, 2.1).

In RV64I, a memory address has the size of 64bit. As the memory model is defined to be single byte addressable, the address space of RV64I is 2^{64} bytes [1](Chapter 1.4).

Like almost all the standard ISAs of RISC-V, RV64I has a standard encoding length of 32bit or 1 *word*. Only the compressed extension C adds instructions with a length of 16bit [1](Chapter 1.5), but it is irrelevant for us. All instructions of RV64I are encoded in one of the six formats shown in Figure 1.

The design of these format results in the following features:

- As of RISC-Vs little endianess, the *opcode*, which encodes the general instruction, is always read first. Also, further specification of the instruction by $funct3$ and $funct7$ is found always at the same location.

- If used by the instruction, the destination register *rd* and the source registers *rs*1 & *rs*2 are always at the same place. This simplifies decoding.

- The highest bit of the immediate value *imm* is always bit 31. This makes finding the sign of a signed immediate value trivial.

Note that each immediate subfield is labeled with the bit position in the immediate value. Immediate values are always sign extended to 31bit and in case of U-, B- and J-type the missing lower bits are filled with zeros.

The instructions relevant to my work are listed in Table 1

INSTR	TYPE	INSTR	TYPE	INSTR	TYPE	INSTR	TYPE
`LUI`	U	`LW`	I	`XORI`	I	`SLT`	I
`AUIPC`	U	`LD`	I	`ORI`	I	`SLTU`	I
`JAL`	J	`LBU`	I	`ANDI`	I	`XOR`	I
`JALR`	I	`LHU`	I	`SLLI`	I	`OR`	I
`BEQ`	B	`LWU`	I	`SRLI`	I	`AND`	I
`BNE`	B	`SB`	S	`SRAI`	I	`SLL`	I
`BLT`	B	`SH`	S	`ADDIW`	I	`SRL`	I
`BGE`	B	`SW`	S	`SLLIW`	I	`SRA`	I
`BLTU`	B	`SD`	S	`SRLIW`	I	`ADDW`	I
`BGEU`	B	`ADDI`	I	`SRAIW`	I	`SLLW`	I
`LB`	I	`SLTI`	I	`ADD`	I	`SRLW`	I
`LH`	I	`SLTIU`	I	`SUB`	I	`SRAW`	I

Table 1: Subset of RV64I instructions

I splitted the instructions of Table 1 into nine groups based on their operations. `LUI` & `AUIPC` move a high immediate into *rd*, `JA*` instructions are unconditional jumps and `B*` instructions conditional jumps. `L*` are loading sign extended values from memory, either at `B`yte, `H`alfword, `W`ord or `D`oubleword length. In contrast, `S*` instructions write the defined length to memory. **(TODO: arithmetic)** Note that the suffix `U` defines an operation where the values are processed as unsigned.

```
REGISTERS:
PC: current pc in hex
x(0 - 31): value of register in hex

MEMORY:
(address in hex): byte, halfword, word or doubleword in hex
```

Figure 2: Construction of .state files

2.3 Simulation of RISC-V

(TODO: Vielleicht besser aufgehoben in Kap.4, aber statefile relevant.)

2.3.1 Saving the State of a RISC-V Processor

To save the current state of a RISC-V processor, the registers and the memory have to be stored. For this I devised the format as shown in Figure 2. The minimal file only consists of the two designators "REGISTERS:"& "MEMORY:", the current *pc* and one empty line.

3 BTOR2

3.1 Model Checking

3.2 The BTOR2 Language

3.3 The BTOR2 Witness

4 Transforming RISC-V to BTOR2

4.1 The Concept

4.2 Encoding

4.2.1 Constants

4.2.2 State Representation

4.2.3 Initialization

4.2.4 Computing values

Opcode

funct3 & funct7

Registers

Immediate

4.2.5 Command Detection

4.2.6 Next-State-Logic

4.2.7 Constraints

4.3 Testing for Correctness

4.3.1 State Fuzzer

4.3.2 Automated Logging

4.4 Functional vs Relational Next-State-Logic

5 Benchmarks

5.1 MultiAdd in Functional and Relational Next-State-Logic

5.2 Memory Operations

5.3 Results

Bibliography

[1] *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*, 2025, version 20250508. [Online]. Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications

[2] A. Niemetz, M. Preiner, C. Wolf, and A. Biere, "Btor2 , BtorMC and Boolector 3.0," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, pp. 587–595.

[3] F. Schrögendorfer, "Bounded Model Checking of Lockless Programs," Master's thesis, Johannes Kepler University Linz, August 2021. [Online]. Available: https://epub.jku.at/obvulihs/download/pdf/6579523

[4] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: Base user-level isa," UC Berkeley, Tech. Rep. UCB/EECS-2011-62, May 2011. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html

[5] "History of RISC-V," https://riscv.org/about/, accessed: 15.08.2025.