RISC-V Simulator Technical Specification

Version: 0.00 | 2/11/2022

Editors: Hector Soto
Created By: Hector Soto (sotohec@pdx.edu)
December 2/11/2022

Preface

This document describes the functionality of the RISC-V Simulator (RVS).

Table of Contents

Section 1 – Introduction

System Requirements: 64-bit OS with more than 4 GiBs of memory.
^ This program can possibly allocate more than 4 GiBs of memory.

This simulator will be coded in C++.

This simulator will simulate the (RISC-V) RV32I Base Integer Instruction Set as described in (https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf). It will have a **default max program size of 64 KiB or smaller**. Since each instruction is 32-bits, this means a default limit of 2048 instructions. With flags, this limit can be changed.

1.1 – Functional Description

The RISC-V Simulator (RVS) will accept input parameters such as a memory image file, starting address (defaulted to 0), and stack address (defaulted to 65535).

Section 2 – Code Specification

This section describes the code structure of the simulator.
It will describe some of the classes, functions, structures, and variables used in the simulation, including some of their relations to one another. A **Pseudo-Definition** is a non-final definition in C++ code or similar to represent the logic or structure of things such as functions, classes, etc. Do not expect a Pseudo-Definition to be a 100% accurate representation of implemented code.

Note: memory => user's memory. simulated-memory => memory affected by simulated instructions.

2.1 – Simulated Memory

The RISC-V Simulator (RVS) needs to simulate memory that instructions read and write to. Furthermore, the **size of the simulated memory will default to 4 GiB** as that's the limit of the simulated ISA (RV32I). Since a default maximum of 64 KiB of instructions can be loaded, instructions will be loaded and ran from simulated memory. A hierarchy for memory management is defined below.

Table 2.1.0

Dynamic Memory Specs	
Max Size	4 GiB
Memory Allocation	Dynamic
Address Space	Circular
Address Range	0x00000000 → 0xFFFFFFFF

Circular: Addresses 0x00000000 and 0xFFFFFFFF are adjacent. Decrementing the former will give the latter and incrementing the latter will give the former.
Dynamic Memory Allocation: Memory will be dynamically allocated when an instruction stores or loads to a memory location that hasn't been referenced yet.

Class Pseudo-Definition

```
class SimMemory
{
	public:
		uint32_t Load(uint32_t Address, int Type); //Type indicates type of load.
		Store(uint32_t Address, uint32_t Data, int Type); //Type indicates type of store.
	private:
		MemPart Partition[32]; //32 element array of Memory Partitions.
}
```

Function Pseudo-Definitions

```
uint32_t Load(uint32_t Address, int Type)
{
	return data at given address based on load type;
}
```

```
Store(uint32_t Address, uint32_t Data, int Type)
{
        store data at given address based on store type.
}
```

2.2 – Simulated Memory Structure

The smallest unit of allocatable memory is the **Page Element** (PE). The PE is an array of 256 32-bit elements and each element will contain actual data that is loaded from or stored to simulated-memory.

A **Page** is a 16 element array of pointers to PEs. It can hold 16 KiBs of allocated memory.
A **Memory Partition Element** (MPE) is a 64 element array of Pages. It can hold 1 MiB of allocated memory.
A **Memory Partition** (MP) is a 128 element array of MPEs. It can hold 128 MiBs of allocated memory.
Simulated memory is a 32 element array of MPs. It can hold 4 GiBs of allocated memory.

Hierarchy: Simulated Memory → MP → MPE → Page → PE

2.3 – Registers

There are 33 32-bit registers. Register x0 is hardwired with all bits equal to 0. Data in general purpose registers x1-x31 will be interpreted as two's complement signed integers, unsigned integers, or a collection of boolean values.
The pc (program counter) register holds the address of the current instruction.

Our simulation will use an array of 33 32-bit elements to represent registers x0 → x31. This array will be called the **Register Array** (RA). At the start of simulation, only registers x0, sp, pc, and ra will be initialized to zero. Register x0 can't be overwritten and will always remain zero.

Table 2.3.0

Register Array Initial State		
Array Index	Register	Value
RegisterArray[0]	x0	0
RegisterArray[1]	x1/ra	0
RegisterArray[2]	x2/sp	0
RegisterArray[3-31]	x3 → x31	NA
RegisterArray[32]	pc	0

2.4 – Instructions

There are four core instruction formats (R, I, S, U). All instructions are 4-byte aligned in memory.
Instruction formats B and J are two extra variant formats to handle immediates.
The RISC-V calling convention described in the following link will be used.
https://riscv.org/wp-content/uploads/2015/01/riscv-calling.pdf
Here is a useful table from above link.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Table 18.2: RISC-V calling convention register usage.

Section 3 – Simulation