

# RISC-V processors with Bluespec

High Performance Processors and System  
A.Y. 2020/2021

**Riccardo Nannini**

June 15, 2021

Tutors: Emanuele Del Sozzo,  
Davide Conficconi

Professor: Marco Domenico Santambrogio



**POLITECNICO**  
MILANO 1863

POLITECNICO MILANO 1863  
**NECST**  
laboratory

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>RISC-V</b>	<b>2</b>
2.1	What is RISC-V . . . . .	2
2.2	Instructions set goals . . . . .	3
2.3	Main features . . . . .	3
2.3.1	Registers . . . . .	4
2.3.2	Instruction encoding . . . . .	5
2.3.3	Instruction format . . . . .	5
2.4	License . . . . .	5
<b>3</b>	<b>BSV</b>	<b>5</b>

## Abstract

Bluespec System Verilog (**BSV**) is a state-of-the-art Hardware Description Language. Bluespec compilation toolchain (**BSC**) has been recently released as open source [1]. The goal of the project was investigating the potentiality of said toolchain implementing different **RISC-V** processors of increasing complexity.

## 1 Introduction

This report covers chronologically the path that I have followed during the development of this project.

It starts from a quick overview on the *RISC-V* ISA, focused on the key ingredients that make this ISA one of the most trend topics in Computer Engineering.

It proceeds with the analysis of the *Bluespec System Verilog* language, outlining its novelties with respect to other hardware description languages, as well as its main features and capabilities.

The last section is devoted to the development of various RISC-V processor designs, describing in details the characteristics of each processor and motivating the various design choices, starting from a one cycle non pipelined processor and ending with a 6 stage pipelined one enriched with multiple branch predictors.

Everything I have produced within this project (code, scripts, this report itself) is available on GitHub [https://github.com/riccardo-nannini/BSV\\_RISC-V](https://github.com/riccardo-nannini/BSV_RISC-V)

## 2 RISC-V

### 2.1 What is RISC-V

Since the processor designs will be based on RISC-V, knowing this ISA is fundamental.

RISC-V was born in 2010 at UC Berkley by the work of graduate students Andrew Waterman, Yunsup Lee and professors David Patterson, Krste Asanovic [2], built on the experience gained from the 4 previous major RISC ISA design from UC Berkeley (thus justifying the ‘V’).

For the scope of the project, the focus was put on the User-Level (or Unprivileged) section of the ISA.

As the name suggests, RISC-V belongs to the family of **RISC** ISAs. In particular, RISC instruction sets are characterized by *small* and *highly optimized* instructions that ease *pipeline* implementation, along with a large number of *general purpose registers*.

RISC-V, like many other RISC ISAs, is a **load-store architecture**; this means that, apart from some exceptions, arithmetic and logic instructions operate on register and not on memory directly. Load and store instructions take care of the transit of data from memory to register and vice versa.

## 2.2 Instructions set goals

When the first paper that declared the birth of RISC-V [2] was released back in 2011, the authors stated some of the goals that they wanted to achieve when designing this instructions set. In particular the main objectives include, but are not limited to:

- Provide a *realistic* but *open* ISA, suitable for direct hardware implementation
- Provide a **small** but **complete** ISA, that avoids "*over-architecting*" for a particular microarchitecture style or implementation technology.
- Support highly-parallel multicore implementations, including heterogeneous multiprocessors
- Support an efficient dense *instruction encoding* with variable-length instructions (generally typical of CISC ISAs).
- **base-plus-extension**: provide a set of standard but optional extensions.
- Be fully *virtualizable*.
- Be simple to subset for educational purpose or to reduce complexity in new implementations.

In order to achieve these goals many design choices were taken, sometimes in line with other similar ISAs, sometimes with unusual decisions.

## 2.3 Main features

RISC-V has 2 base variants, **RV32** and **RV64**, providing 32-bit or 64-bit user-level address spaces respectively. A 128-bit extension is available but mostly aimed at the future of warehouse scale computing.

### 2.3.1 Registers

RISC-V uses 31 general-purpose registers **x1**–**x31** with **x0** hardwired to the constant 0. Moreover, 32 64-bit registers **f0**–**f31** holding single or double-precision floating-point values are available as shown in Figure 1

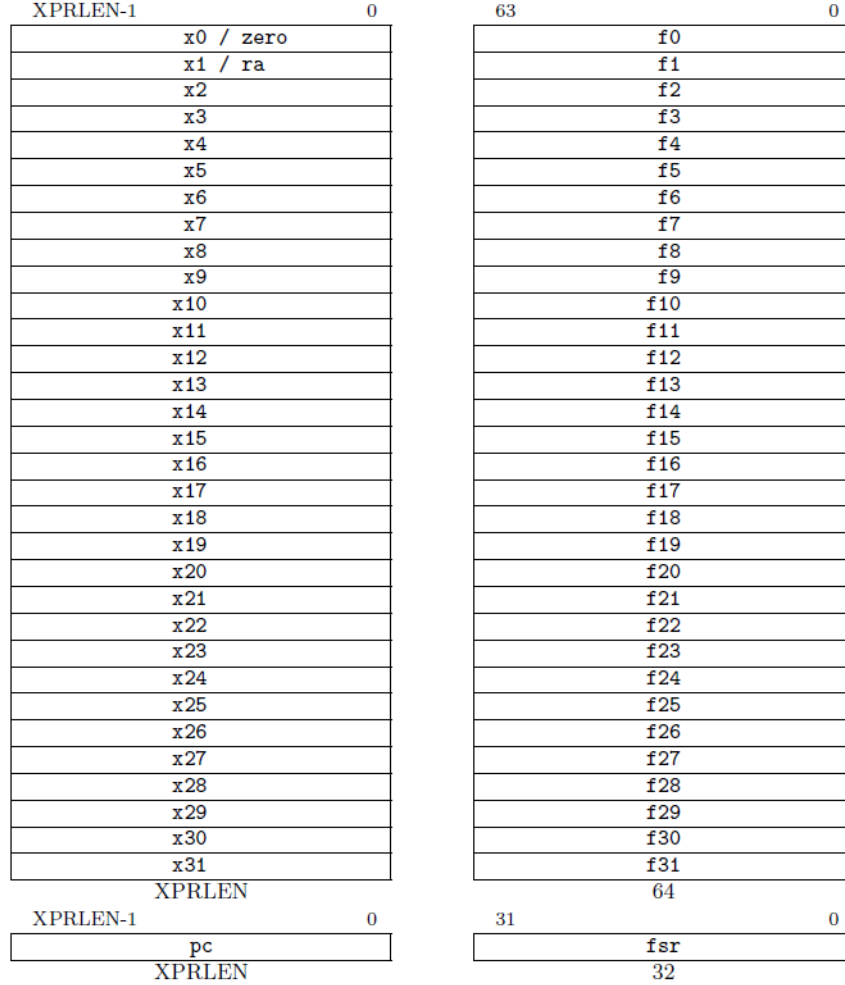


Figure 1: RISC-V user-level registers (XPRLN refers to the width of registers in bit (either 32 or 64))

There are also two special registers: the *program counter* **pc** holds the address of the current instruction while the *floating point status register* **fsr** contains the operating mode and exception status of the floating-point unit.

**2.3.2** Instruction encoding

**2.3.3** Instruction format

**2.4** License

**3** BSV

**4** Processor designs

## References

- [1] Bluespec Compiler. URL: <https://github.com/B-Lang-org/bsc>.
- [2] Andrew Waterman et al. “The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA”. In: May 2011. URL: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.pdf>.