Seminar Report

RISC-V. State of the union

Ilia Kurin

MatrNr: 22123750

Supervisor: Dr. Christian Köhler

Georg-August-Universität Göttingen
Institute of Computer Science

September 25, 2022

Abstract

There are many computer architectures, but dominant are Intel x86 and ARM. In this paper RISC-V architecture is being explored. It contains a brief history, technical details from specifications

Contents

List of Tables iii

List of Figures iii

Listings iii

List of Abbreviations iv

1 Overview 1

1.1 Introduction 1

1.2 Common architectures 1

1.3 Instruction set architecture 1

1.4 RISC-V 1

1.5 History 1

2 Technical details 2

2.1 Architectural decisions 2

2.2 Specifications 3

2.2.1 Unpriveleged 3

2.2.2 Priveleged 4

2.3 Pseudo-instructions 5

2.4 Custom Function Units 5

2.5 RISC-V compilation techniques 5

3 RISC-V ecosystem 6

3.1 Hardware companies 6

3.2 Cores 6

3.3 Boards 7

3.4 Software 7

4 Conclusions 7

References 8

A Code samples A1

List of Tables

1 Ratified extensions [And19] . . . 3
2 General-purpose registers [And19] . . . 4

List of Figures

1 Available instruction encodings [Pal17] . . . 4
2 CFU Playground structure [CFU] . . . 6

Listings

1 Pipeline for compilation and listing . . . 5
2 RISC-V assembly version of multiplication using dedicated extension. Produced by the compiler using RV64IM architecture . . . A1
3 C analog of multiplication . . . A1
4 RISC-V assembly version of multiplication using logical shifts. Produced by the compiler using RV64I architecture . . . A1
5 C analog of multiplication using logical shifts . . . A1

List of Abbreviations

RISC	Reduced instruction set computer
CICS	Complex instruction set computer
SoC	System on a chip
IDE	Integrated development environment
SDK	Software development kit
OS	Operating system
FPGA	Field-programmable gate array
IoT	Internet of things

1 Overview

1.1 Introduction

1.2 Common architectures

To understand the topic it is important to discuss two dominant approaches to computer architectures, such as Reduced instruction set computer (RISC) and Complex instruction set computer (CICS).
These architectures are quite opposite to each other. Where RISC has simple fixed-size instructions, CICS has complex instructions with variable sizes. These approaches allow for simpler pipelines with software orientation and more complicated pipelines that extensively rely on hardware for optimizations.

1.3 Instruction set architecture

Another key topic is Instruction set architectures (ISA). It is an abstract specification of a computer. Meaning that different hardware manufacturers can create different devices that follow the same specification. Dominant ISA families are Intel x86 and ARM.
ISA usually describes instructions, data types, registers, I/O models and other fundamental features.

1.4 RISC-V

Given previous information, RISC-V is ISA than is based on RISC. It is a successor of previous Berkley RISC architectures.
Its advantages are:

- *It is free to use for both academic and industrial purposes.* So there is no need to pay other companies to create RISC-V-based hardware. This approach also has a major drawback that will be discussed further.
- *It is extensible and customizable.* This is a major benefit to other ISAs, as they usually require to support outdated instructions making hardware and software more complicated and making it harder to extend architectures.
- *It avoids previous architecture mistakes.* From the start authors wanted to make architecture as simple and efficient as possible. They made multiple decisions, such as removing delay slots, register windows and other technical decisions that proved to be wrong though Berkley RISC history.

1.5 History

RISC-V history dates back to 1981. That year the first version of Berkey RISC architecture was released. RISC group led by David Patterson was a part VLSI Project at the University of California, Berkeley. This project was a success and later inspired ARM architecture. [Wik]
The second version was released in 1983 and introduced an expansion of 16-bit instructions to 32-bit. The third version was released in 1984 by Patterson's students and was

designed to run Smalltalk. The fourth version was released in 1988 by Patterson's students and was designed to run Lisp
In 2010 began work on RISC-V. It was started at UC Berkeley in Parallel Computing Laboratory. In development participated Prof. Krste Asanović, Prof. David Patterson, Yunsup Lee and Andrew Waterman. The first ISA was published in 2011 and contained base instructions with floating-point and compressed extensions.
In 2015 RISC-V Foundation was founded. It is a non-profit corporation curating ISA development and openly accepting individual members. In 2019 it was renamed to RISC-V International and later in 2020 moved to Switzerland to avoid USA trade regulations. Now this organization consists of many well-known companies, such as Google, Qualcomm, Nvidia, IBM and others.[Hla21]

2 Technical details

2.1 Architectural decisions

To make the RISC-V architecture more universal and extensible authors followed these architectural decisions:

- *Smaller instruction set.* The Base RISC-V instruction set contains only 40 instructions [**unpriveleged**], which is much less than in Intel x86 or ARM. It takes less time to get familiar with the instructions, therefore, making code easier to write. A small instruction set also means that hardware and software can be simpler and faster.

- *Load/store architecture.* In load/store architecture all operations are either memory access (load from memory and store in registers) or arithmetic operations between registers. This allows to change instruction execution order improving pipeline utilization. Costly load instructions can be scheduled earlier to avoid bottlenecks. The majority of new RISC architectures use this approach.

- *No branch delay slot.* Conditional branching is a process of executing or jumping from one command to another based on a condition. This process requires to wait while the branch instruction pipeline has finished to compute the target address and load it into a program counter. This was a problem as pipelined architecture should execute an instruction every cycle. To avoid these delays, branch delay slots were used. They allow to execute following code independently of branch instruction by moving instructions into branch delay slots during compilation if the hardware supports it. But this doesn't scale well and is not needed in the case of multicycle and superscalar CPUs. Another side effect is that exception handling and debugging are becoming more complicated and requiring a special approach to manage these delay slots. [Joh].

- *No register windows.* Branching to subroutines often requires many stack operations to pass parameters and restore register values during return. Register windows are used to address this problem and make branching much faster. All internal registers are divided into overlapping windows assigned for subroutines where only one window is visible at a time. Despite its success in Berkley RICS and its adoption in multiple ISAs like Sun Microsystems SPARC, nowadays it is often considered a bad practice. One of the reasons is with an increasing number of recursive subroutines speed of new subroutine allocation drastically decreases due to needing to save some registers to main memory or cache that may have much slower access speed. Another reason is that multiprocessor systems very often use context switches to save the current state of a program before switching to another one. This process includes saving the state of all the registers and due to the overlapping structure of register windows saving all windows will be more expensive. The solution may lay in compile time optimizations of existing registers utilization. [Mag97]

2.2 Specifications

RISC-V specification is divided into two major ISAs which are privileged and unprivileged and multiple non-ISA specifications. The specifications are well-written and contain many insights into the purposes of architectural decisions that were made.

2.2.1 Unpriveleged

The unprivileged level contains all the core information including architectural approach, naming conventions, extensions, instruction formats, etc.
The unprivileged specification contains some important hardware definitions. *Core* is is an independent fetch unit. *Hart* is a hardware thread. Each core can have multiple harts through multithreading. Another important term is the accelerator. *Accelerator* is a non-programmable unit or core with a fixed function operating autonomously. [And19]
The unprivileged specification defines naming conventions. Base pattern is **RV [xxx] [abc. . . xyz]**, where xxx are address space size (32, 64, 128) and abc. . . xyz are extension letters. Underscores between extensions are supported. Extensions are case insensitive. Non-standard extensions are supported.
Extensibility of RISC-V is achieved by dividing the functionality into multiple extensions. All the standard extensions are developed collectively between commercial and non-commercial institutions. List of currently approved extensions shown in Figure 1. The only required extension is Integer which describes basic integer operations, registers and instruction encodings. This extension can emulate any other except Atomics which can only be implemented in hardware. A combination of I, M, A, S, D extensions is required for OS such as Linux to function, therefore it is used quite often. That is the reason why the such combination is usually written as G.

Name	Letter	
Integer	**I**	
Integer multiplication and division	M	
Atomics	A	G
Single-Precision floating-point	F	
Double-Precision floating-point	D	
Quad-Precision floating-point	Q	
16-bit Compressed instructions	C	
Control and status register access	Zicr	
Instruction-Fetch fence	Zifencei	

Table 1: Ratified extensions [And19]

General-purpose registers are also defined here.

Register	Name	Usage	Preserved
x0	zero	Hardwired to 0, ignores writes	n/a
x1	ra	Return address for jumps	no
x2	sp	Stack pointer	yes
x3	gp	Global pointer	n/a
x4	tp	Thread pointer	n/a
x5-x7	t0-t2	Temporary registers	no
x8-x9	s0/fp-s1	Saved registers	yes
x10-x11	a0-a1	Function arguments / Return values	no
x12-x17	a2-a7	Function arguments	no
x18-x27	s2-s11	Saved registers	yes
x28-x31	t3-t6	Temporary registers	no
pc	(none)	Program counter	n/a

Table 2: General-purpose registers [And19]

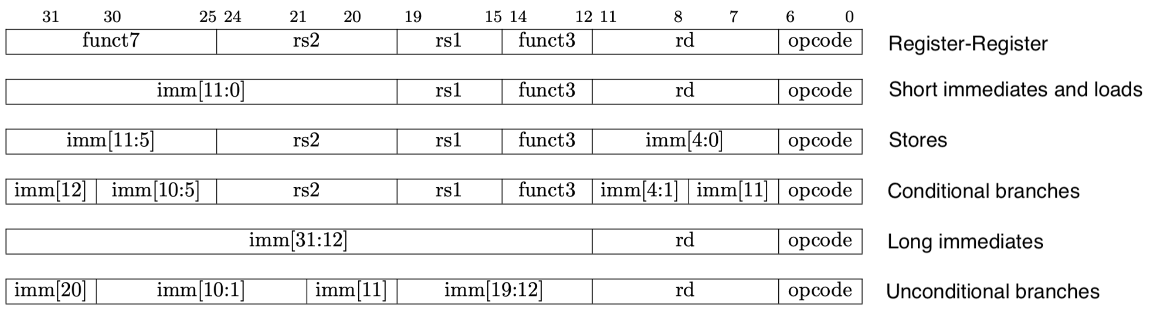
31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode
Register-Register
imm[11:0] | rs1 | funct3 | rd | opcode
Short immediates and loads
imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode
Stores
imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode
Conditional branches
imm[31:12] | rd | opcode
Long immediates
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode
Unconditional branches

Figure 1: Available instruction encodings [Pal17]

2.2.2 Priveleged

The privileged specification defines privilege levels and their complete specifications, control and status registers and some low-level extensions.
RISC-V offers three privilege levels to provide solid isolation and protection. Usually, architectures should have at least two privilege levels: user level and system level. This allows to isolate important system processes from user one avoiding the whole system crashing when a single app crashes. For security reasons, it is also important to forbid access to the memory of one user process from another. The required level of privilege depends on the task and thus all the system and hardware calls are being done using the system level of privilege. A processor can be in only one privilege level at a time but can switch from one to another.
Machine level is the most privileged one. It controls all the physical resources and system interrupts. This level is considered trusted as it has low-level access to hardware. RISC ISA allows this level to be the only mandatory. In that case, the whole environment is considered secure. Thus there will be no protection from incorrect or malicious application code. [And21]

Supervisor. level is intended for operating system kernels and drivers. This level can only be used with both User and Machine privilege levels. The supervisor level can be extended with Hypervisor extension to support hypervisor execution, which can even be considered the fourth privilege level. [And21]
User/Application. level. On this level, an application code is executed. It provides a boundary between applications and hardware access [And21]

2.3 Pseudo-instructions

RISC-V assembly language often uses pseudo-instructions. Pseudo-instruction is shorthand for one or more machine instructions. Most of them use zero register as its usage improves performance due to pipeline and command microcode optimizations. For example, there is no real instruction to compute Two's complement, so **neg rd, rs** is being converted to **sub rd, zero, rs**. It just subtracts the value (rs) from zero and writes to the destination register (rd)

2.4 Custom Function Units

Custom Function Units (CFU) are part of the Composable Custom Extensions Project proposed in 2019. CFU is a hardware core that follows a specific interface and provides custom instructions. The goal of this proposal is to create a solution allowing to create reusable extensions for any hardware with no need to wait multiple years for ratification. By that CFU combines the advantages of standard and custom extensions. The work is in progress and the project is still in the draft stage.
Despite being in the draft stage, there are already options to create your own CFUs. The most noticeable is CFU Playground. It is a framework that allows to build and test CFUs for Machine Learning. It requires LiteX Boards FPGA board or Renode and Verilator to simulate it. [CFU]
The framework consists of three layers: software, gateway and hardware that can be emulated.

2.5 RISC-V compilation techniques

To make the process of writing software easier, RISC-V compilers can adapt generated machine code to the target architecture. In this section, we will explore how multiplication extension affects generated code
For compilation and listing will be used the pipeline shown in Listing 1. The following code examples were compiled with GCC 11.1.0 on MacOS.

```
riscv64-unknown-elf-gcc -march=*Architecture* -g3 -o test test.c
riscv64-unknown-elf-objdump -S test

```

Listing 1: Pipeline for compilation and listing

Target C code consists of a small function to multiply values, which is shown in Listing 3. For RV64IM architecture this function was compiled to efficient code that uses multiplication instruction from a dedicated extension (Listing 2). The same C code was compiled for RV64I architecture, the compiler produced the code presented in Listing 4. In this method multiplication is being done by multiple logical shifts. The C analog of this method is shown in Listing 5.

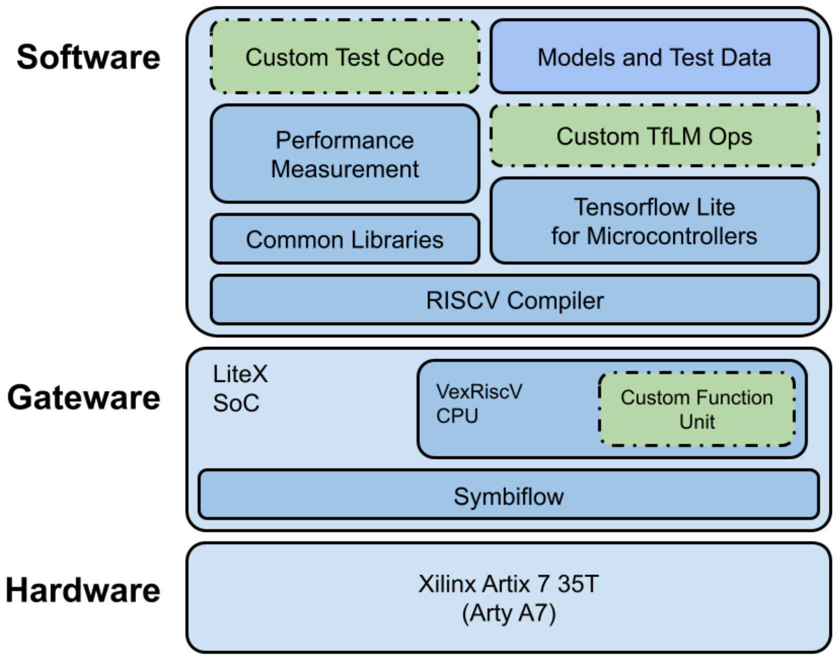

Figure 2: CFU Playground structure [CFU]

3 RISC-V ecosystem

3.1 Hardware companies

There are multiple companies producing hardware based on RISC-V ISA. But there are two noticeable.
SiFive is a leading RISC-V hardware company. It was founded in 2015 by three authors of the original ISA, Krste Asanović, Yunsup Lee and Andrew Waterman. In November 2016 the company released Freedom Everywhere 310 SoC, making it the first company to produce a chip that implements RISC-V ISA. [Tak17] This March the company received $175 million funding, valuing the company at over $2.5 billion [DeL22]
Andes Technology is the biggest supplier of embedded RISC-V cores that is engaged with more than 150 partners for the past 15 years. It was founded in Taiwan in 2005 and joined RISC-V International in 2016 becoming a premier member in 2020. Andes adopted RISC-V ISA in 2017 producing AndeStar™ V5 architecture. [tec] Andes is a major toolchain contributor.

3.2 Cores

Western Digital SweRV EH1. It has an RV32IMC core with a branch predictor. Its target frequency of 1Ghz. It has a 4-way set-associative instruction cache, bus interfaces for instruction fetch, data access, debug access and external Direct Memory Access [Dig17]
SiFive U74. It has RV64GC_Zba_Zbb_Sscofpmf core. It has 8 regions of memory, physical memory protection, and virtual memory support with up to 47 Physical Address bits. The L2 Cache can be configured into high-speed deterministic SRAMs. [SiF]

3.3 Boards

Waveshare ESP-C3-32S-Kit. It has a RISC-V processor ESP32-C3 with a single RV32IMC core. It supports Bluetooth 5. It supports a clock frequency of up to 160 MHz, with 400 KB SRAM, 384 KB ROM, 8KB RTC SRAM. It is intended to be used in IoT, mobile devices, wearable electronics, smart home, etc. The current price is 12.99 Euro. [Wav]
Sipeed Nezha. SoC: D1. Single RV64GCV core (XuanTie C906). Memory: 1 GByte of DDR3 and 256 MByte of Nand Flash. HiFi4 Digital Signal Processor. G2D 2D graphics accelerators. Usage: Linux, IoT. Prices: 128.35 - 250.97 Euro. [Sip]

3.4 Software

RISC-V has a rich software ecosystem. It consists of simulators, IDEs, SDKs, OS, toolchains, debuggers, compilers and runtimes for many languages, machine learning libraries and more. [Int]
It supports all major OS like Linux, MacOS and Windows. It has ports for Golang, Java, Rust and even Node.JS. It includes GCC, Clang, Glibc, Newlib and other compilers. And it also includes Fedora, Debian, Ubuntu and other Linux distributions.
There are different simulation options for different purposes. Thus, *QEMU* is a great option for software developers. It is fast enough for interactive use and easy to understand. *SPIKE* is great for hardware developers.

4 Conclusions

RISC-V is an interesting technology that is still in the development stage. And Today it is far behind Intel or ARM in terms of performance. RISC-V ISA is truly free and open-source, but hardware can be private and companies could easily share the market in such a way that you would need to buy licenses for their cores, accelerators or custom extensions to stay compatible.
Extensibility could also be a great feature in the long-run. Unlike Intel or ARM, RISC-V does not require the support of old instructions. It can make adding new instruction sets easier. Custom Function Units can help to standardize custom extensions.
RISC-V can easily find its way to IoT and micro-controller sectors or become an alternative for Arduino developer boards as it highly customizable and memory efficient.
Although the ISA is also intended to be used by supercomputers, there is almost no dedicated hardware at the moment. However, High-Performance Computing Group has been organized to discuss RISC-V usage in HPC.

References

[And19] Krste Asanovic Andrew Waterman. “Volume I: Unprivileged ISA”. In: *The RISC-V Instruction Set Manual* (Dec. 2019). Version 20191213.

[And21] John Hauser Andrew Waterman Krste Asanovic. “Volume II: Privileged Architecture”. In: *The RISC-V Instruction Set Manual* (Dec. 2021). Version 20211203.

[CFU] CFU-Playground. *The CFU Playground: Accelerate ML models on FPGAs.* Accessed on 01.06.22. URL: `https://cfu-playground.readthedocs.io/en/latest/index.html`.

[DeL22] Allison DeLeo. “SiFive Leadership in RISC-V Powers $2.5B+ Company Valuation”. In: (Mar. 2022). URL: `https://www.businesswire.com/news/home/20220316005396/en/SiFive-Leadership-in-RISC-V-Powers-2.5B-Company-Valuation`.

[Dig17] Western Digital. “RISC-V SweRVTM EH1 Programmer’s Reference Manual”. In: (Feb. 2017). URL: `https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf`.

[Hla21] Nthatisi Hlapisi. *Understanding RISC-V Architecture and Why it could be a Replacement for ARM.* Feb. 2021. URL: `https://circuitdigest.com/article/understanding-risc-v-architecture-and-why-it-could-be-a-replacement-for-arm`.

[Int] RISC-V International. *RISC-V Software Ecosystem Overview.* Accessed on 15.09.22. URL: `https://github.com/riscvarchive/riscv-software-list`.

[Joh] Henry M. Levy John A. DeRosa. “An Evaluation of Branch Architectures”. In: (). URL: `https://dl.acm.org/doi/pdf/10.1145/30350.30352`.

[Mag97] Peter Magnusson. “Understanding stacks and registers in the Sparc architecture(s)”. In: (Apr. 1997). URL: `https://archive.ph/20121224205243/http://www.sics.se/~psm/sparcstack.html#selection-15.0-15.63`.

[Pal17] Alex Bradbury Palmer Dabbelt Michael Clark. *RISC-V Assembly Programmer’s Manual.* Accessed on 15.09.22. 2017. URL: `https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md`.

[SiF] SiFive. “SiFive U74 Core Complex Manual”. In: (). Accessed on 01.06.22. URL: `https://sifive.cdn.prismic.io/sifive/ad5577a0-9a00-45c9-a5d0-424a3d586060_u74_core_complex_manual_21G3.pdf`.

[Sip] Sipeed. *Sipeed Nezha 64bit RISC-V Linux SBC Development Board Allwinner D1@1.0GHz with 1GByte DDR3, Supports Tina/Debian System (Standard Set).* Accessed on 13.09.22. URL: `https://www.amazon.de/Sipeed-Development-Allwinner-Support-Standard-Set/dp/B09BXM2PXY`.

[Tak17] Dean Takahashi. “SiFive launches open source RISC-V custom chip”. In: (Nov. 2017). URL: `https://venturebeat.com/business/sifive-launches-open-source-risc-v-custom-chip/`.

[tec] Andes technology. *Andes technology.* Accessed on 13.09.22. URL: `http://www.andestech.com/en/homepage/`.

[Wav] Waveshare. *WaveShare ESP32 WiFi+Bluetooth Development Board ESP-C3-32S-Kit*. Accessed on 13.09.22. URL: https://eckstein-shop.de/WaveShare-ESP32-WiFiBluetooth-Development-Board-ESP-C3-32S-Kit-EN.

[Wik] Wikipedia. *Berkeley RISC*. Accessed on 15.09.22. URL: https://en.wikipedia.org/wiki/Berkeley_RISC.

A Code samples

```
lw     a5,-20(s0)
mv     a4,a5
lw     a5,-24(s0)
mulw   a5,a4,a5
sext.w a5,a5

```

Listing 2: RISC-V assembly version of multiplication using dedicated extension. Produced by the compiler using RV64IM architecture

```
int multiply(int a, int b) {
   return a * b;
}

```

Listing 3: C analog of multiplication

```
00000000000101f8 <__muldi3>:
  mv    a2,a0
  li    a0,0
  andi  a3,a1,1
  beqz  a3,10204 <__muldi3+0xc>
  add   a0,a0,a2
  srli  a1,a1,0x1
  slli  a2,a2,0x1
  bnez  a1,101fc <__muldi3+0x4>
  ret
```

Listing 4: RISC-V assembly version of multiplication using logical shifts. Produced by the compiler using RV64I architecture

```
 int multiply(int a, int b) {
    int result = 0;
    while (b > 0) {
        if (b \& 1) {
            result += a;
        }
      b = b >> 1;
      a = a << 1;
    }
    return result;
}

```

Listing 5: C analog of multiplication using logical shifts