Advantages and disadvantages of the RISC-V ISA (Instruction Set Architecture) in comparison to the ARMv8 ISA

1st Michael Schneider
Faculity of Computer Science and Mathematics
OTH Regensburg
Regensburg, Germany
michael4.schneider@st.oth-regensburg.de

2nd Florian Henneke
Faculity of Computer Science and Mathematics
OTH Regensburg
Regensburg, Germany
florian.henneke@st.oth-regensburg.de

3rd Alexander Schmid
Faculity of Computer Science and Mathematics
OTH Regensburg
Regensburg, Germany
alexander2.schmid@st.oth-regensburg.de

***Abstract*—text**
***Index Terms*—keyword1, keyowrd2**

I. Introduction

A. Topic

B. Motivation

C. Goal

D. Overview of paper

II. Background

A. Instruction Set Architectures

B. RISC

C. ARM

text

D. RISC-V

text

III. Concept and Methods (Initial section written by Alexander Schmid)

Given that the goal of this paper is to compare the two Instruction Set Architectures (ISAs) across a set of criteria that are relevant to semiconductor companies when evaluating which ISA to use with a new CPU design, the following section defines these criteria.

The first of these criteria is the ISA's **business model**. There are often a number of patents protecting an ISAs that prohibit anyone not licensed by the patent owner from distributing Central Processing Units (CPUs) that implement that ISA [**?**].

Whether these patents exist and the licensing terms are an important factor when deciding which ISA to use. The two ISA's business models are compared in chapter **??**.

The ISA's **complexity** refers to the amount of effort required to implement the ISA. The more complex an ISA is, the more developer time is spent on implementing and verifying the implementation of the ISA, instead of optimizing the CPU for performance and efficiency, increasing a CPU's development cost [**?**]. For the comparison of both ISAs's complexity, see chapter **??**.

The next aspect is a CPU's **performance**. There are various aspects that make up a CPU's performance. Among these are the code size and execution speed. The CPU's ISA influences each of these performance aspects to varying degrees. Chapter **??** elaborates the extent of that influence and compares the two ISAs in these performance aspects, where possible.

ISAs often allow for a number of instruction set **extensions** that may or may not be implemented by a given CPU. The choice of extensions greatly influence the flexibility of an ISA and its performance for specific use cases. Thus they are an important factor to consider which ISA to implement for a new CPU.

An ISA's **ecosystem** refers to the software that supports that ISA, especially compilers that compile to that ISA, operating systems and libraries. When developing a new CPU it is preferrable to use an ISA with a large ecosystem, in order to maximize the amount of software that can run on that CPU. This is especially important in consumer desktop and mobile devices where a large variety of software is to be executed.

A. Business Models (written by Florian Henneke)

[Introduction shortend because of word limit]

Beginning with the classic model of ARM, the following sections will cover the two license models: ARM sells its ISAs in various licensing models. [**?**] These are staggered in multiple levels of access. The 'Design Start' level includes

free access to the IP-Core of the simplest ARM Chips Cortex-M0 and Cortex-M3 aswell as the corresponding toolchain and processor models. For a fee between $0 and $75K the IP-Core of the Cortex-A5 aswell as the permission for 'single use' chip production can be aquired. This allows the customer to produce and sell one type of chip for a single purpose e.g. a network controller. For every chip produced, a royality must be given to ARM. The 'Design Start' access level also includes a license for accessing a 'artisan physical IP library', a license for universities which includes teaching and prototyping and allows non commercial production of own chips without royalities in small volumes. At last there is an '' license which is free and includes a optimized version of the Cortex-M3 and M1. Production is not allowed with this license.

[Removed short section due to word limit]

Above those access levels, there only officially exists the 'Standard' licensing model. This means one makes a individual contract with ARM. Several articles from 2013 [**?**] [**?**] talk about an older licensing model which contains special categories for higher access licenses. The highest of these, often referred as the 'Architectural' license is the only one that allows editing of the ISA and developing completely freely. The most prominent companies with such a license are Qualcomm which develops and sells mobile phone chips and Apple who just announced a 'Apple Silicon' developed laptop chip based on ARM. [**?**] The article also mentions that preparing a license of this form often takes about 6-24 months and states per-chip royalities of about 1-2.5%. It also notices so called 'foundry contracts' where customers can buy silicon ready ARM designs in cooperation with a silicon foundry. The most prominent example here are the Mali GPUs. This offers customers a fast and easy way to expand their chip with, for example, graphic accelerators.

In contrast to the ARM license model, RISC-V is published using the 'Creative Commons Attribution 4.0' license. [**?**] [**?**] This license allows the user to 'share' and 'adapt'. This means you are free to copy and redistribute as well as modify, change, build upon and sell it commercially. It is not necessary to share changes in an open source manner and you are only restricted by the obligation to give credit to the original licensor. [**?**] An important addition is also, that the license cannot be revoked by the licensor. This means everything about RISC-V that is already published will always be free to use. Originally founded by Berkley University, the RISC-V ISA standard is now managed by the nonprofit organization 'RISC-V International', founded in 2015. [**?**] As the statutes of the organization include, the association has 'no pecunuary, self-help or commercial purpose' [**?**]. Running expenses and further development of the standard do however require a certain liquidity. This is ensured by a membership program surrounding the specification. [**?**] Resembling the ARM licensing model, it contains three levels: 'Premier', 'Strategic' and 'Community'. Costing between $2K and $250K annually, these levels do not restrict access to the ISA, but grant several levels of taking influence on the future development of the standard through seats in the 'Technical Steering Commitee', speaker slots on conferences and representation on the official RISC-V International website and blog. There are also three 'Strategic Directors', which are elected out of the 'Premier' and 'Strategic' Members and one Academic as well as one Communtiy Director, which is elected by the 'Communtiy' level of members. [**?**]

Besides taking influence in the development process, the membership also includes help in designing CPU Cores, teaching for employees and more. It also allows the usage of the trademark 'RISC-V'.

B. Complexity (written by Michael Schneider)

Risc-V and ARMv8 are both Reduced Instruction Set Computer/Computing (RISC) based architectures. Various RISC ISAs are different in complexity. To compare those differences, the basic instruction sets with corresponding extensions, the different realisations and two basic assembly instructions will be covered in the next chapters.

1) Instruction sets: In RISC-V the only mandatory instruction set is the Integer instruction set. Those base integer instructions cannot be redefined, only extended by optional instruction sets. Further details about the extensions are in chapter **??**. This concept makes the RISC-V architecture only as complex as necessary, because the ISA can be tailored to a specific application. [**?**]

ARM instead defines the ARMv8 architecture in a completely different way. The ARMv8 alread supports many more extensions in the basic version, also called v8.0. Further extensions are available in later versions, as explained in the chapter **??**. [**?**]

Because almost all the optional extensions of RISC-V are covered by the basic v8.0, the ARMv8, regarding only the instructions, is at least as complex as a fully extended RISC-V architecture.

2) Instruction set implementations: The different ISAs are able to implement the explained instruction sets in various ways. The RISC-V architecture is able to implement the instruction sets in 3 different word length, a 32-bit (RV32I), a 64-bit (RV64I) and a 128-bit version (RV128I). For the 32-bit and the 64-bit implementations are also multiple subversions available, RV32E and RV32G/RV64G. For 128-bit, RV128I is the only 128-bit implementation so far. RV32E is a version with only 15 instead of 31 registers. The subversion RV32G/RV64G is less a own version than a stable release. The RV32G/RV64G is combining a basic ISA (RV32I or RV64I) plus different selected standard extensions (IMAFD). [**?**]

Also ARMv8 has, different implementations, A64, A32 and T32. A64 is the 64-bit version and A32, T32 are both 32-bit versions. AArch64 and AArch32 are two different execution states in ARM (AArch64 for 64-bit and AArch32 for 32-bit). These execution states support the A64 instruction set in AArch64 and A32 and T32 in AArch32. [**?**]

3) registers and access: To complete the overview in a, for users more abstract point of view, two basic assembler commands (load and store) are compared. To load a value from a RISC-V register, LW, LH or LB is used. The "L" means load and the following characters stand for word (32 bit), halfword (16 bit) and byte. LH and LB are signed and can be extended by an "U" (LHU, LBU) to load unsigned values. [**?**] All instructions take 2 parameters, a register to store the value in and an address to load the value from. The address consists of the value stored in a register with an immediate offset. The store instructions SW, SH and SB work in the same way. SW stands for store word, SH store halfword and SB means store byte. The commands are structured in the same way as the loading commands are. The left side of the command is the register to take the value from and the second parameter is the register, containing the address, and the offset, where the value should be stored. [**?**]
ARM instead has a few more instructions but the basics are almost the same. The (LDR) command can be extended by an B to load only a byte, SB to load a signed byte, H to load a half word, SH to load an signed halfword and SW to load a signed word. To store a value, there are 3 possible ways STR to store the complete register, STRB to store a byte and STRH to store a halfword. These basic load and store commands are followed by: load/store pairs, unscaled offsets and much more. Because there is no such possibility in RISC-V, there is no comparison about them. [**?**]

C. Performance (Alexander Schmid)

Code size is the performance parameter most influenced by the ISA and not by the CPU's implementation, given that the ISA and the compiler are the only two factors that influence code size. Because of this, code size is the most meaningful performance aspect to compare when evaluating the performance of different ISAs. Given that code size is most critical in embedded applications, the Embench benchmark suite is a good benchmark with which to compare code sizes [**?**]. It consists of a number of programs frequently used in embedded applications, such as CRC, signal filtering, AES and QR code reading [**?**]. When compiling the Embench suite for both RV32IMC as well as 32-bit ARM with the Thumb-2 extension for code compression (called T32 in ARMv8, see chapter **??**) using GCC 7, the code for RISC-V is approximately 11% larger than the code for ARM [**?**]. Part of this gap in code size can be explained by the relative immaturity of the RISC-V implementation of GCC. RISC-V support for GCC was introduced in 2017 and the code size of the Embench suite compiled for RISC-V is lower with newer versions of GCC, however it is still larger than the code generated for ARM as of 2019 [**?**].

In [**?**] an extension for RISC-V is introduced, called HCC, that is aimed at reducing the code size of RISC-V. This extension brings the code size gap down to 2.2% for the Embench suite and makes the RISC-V code smaller than ARM by 1.75% in a proprietary IoT benchmark developed by Huawei. [**?**]

For 64-bit code size, the SPEC CPU2006 benchmark, when compiled to both RV64C as well as ARM A64 using GCC 5.2, is over 20% larger on ARM as compared to RISC-V [**?**, page 62]. This difference is mainly caused by the fact that ARM does not include a compressed instruction set like T32 for 64-bit, while RISC-V does. Code size is less important in applications where 64-bit CPUs are typically used, because these applications are not as constrained in program memory size as embedded applications. A more dense 64-bit instruction set does however allow better utilization of the instruction cache, leading to better execution time [**?**, slide 46].

An issue when comparing execution speed is that the implementation of a specific CPU has a far greater impact than the ISA. One way to isolate the influence of the ISA is by only comparing the dynamic instruction count. The dynamic instruction count refers to the number of instructions the CPU executes when executing a specific program and, like code size, it is only influenced by the ISA and compiler.

For the SPECint portion of the SPEC CPU 2006 benchmark compiled with GCC 5.3, the dynamic instruction count is approximately 10% larger for RV64G as compared to ARM A64 [**?**, slide 38]. This difference can be explained by the presence of complex memory instructions in ARM that require multiple instructions for the same task in RISC-V [**?**, slide 40]. These instructions may however be split into multiple micro-operations on an ARM CPU via a process called micro-op generation [**?**, slide 40]. Similarly, an optimizing RISC-V CPU might execute multiple instructions that often occur together as if they were one instruction. This process is called macro-op fusion [**?**, slide 16]. When accounting for both micro-op generation as well as macro-op fusion, there is no significant difference in cycle count between RISC-V and ARM in the aforementioned SPECint benchmark [**?**, slide 38].

TODO: Remove power consumption.

D. Extensibility

What instruction set extensions are there for both ISAs? Who can develop new extensions?

E. Ecosystem

Which compilers support ARM and RISC-V? Which operating systems and libraries?

IV. Discussion

A. Advantages of ARM

What are the advantages of ARM compared to RISC-V?

B. Advantages of RISC-V

What are the advantages of RISC-V compared to ARM?

C. Future directions and challenges

How can we more accurately measure performance differences between ARM and RISC-V and how do ISA extensions affect performance?

V. Conclusion and Outlook

A. Summary of results

B. Interpretation of results

C. Future directions

VI. Overview of Literature

Alexander Schmid [?] [?] [?] [?] [?] [?] [?] [?] [?]
Florian Henneke [?] [?] [?] [?] [?] [?] [?] [?]
Michael Schneider [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?]