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 (Alexander Schmid)

An Instruction Set Architecture (ISA) refers a specification of instructions that a computer's Central Processing Unit (CPU) can execute. The ISA defines the binary format of these instructions, their semantics and certain aspects of the assembler language associated with that ISA [1, page 20-23]. A program in it's binary form can be executed on any CPU that implements the ISA and runs the operating system for which the program was compiled for [1, page 71].

As of 2020, there exist many competing ISAs, which limits the portability of compiled software. Reasons for this fragmentation include the fact that a CPU's ISA influences the CPU's development cost, performance (see chapter III-C) and complexity (see chapter III-B) and the fact that ISAs are usually protected by patents owned by the developer of that ISA (see chapter III-A).

On desktop computers and servers, the x86-64 architecture has been traditionally dominant, while the ARM architecture is widely used in mobile and embedded devices [2].

B. RISC (Alexander Schmid)

x64 and ARM are examples for two different paradigms in ISA design, the Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC) architecture. Early designs of integrated microprocessors have featured powerful instructions with most of them allowing access to memory, and a small number of registers [3]. This is mainly done to make programming in assembly easier [1, page 73].

This approach became known as CISC in contrast to newer designs using the RISC approach. RISC ISAs feature a larger number of registers with simpler instructions. Most RISC instructions only allow registers as operands with dedicated load and store instructions for accessing memory [3].

The RISC approach was initially favored due to improved performance and power efficiency [3].

However, a recent empirical study comparing CISC x86-64 and RISC ARM processors shows no difference in performance or power efficiency for workloads exceeding those of embedded microcontrollers [2].

RISC remains the preferred approach for new ISAs due to the lower complexity resulting in less expensive and faster development of new CPU designs [3].

C. ARM (Michael Schneider)

ARM (advanced RISC machines) is one of the most common RISC based ISAs. In 1985 the first ARM processor, at this time named Acorn RISC Machine was released. In 1990 the ARM Ltd. like we know it, was born by a joint venture of Apple, VLSI Technology and Acorn. [4]

Now ARM is one of the most licensed an well known manufacturer of ISAs. Especially for IOT and mobile devices ARM is very common, wherefore 2016 34% of shipped SoCs where ARM based. [5, slide 7-10]

From the beginning till now ARM is strictly closed source, which means everybody who wants to develop a processor based on the ARM architecture has to pay for the license to legally build a processor (see chapter III-A). ARM has a very

long history (up to 35 years) of building architectures and this made it possible to give the ARMv8 a big feature set, good performance (see chapter III-C), as well as the ability to be extended by extensions (discussed in chapter III-D).

D. RISC-V (Michael Schneider)

In 2010 the UC Berkeley with director Prof. David Patterson, more precisely professor Krste Asanovi´c and the graduate students Yunsup Lee and Andrew Waterman, started the foundation of the RISC-V instruction. The start of the architecture was made in the Parallel Computing Laboratory (Par Lab), where also the Chisel hardware construction language, which was used to design many RISC-V procesors, was developed. As it was only a five-year project funded by Intel and Microsoft, later on in 2015 the final RISC-V Foundation was founded to create an community of software and hardware innovators and later on in 2018 a collaboration with the Linux Foundation was announced. [6] Further RISC-V is developed as a completely open ISA, means it is freely available to academia and industry. The ISA trys also to avoid "over-architecting" by providing only a small base ISA together with different extensions. [7, page 1]

III. Concept and Methods (Initial Section by Alexander Schmid)

Given that the goal of this paper is to compare the two 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 CPUs that implement that ISA [8].

Whether these patents exist and the licensing terms are an important factor when deciding which ISA to use.

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 a CPU's compatibility to the ISA, instead of optimizing the CPU for performance and efficiency, increasing a CPU's development cost. [9]

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 III-C elaborates the extent of that influence and compares the two ISAs in these performance aspects, where possible.

A program's code size is greatly influenced by the ISA, since the instruction set and the compiler used are the only two factors that influence code size. As such, the code size is an important factor to consider when choosing which ISA to implement for a new processor design, especially for microcontrollers that are usually very constrained in the size of their program memory. For the other performance aspects, it is debatable to which extent they are influenced by the CPU's instruction set as opposed to the concrete implementation of the CPU. [2] [10]

ISAs often allow for a number of instruction set **extensions** that may or may not be implemented by a given CPU. These usually allow faster and more efficient processing of programs for a given use case, such as Single Instruction, Multiple Data (SIMD) extensions that optimize signal processing and media applications [11, page 52], Advanced Encryption Standard (AES) extensions that optimize cryptography [12] or ISA-extensions with shorter instructions for applications that are constrained in program memory. [13] Having a small base instruction set with many fine grained extensions improves the flexibility of the ISA, allowing CPUs to be optimized for specific use cases, increasing performance and efficiency for those use cases. ISA extensions do however pose a disadvantage when distributing precompiled software to end users, as a piece of software that uses a certain ISA extension can't be executed on CPUs that don't implement that extension, potentially increasing the number of different versions of that software that need to be distributed.

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 (Florian Henneke)

When taking a look at the business model of the two rivaling ISAs there will be two substantially different approaches. While ARM takes the traditional path of licensing its intellectual property to semiconductor companies, RISC-V stands out with the completely different way of publishing its ISA in an open source manner. This includes giving away their ISA definition for free, which raises the standard questions criticizing open source material.

Beginning with the classic model of ARM, the following sections will cover the two license models: ARM sells its ISAs in various licensing models [14]. Multiple levels of access are definded within them. The 'Design Start' level includes free access to the IP-Core of the simplest ARM Chips Cortex-M0 and Cortex-M3 as well as the corresponding toolchain and processor models. For a fee between $0 and $75K [14] you can aquire the IP-Core of the Cortex-A5 aswell as the permission for 'single use' chip production. 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, ARM demands a specified royality. 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 royalties in small quantities. At last there is an 'FPGA' license which is free and includes a Field-programmable Gate Array (FPGA) optimized version of the Cortex-M3 and M1, but does not allow production.

The next level of access is called 'Flexible Access' and contains two license models, one for $0 to $75K which allows one tape-out per year. On top of the entry price one pays per used processor design and a royality per produced chip. The other model starts at $200K per year and uses the same payment additions as the first one. But it allows unlimited tape-outs and includes employee trainings, design tools and design support. [14]

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 [15] [16] 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 unobstructed. 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 [17]. The article also mentions that preparing a license of this form often takes about 6-24 months and states per-chip royalties 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 publishes its intellectual property using the 'Creative Commons Attribution 4.0' license [7] [18]. 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 [19]. 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 [20]. As the statutes of the organization include, the association has 'no pecunuary, self-help or commercial purpose' [21]. Running expenses and further development of the standard do however require a certain liquidity. This is ensured by a membership program surrounding the specification [22]. 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. This is done 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. [23]

Beside 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 (Michael Schneider)

Risc-V and ARMv8 are both RISC based architectures but still different in complexity e.g amount of different implementations and the way to extend the ISA, the different instruction formats and assembly operations. 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 [7, page 3f]. Further details about the extensions are in chapter III-D.

ARM instead defines the ARMv8 architecture in a completely different way. The ARMv8 already supports many more extensions in the basic version, also called v8.0 [11, page 60]. Further extensions are available in later versions, as explained in the chapter III-D.

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 combines a basic ISA (RV32I or RV64I) plus the different selected standard extensions IMAFD, which are explained in chapter III-D. [7, page 3ff]

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. While A32 uses fixed 32-bit instruction encodings, T32 is variable, which means it uses both 16-bit and 32-bit instructions. [11, page 38]

To complete the overview in a, for users more abstract point of view, two basic assembler commands (load and store) and the different instruction formats 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. [7, page 18] 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. [24, slide 30ff]

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. [11, page 207 - 218]

The explained load and store instructions are, just like all other instructions defined in a special format. In RISC-V (RV32I) the instructions are summarised to Branch instructions, jump instructions, store instructions, instructions with immediates, arithmetic/logical ops and instructions with upper immediates. In total the six different formats hold 38 different instructions. The arithmetic/logical ops (R-Format) instructions using 3 register inputs, e.g. the add command takes 2 source registers and as a third one, the destination register to store the sum of the 2 source registers. The immediate (I-Format) and the upper immediate (U-Format) work in a similar way. Both add an immediate to the source register, but I-type operations only have an immediate up to 12-bit – which is also supported by load instructions – and U-type operations have an immediate up to 20-bit. Because of the higher amount of immediate bits, the upper immediates only contain 1 register. Branching (B-Format) instructions – mostly used for loops – as well as store (S-Format) instructions take to registers and both do not write values to the register file. As last (J-Format) jump instructions are used to jump to a specific address, to define 32-bit address a immediate together with a destination register is used. [25, slide 10 -64]

In the ARMv8 ISA (A32) the instructions are structured in a different way, as well as the base version has much more different instructions, e.g. already more than 40 different load and store instructions. ARM contains, data-processing and miscellaneous instructions, Load/Store Word (immediate, literal), Load/Store Word (register), media, branch, system register access and unconditional instructions. Each instruction is encoded by a 32 bit word containing a condition at the highest positions bits[31:28], followed by an op0 (3-bit) field to define the type of instruction (explained above). For every one of the 7 formats further definitions are done and deeper lets say subformats make more specifications for the structure. There are the clear definitions about source and destination register made as well as the size of the immediate is defined. [11, page 4218 - 4278].

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 [26]. It consists of a number of programs frequently used in embedded applications, such as CRC, signal filtering, AES and QR code reading [26]. 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 III-B) using GCC 7, the code for RISC-V is approximately 11% larger than the code for ARM [27]. 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 [26].

In [27] an extension for RISC-V is introduced, called HCC, that contains a number of instructions 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. [27]

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 [28, 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 [29, 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 [29, 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 [29, slide 40]. These instructions may however be split into multiple micro-operations on an ARM CPU via a process called micro-op generation [29, 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 [29, 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 [29, slide 38].

D. Extensibility (Michael Schneider)

Extensibility is very important to build a tailored CPU to the specified needs. Therefore ISAs like RISC-V and ARMv8 define extensions in different ways. While RISC-V defines the basic integer instructions as the only mandatory instruction set [7, page 3f], ARMv8 defines the more extended ARMv8.0 as the basic version [11, page 60].

RISC-V supports in addition to the integer instructions different extensions. To mark the implemented extensions, representable letter are used. 13 extensions are provided and everyone can add new features on top of the ISA, as followes. The M extension for integer multiplication and division, A extension for atomic instructions (needs additional hardware to be emulated), C extension for compressed instructions, B extension for bit manipulations, J extension to understand dynamically translated languages, T extension to work with transactional memory, P extension for Packed-SIMD instructions and the N extension to support User-Level interrupts. To work with further number formats (floats, vectors) additional to standard integers, V extension (vectors), F extension (single-precision floating-point), D extension (double-precision floating-point), Q extension (quad-precision floating-point) and L extension (decimal floating-point) are defined. The implemented extensions further give the name to the ISA, e.g. if single floating-points and vectors extend a RV32I together with the integer multiplication, the ISA will be named RV32IMFV. [7, page 31-52, 71-103]

ARM provides extensions in a different way. The ISA has a base version which contains almost all extensions of RISC-V, called ARMv8.0. The ARMv8.0 is already able to work with vectors and floating-points and it is also able to multiply and divide integers. The higher versions of ARMv8 (ARMv8.1 up to ARMv8.6) add further additional requirements and architectural features. Only the version ARMv8.4 adds only architectural features and no further requirements. [11, page 60ff]

Additional to the different versions of the ISA, both provide to create custom extensions. To create a RISC-V custom extension two steps are important: defining the instructions and include the instruction to the application [30, page 16 - 21]. To realise these steps different tools can be used, for example the OVP APIs, which are helpful to simulate the extended processor model for simpler tests and creation of a custom instruction [31]. To define the instructions, the first step is the decoding of the instruction, to define the fixed fields for instruction class and the fields for source and result register. Later on the defined decoding will be used to decode the instructions from the PC. The instruction behaviour can be done in two different ways, either by simply using a C algorithm function or by using the VMI Morph Time Functions which is the recommended approach. Finally the custom instructions can be used in C applications after they are added to the processor model. [30, page 16 - 21]

ARM added also the possibility to create customized extensions. So far this is only supported for Cortex-M33, but in 2021 ARM wants to add support for the new Cortex-M55 as well. Accelerators can be categorized in the 3 different types. Memory mapped accelerators (connected to memory bus), coprocessor interface (allows to build closely coupled accelerators) and tightly to the CPU coupled accelerators. While decoupled memory mapped accelerators run parallel and unaffected from the CPU, coprocessor and tightly coupled accelerators interact with the CPU and further tightly coupled accelerators are not able to run in parallel with the CPU. To build one of the 3 types of custom instruction only two steps are mandatory to do:

1) "Providing a configuration file that lists the regions you want to use for adding your own custom instructions." [32, page 4]
2) "Building the datapath for your own custom instructions and integrating it into the configuration space." [32, page 4]

To finally decode the instruction and control the data path, the logic is automatically configured. [32, page 2ff]

E. Ecosystem (Florian Henneke)

Getting access to the ISA is only the first step in producing and offering self designed silicon. The following steps can be broken down roughly into the subsequent points [33]:

1) Designing the CPU Core.
2) Checking for compliance to the ISA [34].
3) Testing the core for formal correctness [34].
4) Testing for function with random and edge case data [34].
5) Finding a factory partner for production.
6) Set up at least a software toolchain for creating and comiling applications on your hardware. It is also advantageous, if the toolchain contains a hardware abstraction layer and is sufficiently tested.

ARM makes this process significantly easier: They already supply a completed CPU Core, which is compliant to their ISAs and tested formally as well as in function. It is also proven many times in existing hardware. The ARM package also includes a complete software toolchain which enables chip designers to create chips according to their own specification within the ARM environment. Having a rather firm ecosystem, they are also able to provide a hardware abstracion layer as well as a compiler toolchain [35]. Lastly they offer connections to chip foundries who already have experience in manufacturing ARM based silicon [33]. A good ecosystem for the end user, consisting of well tested compiler toolchains [36] and operating systems, also exists [37] [38].

What RISC-V currently offers is completely different. The only thing obtained from RISC-V International is the ISA definition. This does not mean, that the chip designer must go through the aforementioned list on their own. With a supportive community for RISC-V one is able to set one's own starting point in the design process. For example one could start at a finished, probably already compliance checked and tested core [39] and adapt it according to one's use cases.

The core implementation also dictates the choice of design toolchain. If one wants to check their extended design, there are already some verification and testing tools available [34]. All those cores and tools can however be published under another license than the ISA specification. This means the publisher can also limit the freedom that the chip designer has. With a production ready core, one must find a silicon production partner for himself. At last, one has to provide an appropriate toolchain for one's chip. This is easy, if the core provider already provides one [40] that is suitable for one's changed core or can be built with community driven toolchain generators [41]. In case of operating systems, the end user is of course able to compile embedded operating systems like Zephyr [42] or FreeRTOS [43], desktop operating systems however are not yet supported. Nevertheless it is proven, that Linux is able to run on RISC-V based processors [44].

IV. Discussion

A. Performance (Alexander Schmid)

32-bit ARM with the T32 extension features a lower code size than RV32IMC in two popular benchmarks [27]. This poses a significant advantage on embedded microcontrollers with a small program memory. Currently, this disadvantage can be mitigated on RISC-V with custom ISA extensions. However these pose the risk of excessive fragmentation with different extensions, if they are not standardized. This would make it more difficult for compiler developers to support and optimize for all of the custom RISC-V extensions.

In the case of 64-bit ARM and RV64G(C), while RISC-V offers a significantly lower code size, it also has a larger dynamic instruction count. The authors of [29] find no difference in cycle count between the two ISAs, after applying a model that takes into account both micro-op generation as well as macro-op fusion. However, this is based on assumptions, that are not verified against actual ARM CPUs, about the number of micro-ops that certain ARM instructions break into [29, slide 40].

A more accurate comparison of the influence of the ISA on a CPUs cycle count could be gained by simulating two CPUs with as many of the same components as possible, adapted for each of the two ISAs. This is done for ARM and x86-64 in [10].

Ultimately, the influence of ISA-specific characteristics plays a lesser role for a CPUs's performance than the optimization performed for a specific implementation of that ISA. As such, the largest factor determining which ISA will feature faster processors is simply the amount of investment large companies will put into an ISA [2]. This can be studied by empirically comparing common CPUs of both ISAs, as done in [2] for ARM and x86-64, as soon as enough RISC-V CPUs are on the market.

B. Extensibility (Michael Schneider)

To discuss the extensibility of the architectures, two different ways to extend ISAs need to be considered, the use of defined standard extensions and the use of custom extensions.

The RISC-V architecture has only one mandatory component (integer instruction set) [7, page 3f], while the basic ARM version (ARMv8.0) supports already many extension [11, page 8120]. This results for RISC-V in an only as complex as needed CPU, because only the mandatory instruction sets have to be implemented [7, page 3f]. The ARM version instead, which includes a big amount of instructions, results in an maybe more complex CPU (unnecessary implemented instructions). Related to the realisation of extensions, as shown in chapter III-D, the ARM ISA forces to implement instructions and features which are maybe not mandatory for the use of the CPU, but could be usefull in the future.

In form of custom extensions it depends on the personal preferences. While ARM is very strict and a official white paper [32] defines the way to do, RISC-V doesn't define a official way and there are a lot of tool triggered projections, like the one from Imperas [30], are given, but the loose definition of RISC-V makes it a bit more complex to understand the process.

C. Complexity (Michael Schneider)

The complexity is influenced by different aspects, as shown in chapter III-B. ARM has much more instructions compared to RISC-V (see chapter III-B). Due to this in ARM it is more difficult to understand the differences between every single instruction. RISC-V instead has a more limited amount of instructions that is why the difference between each instruction is more transparent.

The differences in complexity of both ISAs are also related to the base versions with the corresponding extensions (shown in chapter III-D) and the amount of instructions and formats (elaborated in III-B). ARM has for the higher amount of instructions also more different instruction formats, which are a bit variable due to the different subversions, as the RISC-V format (see chapter III-B).

Both ISAs define a way for 32-bit and 64-bit. While ARM defines with T32 a version, which works with 16-bit and 32-bit [11, page 38], RISC-V defines a version for 128-bit [7, page 3ff]. As the usage of the format of the ISA depends on the usage, also the definition, if it is better to define a 128-bit version or a version for 16-bit and 32-bit depends on the usage of the ISA.

D. Business Model and Ecosystem (Florian Henneke)

Because the Business Model and the Ecosystem are stronly connected from a chip developers perspective, while deciding which architecture to use, they will be discussed together in the following section.

In summary, there are three main points to consider when choosing a architecture from an economical point of view: the price, how much work you have to put in and how much revenue can be expected.

Starting with the financial aspect, the costs can theoretically start at zero for both platforms. However considering, that one is not allowed to commercially produce ARM based chips with the lowest tier licenses and has to pay royalties [14], RISC-V

gets, beacuse of beeing totally free, an advantage at the fixed expenses.

Anyhow most of the expenses will not originate from the license, but will develop during the development phase (III-E) in form of the amount of work, that has to be done and in other licenses concerning development and testing software. ARM already includes many of these upcoming costs in their licenses by offering an already designed and tested core as well as related software [33]. In case of RISC-V the semiconductor company has to choose and evaluate their options at every step, which definitely creates a lot of work, before even starting with the main design, but can also result in less work down the development path or cost as low as zero in terms of software licensing.

Finally developing and offering an own ecosystem for the end user can also create additinal costs, but defintely has a big influence on the revenue stream and the acceptance in the end user community. For a chip with little additions to the provided core, this step is already done by ARM. Looking at RISC-V, this depends mon the choosen core. So here again, the RISC-V route can be as optimal as the ARM way, but can also be worse. Eventually the chip use case has also to be considered. While ARM already supports many operting systems, RISC-V has not yet a really usable operating system environment (III-E).

Ultimately the decision depends on who you are (Can you choose low tier ARM licenses and achieve your goal?), what you want to achieve (Is there a future interest in developing more chips or is the effort only for one chip), what resources you have (Can you afford higher development costs for bigger future revenue?) and what risk you can take (Do you want to take the risk of going with a new platform or do you want to take the proven way?).

V. Conclusion and Outlook

A. Summary of results

B. Interpretation of results

C. Future directions

The results of the chapter III and the resulting discussion IV try to compare the advantages and disadvantages of both, the RISC-V and the ARM architecture. Most time the advantages and disadvantages depend on different influences, like personal preferences, resources and so on. To clearly define the benefits and drawbacks of the ISAs it would be worth to analyse the surrounding conditions to weigh the results to them. A further question would be, if there are some regions, where RISC-V is clearly the better option compared to ARM and if it is possible for RISC-V to gain a nameable market share of ARM.

VI. Overview of Literature

Alexander Schmid [10] [11] [45] [46] [26] [27] [47] [28] [48]

Florian Henneke [28] [49] [45] [50] [37] [51] [43] [37]

Michael Schneider [7] [11] [25] [31] [6] [4] [24] [5] [30] [32]

References

[1] J. Stokes, *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture*. No Starch Press, 2006.

[2] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 1–12.

[3] A. George, "An overview of RISC vs. CISC," in *Proceedings The Twenty-Second Southeastern Symposium on System Theory*. Los Alamitos, CA, USA: IEEE Computer Society, mar 1990, pp. 436,437,438. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SSST.1990.138185

[4] M. Levy, "The History of The ARM Architecture: From Inception to IPO," 2005.

[5] A. Holdings, "Q3 2017 roadshow slides," 2018. [Online]. Available: https://www.arm.com/-/media/global/company/investors/PDFs/Arm_SB_Q3_2017_Roadshow_Slides_Final.pdf

[6] . R.-V. International, "History of risc-v," 2020. [Online]. Available: https://riscv.org/about/history/

[7] Waterman et al., *The RISC-V Instruction Set Manual Volume I: User-Level ISA*, 2.2 ed., May 2017.

[8] G. Tang and I. W. Brown, "Intel and the x86 Architecture: A Legal Perspective," Harvard Journal of Law & Technology Digest, 2011, accessed on 2020-11-02. [Online]. Available: https://jolt.law.harvard.edu/digest/intel-and-the-x86-architecture-a-legal-perspective

[9] D. A. Patterson and D. R. Ditzel, "The Case for the Reduced Instruction Set Computer," *SIGARCH Comput. Archit. News*, vol. 8, no. 6, p. 25–33, Oct. 1980. [Online]. Available: https://doi.org/10.1145/641914.641917

[10] A. Akram, "A Study on the Impact of Instruction Set Architectures on Processor's Performance," Ph.D. dissertation, Western Michigan University, 08 2017.

[11] *Arm® Architecture Reference Manual. Armv8, for Armv8-A architecture profile*, Issue F.c ed., ARM, Jul. 2020.

[12] ARM, *ARM® Cortex®-A53 MPCore Processor Cryptography Extension*, f ed., 12 2015.

[13] ——, *ARM Architecture Reference Manual Thumb-2 Supplement*, Dec. 2005.

[14] ——, "How licensing works," accessed on 2020-11-09. [Online]. Available: https://www.arm.com/why-arm/how-licensing-works

[15] C. Demerjian, "A long look at how ARM licenses chips," Aug. 2013, accessed on 2020-11-09. [Online]. Available: https://semiaccurate.com/2013/08/07/a-long-look-at-how-arm-licenses-chips/

[16] ——, "How ARM licenses it's IP for production," Aug. 2013, accessed on 2020-11-09. [Online]. Available: https://semiaccurate.com/2013/08/08/how-arm-licenses-its-ip-for-production/

[17] Apple Inc., "Small chip. Giant leap." Nov. 2020, accessed on 2020-11-12. [Online]. Available: https://www.apple.com/mac/m1/

[18] Waterman et al., *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*, May 2017.

[19] Creative Commons, "Attribution 4.0 International," accessed on 2020-11-15. [Online]. Available: https://creativecommons.org/licenses/by/4.0/

[20] RISC-V International, "About RISC-V," accessed on 2020-11-15. [Online]. Available: https://riscv.org/about/

[21] ——, "Articles of association of RISC-V International Association," Jan. 2020, accessed on 2020-11-15. [Online]. Available: https://riscv.org/wp-content/uploads/2020/04/Certified-copy-of-articles-of-RISC-V-International-Association.pdf

[22] ——, "Membership," accessed on 2020-11-15. [Online]. Available: https://riscv.org/membership/

[23] ——, "RISC-V International Association," 2020, accessed on 2020-11-15. [Online]. Available: https://riscv.org/wp-content/uploads/2020/03/RISC-V-International-Regulations-03-11-2020.pdf

[24] N. Weaver, "Introduction to Assembly Language and RISC-V Instruction Set Architecture," 2019, accessed on 2020-11-20. [Online]. Available: https://inst.eecs.berkeley.edu/~cs61c/sp19/lectures/lec05.pdf

[25] S. Ho, "RISC-V Instruction Formats." [Online]. Available: https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture7.pdf

[26] D. Patterson, J. Bennett, P. Dabbelt, C. Garlati, and O. Shinaar, "Initial Evaluation of Multiple RISC ISAs using the Embench™ Benchmark Suite," Dec. 2019, accessed on 2020-10-24. [Online]. Available: https://riscv.org//wp-content/uploads/2019/12/12.10-12.50a-Code-Size-of-RISC-V-versus-ARM-using-the-Embench%E2%84%A2-0.5-Benchmark-Suite-What-is-the-Cost-of-ISA-Simplicity.pdf

[27] M. Perotti, P. D. Schiavone, G. Tagliavini, D. Rossi, T. Kurd, M. Hill, L. Yingying, and L. Benini, "HW/SW Approaches for RISC-V Code Size Reduction," in *Workshop on Computer Architecture Research with RISC-V. CARRV 2020*, 2020.

[28] A. S. Waterman, "Design of the RISC-V Instruction Set Architecture," Ph.D. dissertation, University of California, Berkeley, 2016.

[29] C. Celio, K. Asanovic, and D. Patterson, "ISA Shootout: Comparing RISC-V, ARM, and x86 on SPECInt 2006," UC Berkeley, Tech. Rep., 2016.

[30] I. S. Limited, "Imperas riscv custom instruction flow application note," Tech. Rep., 2019. [Online]. Available: https://www.ovpworld.org/documents/Imperas_RISCV_Custom_Instruction_Flow_Application_Note.pdf

[31] I. Software, "Ovp documentation." [Online]. Available: https://www.ovpworld.org/documentation

[32] F. P. Lauranne Choquin, "ARM Custom Instructions: Enabling Innovation and Freater Flexibility on ARM," ARM Ltd., Tech. Rep., 2020.

[33] ARM, "Designing a custom SoC: 9 next steps after downloading your Arm IP," 2019, accessed on 2020-12-04. [Online]. Available: https://community.arm.com/developer/ip-products/processors/designstart/b/blog/posts/designing-a-custom-soc-9-next-steps-after-downloading-your-arm-ip

[34] M. Bartley, "RISC-V Design Verification Strategy," 2020, accessed on 2020-12-04. [Online]. Available: https://verificationacademy.com/verification-horizons/november-2020-volume-16-issue-3/risc-v-design-verification-strategy

[35] ARM, "CMSIS," accessed on 2020-12-04. [Online]. Available: https://developer.arm.com/tools-and-software/embedded/cmsis

[36] ——, "GNU Arm Embedded Toolchain," accessed on 2020-12-04. [Online]. Available: https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

[37] Microsoft, "Windows 10 on ARM," 2020, accessed on 2020-11-27. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/porting/apps-on-arm

[38] R. Hat, "Red Hat Enterprise Linux - Get started - Additional Architectures," accessed on 2020-12-04. [Online]. Available: https://access.redhat.com/products/red-hat-enterprise-linux/#addl-arch

[39] RISC-V International, "RISC-V Exchange: Cores and SoCs," accessed on 2020-12-04. [Online]. Available: https://riscv.org/exchange/cores-socs/

[40] SiFive, "SiFive Freedom RISC-V Tools for Embedded Development," accessed on 2020-12-04. [Online]. Available: https://github.com/sifive/freedom-tools

[41] crosstool-NG, "crosstool-NG," accessed on 2020-12-04. [Online]. Available: https://crosstool-ng.github.io/

[42] Linux Foundation, "Zephyr Project," accessed on 2020-12-04. [Online]. Available: https://www.zephyrproject.org/

[43] Amazon Web Services, "Using FreeRTOS on RISC-V Microcontrollers," accessed on 2020-10-28. [Online]. Available: https://www.freertos.org/Using-FreeRTOS-on-RISC-V.html

[44] SiFive, "SiFive Freedom Unleashed SDK," accessed on 2020-12-04. [Online]. Available: https://github.com/sifive/freedom-u-sdk

[45] K. Asanović and D. A. Patterson, "Instruction Sets Should Be Free: The Case For RISC-V," University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, 2014.

[46] Heui Lee, P. Beckett, and B. Appelbe, "High-performance extendable instruction set computing," in *Proceedings 6th Australasian Computer Systems Architecture Conference. ACSAC 2001*, 2001, pp. 89–94.

[47] C. Shore, *ARMv8-A Architecture Overview*, ARM Limited, Sep. 2015.

[48] X. H. Xu, S. R. Jones, and C. T. Clarke, "ARM/THUMB code compression for embedded systems," in *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (Cat. No.03CH37442)*, 2003, pp. 32–35.

[49] L. Ryzhyk, "The ARM Architecture," Tech. Rep., 2006.

[50] S. Furber, *ARM System-on-Chip Architecture*, 2000, no. a.

[51] Greenwaves Technologies, "Arm® Mbed™ OS Porting Manual for GAP8 ," accessed on 2020-10-28. [Online]. Available: https://greenwaves-technologies.com/manuals/BUILD/MBED-OS/html/index.html

[52] D. Patterson, "50 Years of computer architecture: From the mainframe CPU to the domain-specific TPU and the open RISC-V instruction set," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 27–31.

[53] J. Hennessy, *Computer architecture : a quantitative approach*. Waltham, MA: Morgan Kaufmann, 2012.

[54] R. D. Vladimir Herdt, Daniel Große, *Enhanced virtual prototyping featuring risc-v case studies*. S.l: SPRINGER NATURE, 2020.

[55] M. D. H. of Wisconsin-Madison; Dave Christie; David Patterson; Joshua J. Yi; Derek Chiou; Resit Sendag, "Proprietary versus Open Instruction Sets," *IEEE Micro*, 2016.

[56] S. Higginbotham, "The Rise of RISC," *IEEE Spectrum*, 2018.

[57] R. Dirvin, "The arm ecosystem: More than just an ecosystem, it's oxygen for soc design teams," April 2019, accessed on 2020-10-25. [Online]. Available: https://www.arm.com/company/news/2019/04/the-arm-ecosystem-more-than-just-an-ecosystem

[58] Z. Bandic, IEEE Computing Society, March 2019, accessed on 2020-10-25. [Online]. Available: http://www.hsafoundation.com/the-inherent-freedom-of-heterogeneous-systems/