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

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.

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.

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 and complexity 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.

B. RISC

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 instructions allowing access to memory, and a small number of registers, mainly to make programming in assembly easier [1].

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 [1].

The RISC approach was initially favored due to improved performance and power efficiency [1].

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 [1].

C. ARM

History, ... closed source (pay for licenses), exists longer -¿ more implementations, weiter entwickelt, ...

D. RISC-V

geschichte, ... open source, leichter und ggf kostengünstiger zu erstellen, ...

III. Concept and Methods (Initial section written 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 [3].

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. [4]

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] [5]

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 [6, page 52], Advanced Encryption Standard (AES) extensions that optimize cryptography [7] or ISA-extensions with shorter instructions for applications that are constrained in program memory. [8] 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 (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. [9] 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 'FPGA' license which is free and includes a Advanced Encryption Standard (FPGA) 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 [10] [11] 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. [12] 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. [13] [14] 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. [15] 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. [16] As the statutes of the organization include, the association has 'no pecunuary, self-help or commercial purpose' [17]. Running expenses and further development of the standard do however require a certain liquidity. This is ensured by a membership program surrounding the specification. [18] 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. [19]

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 (Michael Schneider)

Risc-V and ARMv8 are both 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: The extension concept of the RISC-V architecture makes it only as complex as necessary, because the ISA can be tailored to the specific needs. 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. [13] 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. Further extensions are available in later versions, as explained in the chapter III-D. [6]

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, explained in III-D). [20]

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. This is good to optimize your code, because a common instruction selection rule says, if a 16-bit encoding and a 32-bit encoding are available, the 16-bit is the one you should use, to optimize the code density. [6]

3) registers and access: 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. [13] 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. [21]

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. [6]

Additional to the explained load and store instructions both architectures have many other instructions, viewed are both times the clear 32-bit instructions RV32I for RISC-V and A32 for ARMv8. In RISC-V the instructions are summarised to Branch instructions, jump instructions, instructions with immediates, arithmetic/logical ops, instructions with upper immediates. The arithmetic/logical ops are called R-Format instructions and RISC-V has 10 of them. For the immediates (I-Format) RISC-V has 9 plus the 5 load instructions from above and EBREAK, ECALL as environment Call and Breakpoints calls. The Store instructions are limited to 3 and the branch instructions to 6. 2 Upper Immediate instructions and only 1 jump instruction complete the RISC-V instructions in RV32I. [13] In the ARMv8 ISA the instructions are summarised in a different way, because there are already more instruction sets in the base version. The branch instructions (10 - 5 for conditional and 5 for unconditional branches) is the only format, which is the same as in RISC-V and already this is bigger in ARM. Further ARMv8 has data-processing instructions (18 standard instructions, 10 shift instructions, 48 multiply instructions, 8 saturating instructions, 14 packing and unpacking instructions, 2 divide instructions and 13 miscellaneous data-processing instructions). PSTATE and banked register access instructions (5), 22 miscellaneous instructions

and 20 instructions to generate and handle exceptions. To complete the ARMv8 instructions the store/load instructions are need to be counted as well. In ARMv8 there are 40 different instructions to load and store. [6]

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 [22]. It consists of a number of programs frequently used in embedded applications, such as CRC, signal filtering, AES and QR code reading [22]. 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 [23]. 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 [22].

In [23] 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. [23]

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 [20, 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 [24, 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 [24, 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 [24, slide 40]. These instructions may however be split into multiple micro-operations on an ARM CPU via a process called micro-op generation [24, 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 [24, 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 [24, slide 38].

D. Extensibility (Michael Schneider)

Extensibility is very important if you want to build a new CPU, so you can build it up completely the way you need it and integrate the necessary features for it. To support this, ISAs like RISC-V and ARMv8 defines the extensions in complete different ways. While RISC-V defines the basic integer instructions as only mandatory instruction set [13], ARMv8 defines the more extended ARMv8.0 as there basic version [6].

RISC-V supports also further instruction sets than only the integer one. The ISAs defines therefore different extensions marked with an representable letter. 13 extensions are provided by the ISA. The M extension for integer multiplication and division, A extension for atomic instructions (need 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 integers, V extension (vectors), F extension (single-precision floating-point), D extension (double-precision floating-point), Q extension (quad-precision floating-point) and D extension (decimal floating-point) are defined. The implemented extensions further define the name of the ISA, e.g. single floating-points and vectors are extending an RV32I together with the integer multiplication, the ISA will be named RV32IMFV. [13]

ARM does the versioning in a different way. They defined 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. [6]

Additional to the different versions of the ISA, both extensions 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 [25]. To realise these steps different tools can be used. For example the OVP APIs to simulate the extended processor model for simpler testing and creating a custom instruction. [26]

To define the instructions, the first step is the decode of the instruction, this is important 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 behavior can be done in 2 different ways either by simply using a C algorithm function or using the VMI Morph Time Functions which is the recommended approach. Finally the custom instructions can be used in C applications after it is added to the processor model. [25]

ARM added also the possiblity to create customized extensions but so far its only supported for Cortex-M33 and in 2021 ARM want to support it in the new Cortex-M55. 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 runs parallel and unaffected from the CPU, coprocessor and tightly coupled accelerators interact with the CPU and 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." [27], S.4
2) "Building the datapath for your own custom instructions and integrating it into the configuration space." [27], S.4

To finally decoding the instruction and control the datapath, the logic is automatically configured. [27]

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? - more instruction formats and more basic instructions and extensions makes it more various applicable. - more different instructions makes the assembler code in a good written way smaller and more performant

B. Advantages of RISC-V

- lower amount of basic instruction sets and only optional extensions makes it less complex to implement. - less instruction formats makes it more easy to build - less different instructions makes the familiarization with programming easier 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 [5] [6] [28] [29] [22] [23] [30] [20] [31]
Florian Henneke [20] [32] [28] [33] [34] [35] [36] [34]
Michael Schneider [13] [6] [1] [20] [37] [38] [39] [40] [41] [42] [43] [21]

References

[1] 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

[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] 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

[4] 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

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

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

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

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

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

[10] 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/

[11] ——, "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/

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

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

[14] ——, *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*, May 2017.

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

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

[17] ——, "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

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

[19] ——, "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

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

[21] 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

[22] 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

[23] 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.

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

[25] 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

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

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

[28] 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.

[29] 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.

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

[31] 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.

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

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

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

[35] 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

[36] 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

[37] 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.

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

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

[40] 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.

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

[42] 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

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