ARMv8 advantages and disadvantages to RISC-V

***Abstract*—text**

***Index Terms*—keyword1, keyowrd2**

I. Introduction

A. Overview

B. Motivation

C. Goal

II. Background

A. Instruction Set Architectures

B. RISC

C. ARM

text

D. RISC-V

text

III. Concept and Methods

A. Business Models

Who develops the CPU cores, how can you get access to them? Who supports chip manufacturers in designing a chip with that CPU core?

B. Complexity

Risc-V and ARMv8 are both Reduced Instruction Set Computer (RISC) based architectures and compared to Complex Instruction Set Computer (CISC) they are much more limited and only writing into registers is supported. RISC machines are having a lot of characteristics, which are not necessary to be completely implemented, but they can lead to a few advantages of RISC compared to CISC, like an simpler control unit and faster decode. [1]
Even various RISC Instruction Set Architectures (ISAs) are different in complexity. To compare those differences, it is important to take a look into the RISC specific features and compare them in flexibility, amount and implementation. So the basic Instruction sets, with corresponding extensions are compared together with the different realisations and the comparison of two basic assembly instructions.

1) Instruction sets: While the ARMv8 is clearly defined in relation to the instruction set, RISC-V is there much more variable. In RISC-V the only necessary Instruction set is the Integer instruction set. Those base integer instructions cannot be redefined, only extended by more optional instruction sets. Additional to operations for the base ISA, completely different extensions are available. Further details are in chapter extensibility. This makes the RISC-V architecture only as complex as necessary, because all the gratuitous instructions are not implemented. [2]
ARM instead is defining the ARMv8 architecture in a completely different way. The ARMv8 supports already much more extensions in the basic version, also called v8.0. To build a more complex CPU the ARMv8 also provides extensions like ARMv8.1, ARMv8.2,up to ARMv8.6 plus a few optional extensions, which are also further compared in the chapter extensibility. Becaue almost all the optional extensions of RISC-V are covered by the basic v8.0, the ARMv8, only from the instructions set of view, is at least as complex as a fully extended RISC-V architecture. [3]

2) Instruction set implementations: The different ISAs are able to implement the explained instructions in different ways. // The RISC-V architecture is able to implement the instruction sets in 3 different versions a 32-bit (RV32I), a 64-bit (RV64I) and a 128-bit version (RV128I). While for the 32-bit and the 64-bit implementations are also multiple subversions available, RV32E, RV32G/RV64G and RV128I, which is the only 128-bit implementation so far. RV32I is a version with only 15 instead of 31 registers. The subversion RV32G/RV64G is less a own version of implementation than a stable release. The RV32G/RV64G is combining a basic ISA (RV32I or RV64I) plus different selected standard extensions (IMAFD). [2]
Also ARMv8 has, as expected, different implementations, as well. RISC-V defines the implementations based on a number followed by the extensions, which are implemented. ARMv8 does it in a different way, because there are all the standard extensions already in the basic version included. ARMv8 provides 3 different implementations A64, A32 and T32. As mentioned 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) The execution states are supporting the A64 instruction set in AARCH64 or A32/T32 in AARCH32. [3]

3) registers and access: To complete the overview in an maybe for users more abstract point of few, the basic assembler commands (load and read) are compared.

To load an value from the RISC-V register lw, lh or lb is called. lw means load word, which loads the complete register, lh means load halfword and loads only 16 bit, lb means load byte. All are built-on the same way and lh and lb can be extended by an U (LHU, LBU) to load them as an unsigned value. All instructions are taking 2 parameters, a temp register to store the value in and an address to load the value from. The address is a pointer to the register together with an offset. In the same way the store instructions sw,sh and sb are working. sw means store word, sh store halfword and sb means store byte. The built-on is the same way as it is for the loading. But the left side of the command is the temp register to take the value from and the second parameter is the register to store the value in (again with an offset). [2] 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 an 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 in a register there are 3 possible ways STR to store complete register, STRB to Store byte, STRH to store halfword. So far its the same as for RISC-V. These basic load and store commands are followed by a few more, to load/store pairs, non temporal pairs, unscaled offsets and much more. Because there is not such an possibility in RISC-V there is no comparison about them. [3]

C. Performance

What are the differences in code size? Can we accurately compare the execution speed of both ISAs?

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. ARM

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

B. 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. Accuracy of results

C. Future directions

D. Literaturverzeichnis

Die Quellen befinden sich in der Datei *biblography.bib*. Meine Quellen sind:l [4], [3], [5], [6] [7], [2], [8] [9], [10].

References

[1] A. D. George, “An overview of risc vs cisc,” 1990.

[2] A. W. L. A. P. Asanović, “The risc-v instruction set manualvolume i: User-level isa,” 2016.

[3] A. Limited, “Arm architecture reference manual,” 2013.

[4] G. David Patterson and U. Berkeley, “50 years of computer architecture,” 2018.

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

[6] 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.

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

[8] IEEE, “The rise of risc,” *IEEE Spectrum*, 2018.

[9] R. Dirvin, “The arm ecosystem: More than just an ecosystem, it’s oxygen for soc design teams,” April 2019. [Online]. Available: https://www.arm.com/company/news/2019/04/the-arm-ecosystem-more-than-just-an-ecosystem

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