RISC V

Stuart Miller

Missouri University of Science & Technology

CpE 6110

Abstract: As today's electronics demand lower and lower power consumption and ever more efficient operation, RISC processors have gained a foothold in the industry. ARM sees widespread use, but industry is often discouraged by its high licensing fee. As Internet-of-Things devices become ever more prominent, no single chip has gained true prominence, thus providing an opportunity for a newcomer to gain a foothold. RISC V has been proposed as a modern alternative to ARM and is rapidly gaining acceptance in the industry. RISC V is both free and open-source meaning that developers are free to use it as they wish. Not only that, but the accessibility of the open source hardware has created many initiatives to expand upon the instruction set and implement custom special-purpose hardware; something that was never possible with proprietary chips like ARM. In this paper, a survey of the current state of RISC V initiatives is summarized and an overview of the technical and performance attributes of the platform is presented.

1 Introduction

1.1 Current Instruction Set Architectures

The market is currently dominated by two incredibly prevalent instruction set architectures (ISAs); Intel's x86 and ARM. Both current generation Windows and Apple computers run exclusively on x86 processors (with a slight exception for Windows 10 which is just recently making inroads into the ARM market with Windows 10 IoT and ARM64). Current generation mobile computing devices such as cell phones are almost exclusively run by 32-bit variants of ARM [1]. ARM Holdings themselves estimate that they have 37% of the global market share of all processors; embedded to server-level [2]. IBM's once-dominant PowerPC architecture hasn't seen widespread consumer use since Apple dropped it from their MacBook product line in 2006 in favor of Intel's x86. While computing is doubtless a growing field, there is a distinct lack of serious alternatives beyond these two major players.

There were, and still are, other existing open source free ISAs in addition to RISC V: OpenRISC and SPARC V8. However, they each have fallacies. SPARC V8 was aimed at server-scale devices and had no place in an embedded world, dominated by mobile, low-power devices [3]. OpenRISC was started in the year 2000 and has seen almost no industry adoption at all, possibly due to perceived technical shortcomings [3].

1.2 RISC V

The RISC V project was begun as an academic endeavor in 2010 at the University of California Berkeley. RISC V was started with the broad goal of "Becoming the standard ISA for all computing devices", being designed for research, education, and commercial use. From the start it has been open source and license free, the first of its kind. The user level specification was set in 2014 and the RISC V Foundation was founded in 2015. The RISC V Foundation states that they are "…a non-profit consortium chartered to standardize, protect, and promote the free and open RISC-V instruction set architecture together with its hardware and software ecosystem for use in all computing devices."

As the base instruction set used by RISC V is now frozen, software developers can be assured that their programs will function on RISC V cores forever. Hardware developers are free to optimize the architecture to their own specific use case. Such a concept is unique to the industry and the RISC V Foundation hopes that this will help ensure their success in the market.

ISA	Base+Ext	Compact Code	Quad FP	Address 32-bit	Address 64-bit	Address 128-bit	Software GCC	Software LLVM	Software Linux	Software QEMU
SPARC V8			✓	✓			✓	✓	✓	✓
OpenRISC				✓	✓		✓	✓	✓	✓
RISC-V	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Figure 1: Comparison of open-source architectures [3]

Figure 1 shows the features of RISC V compared to the two existing open ISAs. While all three are built for the GCC/LLVM toolchain and aimed at supporting versions of the Linux operating system, only RISC V supports multiple bitness, an emphasis on compact, lightweight code, and most importantly accounts for extensions onto the base ISA to encourage further development.

1.3 Why Open Source?

Open source is a relatively new concept in the history of computing. Historically, important projects such as the Linux operating system have been open source. Furthermore, there is no doubt that open source software is extremely popular in today's market. Figure 2 shows the prominence of open source software; essentially every sector of the industry that possesses a dominant commercial player has an open source alternative that is just as dominant.

Field	Standard	Free, open implementation	Proprietary implementation
Networking	Ethernet, TCP/IP	Many	Many
Operating system	Posix	Linux, FreeBSD	Microsoft Windows
Compilers	C	GCC, LLVM	Intel icc, ARMcc
Databases	SQL	MySQL, PostgreSQL	Oracle 12C, Microsoft DB2
Graphics	OpenGL	Mesa3D	Microsoft DirectX
Architecture	None	None	x86, ARM

Figure 2: Free vs. proprietary projects [4]

However, there exist no major open source hardware initiatives. Why is this? Bessen proposes that the primary driver to keep projects closed-source is the "cost of complexity" [5]. The cost required to produce a product that works in a wide variety of situations or applications is extremely high. Additionally, the owner is tasked with maintenance, a task which is estimated to account for as much as 82% of the project cost [5]. This is especially true with something as complex as a hardware architecture and the ISA backing it. The demand to support a great many hardware platforms with differing features and to solve the technical challenges of each is a monumental effort. To combat this, RISC V actively rebukes this trend by offsetting much of the cost of complexity to academic contributions. The academic world already has an established system of receiving grants from corporations or government entities to

fund its work. Corporations are then entitled to use the resulting work free of charge. Many corporations have already bought into this strategy; the board of the RISC V Foundation features names such as Google, Nvidia, Samsung, Qualcomm, and Western Digital just to name a few.

Patterson, one of the creators of RISC V proposes that being open sources saves cost for everyone involved [4]. The ISA developers can rely on community developers to report and propose resolutions to issues and test compatibility. This has the added benefit of multiplying the number of people examining the internal workings of the core, usually something either locked down by patents or hidden away entirely. Finally, allowing the academic world access to the property spurs development even further. While popular in the market, ARM licenses are so expensive that the academic world often avoids extensive hardware research with ARM simply due to cost constraints.

2 Technical Design

2.1 Instruction Set Architecture

At its core; RISC V is merely an instruction set architecture. It has mild constraints on hardware requirements only in as much as the hardware must support the methodology of the ISA. The core ISA contains everything necessary to execute a basic program, but RISC V encompasses several instruction set extensions that provide for added functionality as well.

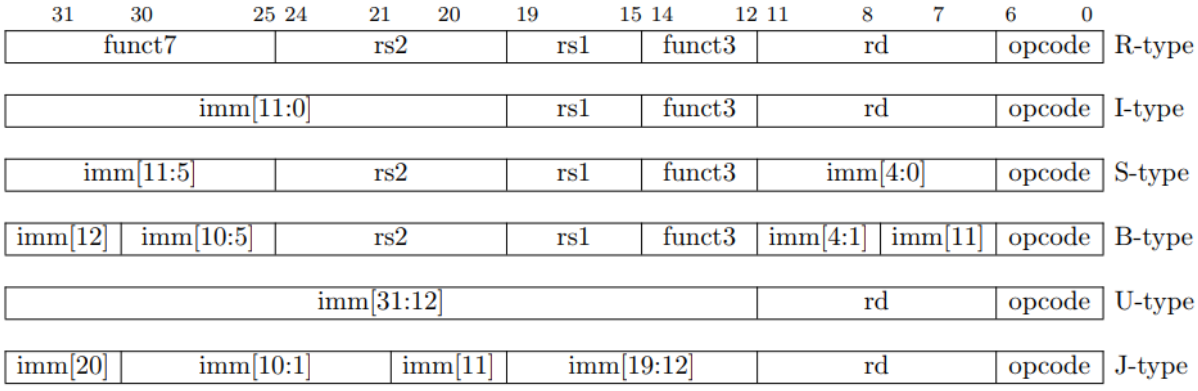

Figure 3: RISC V Instruction Types [6]

2.1.1 Base Instructions

Instruction-encoding in RISC V is quite similar to that of the already well-established MIPS instruction set architecture. Note that RISC V supports both 32-bit and 64-bit instructions (and has an

extension set for 128-bit instructions), but only the encodings for 32-bit are shown here. RISC V's base ISA can be classified as register-to-register (R-Type), register-immediate (I-type), store (S-type), branch (B-type), extended immediate (U-type), and jump (J-type) operations. These encodings are shown in Figure 3.

Of particular note are the S-type instructions. S-type instructions are for store operations used to write register data to memory. The instruction writes the data in *rs2* into the memory address obtained by adding *rs1* and *offset*. The unique format of the S-type instructions splits the immediate value in half. While this does slightly increase the hardware complexity as it requires the bits to be recombined before they can be used, it has the convenient effect of keeping *rs1*, *rs2*, and *rd* to the same bit-alignment across all instructions in which they are used. This is a technique that RISC V uses quite often.

U-type instructions are for extended immediate values. The immediate value is placed into the top twenty bits of the destination register, then the remaining twelve bits are filled with zeroes. These instructions are usually followed by an I-type instruction to fill out the remaining twelve bits and perform the intended operations. Thus, any operations involving immediate values (including load operations with 32-bit memory addresses) require two instructions to complete.

B-type instructions provide branching operations. Conditional branching operations compare the values in *rs1* and *rs2* according the the comparison operator in *funct3*. Once again, the immediate value is broken up to retain the register alignment. Additionally, there is no zero bit in the immediate. This bit is assumed to be 1 meaning that branches must land on an instruction that is a multiple of two. This effectively increased the branch range to ±4KiB. To facilitate this, and to keep the immediate as closely byte-aligned as possible, the 0 bit is replaced with bit 11.

Finally, jump operations are defined in J-type instructions. Once again, the immediate is broken up into parts. Looking back across all the instructions, it can be noted that bit 10 of the immediate in every I-type, S-type, B-type, and J-type always falls in bit 30 of the instruction word. Bits 12-19 fall into the same $funct3$ placement for both U-type and J-type and bits 1-4 fall into the same rd placement for S-type and B-type. These kinds of alignments are representative of the careful thought and planning that went into making RISC V a coherent hardware-centric language from the ground up.

2.1.2 Compressed Instructions

RISC V offers a superset of compressed instructions (known as RISC V Compressed, or RVC) for programmers wishing to reduce code size. Instructions are reduced from their 32-bit versions down to

fit into only 16 bits. This is by no means meant to be a standalone instruction set architecture but meant to be mixed in with standard 32-bit instructions during compilation. A primary feature of RISC systems is an emphasis on reducing hardware complexity, achieved by sacrificing complex instructions for a greater number of simpler ones. RISC V aims to be successful on embedded systems as well, many of which have constraints on code size and as such, the RVC compressed instruction set presents a happy medium.

Format	Meaning	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2		op
CIW	Wide Immediate	funct3	imm				rd′	op
CL	Load	funct3	imm		rs1′	imm	rd′	op
CS	Store	funct3	imm		rs1′	imm	rs2′	op
CB	Branch	funct3	offset		rs1′	offset		op
CJ	Jump	funct3	jump target					op

Figure 4: RVC compressed instruction types [6]

RVC Register Number	000	001	010	011	100	101	110	111
Integer Register Number	x8	x9	x10	x11	x12	x13	x14	x15
Integer Register ABI Name	s0	s1	a0	a1	a2	a3	a4	a5
Floating-Point Register Number	f8	f9	f10	f11	f12	f13	f14	f15
Floating-Point Register ABI Name	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5

Figure 5: RVC register mappings [6]

To help identify these instructions in hardware, all 32-bit instruction have their two least significant bits set to 11, saving bit patterns 00, 01, and 10 for 16-bit instructions. Additionally, several 16-bit instructions only provide 3 bits to identify a source or destination register, restricting the available range. In these situations, RVC only supports addressing the most popular register according to the translation table in Figure 5 (registers 8-15 for the base integer set, with stack pointer values available to load and store operations).

When benchmarked against the SPEC CPU2006 suite, RVC instruction were shown to produce a 23% smaller compiled code size than base RISC V [7]. Interestingly, base RISC-V code was also shown to be only 12% smaller than x86 code; a rather small difference considering one of the core tenants of CISC architectures such as x86 was originally their dramatically reduced code size.

In addition to taking up less space in memory, RVC instructions take up less space in the instruction cache. RVC has been shown to fetch 25%-30% fewer bits, thereby reducing instruction cache misses by 20%-25%, thus having the same benefit as doubling the cache size [7]. Reducing the cache

miss rate greatly improves performance and effectively reduces the rate of power consumption, a major objective for modern embedded RISC V based systems, many of which are mobile and battery-powered.

2.1.3 Vector Instructions

The RISC V ISA includes support for vector instructions. These instructions are strictly optional and only meant to be implemented if the hardware designer so desires.

In order to perform a vector instruction, RISC V defines *vcmaxw* and *vctype* -type registers which must first be configured with the maximum bit width and datatype being operated upon. These, in addition to the vector predicate register, must be configured before each operation. Since large vector operations can require several writes (one each for up to 32 vector registers), RISC V provides several *vcfg* vector configuration instructions to configure groups of registers with the same information at once.

While itself rather underused in academia, RISC V bases its vector architecture on current projects such as Hwacha, a new ISA aimed at purely vector processors [8]. Hwacha (aptly named after the parallel-firing 16th-century Korean rocket launcher) works by pulling vector fetch operations into a separate faster processing loop. This loop is specialized for just vector fetch and able to complete operations much faster, while the main processor is free to continue on.

Again, since RISC V defined no specific hardware architecture, only the instructions such an architecture must implement, hardware designers are free to make any necessary adaptations. In the spirit of the Hwacha project, implementing a vector coprocessor alongside the main RISC V processor is a popular option. Once the vector coprocessor sees a relevant operation issued from the main RISC-V processor (such as, say the group $vcfg$ vector configure instruction, it goes to work setting up the vector registers as necessary. Lee, et al. describe just such an architecture in their design of a dual core RISC V processor with separate vector accelerators [9].

2.1.4 Additional Instruction Sets

Designed from the ground up as an open-source hardware language, rooted in academia, RISC V outlines procedures for and actively encourages researchers to propose their own instruction sets for various use cases.

A prime example of this is Alizadeh and Sharifkhani's design of an H.264 decoding filter [10]. H.264 is an extremely popular standard for encoding/decoding video. It greatly compresses the video and results in a bitrate of up to 50% less than uncompressed video at the cost of increased algorithmic complexity. As such it is aimed at streaming video and video broadcast markets. Their work noticed that

the H.264 algorithm made abnormally heavy use of the absolute value (6%) and clip (13%) functions. Both were implemented as new assembly-level instructions and assigned opcodes in the RISC V extension space. Furthermore, the H.264 algorithm requires analyzing a 4x4 range of surrounding pixels. To accommodate this, they added a 128-bit wide register file to be able to store the results of a row of eight pixels at a time. All in all, they were able to provide an 11% speedup on the H.264 algorithm by adding their custom instructions. This is something that would never have been possible with a proprietary architecture and copyrighted instruction set like ARM.

2.2 Memory Model

RISC V acknowledges the effects of the memory bottleneck and as such, provides inherent support for multiple hardware threads. RISC V refers to these threads as "harts". Each hart has its own program counter and register set. Each hart executes its memory operations sequentially, with no opportunity for reordering. In order to maintain memory consistency between multiple harts accessing memory, RISC V supports the release consistency model.

2.2.1 Weak Consistency

The concept of release consistency extends that of weak consistency. Weak consistency is a more simplistic memory model for multiple instruction, shared memory (MIMD) processors such as those that RISC V is designed for [6]. Weak consistency involves regularly synchronizing memory at the end of critical sections. Within each critical section, each memory access does not need to wait on the previous to complete since these are assured not to conflict. At the end of a critical section, a synchronization takes places and any new memory accesses must wait [11].

The conditions for weak consistency are outlined below.

Conditions for Weak Consistency [12]

A. Before an ordinary load or store access is allowed to perform with respect to any other processor, all previous synchronization accesses must be performed, and
B. Before a synchronization access is allowed to perform with respect to any other processor, all previous ordinary load and store accesses must be performed, and
C. Synchronization accesses are sequentially consistent with respect to one another.

2.2.2 Release Consistency

Release consistency first requires that all memory synchronizations properly set an acquire and release flag on the section of memory they are operation on. An acquire flag is used to gain access to a part of shared memory, while a release flag grants that access. This allows for the relaxation of several ordering constraints. The first is that new load and store operations do not need to wait for the release signal to complete, as the purpose of the release signal is only to represent the end of the previous section of critical accesses.

Release consistency also allows for special access. A process can make a non-synchronizing special request directly to its target memory location. Since this bypasses consistency and reads the memory location directly, there is no risk of reading an outdated local copy.

Finally, condition C requires that special accesses be processor consistent. This means that each processor must see and respond to requests in the order in which they were issued, however these requests do not need to be globally consistent, but any pair of processors must see each other's requests in the same order. Additionally, processor consistency does require cache coherence [13].

Condition 3.1: Conditions for Release Consistency [12]

A. Before an ordinary load or store access is allowed to perform with respect to any other processor, all previous acquire accesses must be performed, and
B. Before a release access is allowed to perform with respect to any other processor, all previous ordinary load and store accesses must be performed, and
C. Special accesses are processor consistent with respect to one another.

2.3 Pipelining

At its core, RISC V is merely an instruction set architecture and does not have a set hardware specification. In some degrees, the specification dictates the hardware, such as in its methods of memory access and the instructions which support them. Additionally, the hardware must support the decoding and execution of the various types of instructions. However, there are several areas of hardware that are open to the designer's interpretation. Pipeline is one of those.

Most RISC hardware implementations make use of a 5-stage pipeline, similar to the classical MIPS pipeline and consisting of fetch, decode, execute, memory, and writeback stages. The open-source nature of RISC-V leaves open the option for designers to implement any hardware to comply with the

instruction set architecture. Many have done just this and there are a great many open source hardware and software projects that demonstrate this.

2.3.1 RISC V Pipeline Example Implementations

One example of a RISC V pipeline implementation is provided by Ripes, a graphical 5-stage processor pipeline simulator and assembly code editor built for illustrating hardware design concepts [14]. This processor design (shown in Figure 6) features a standard 5-stage pipeline and is nearly indistinguishable from MIPS.

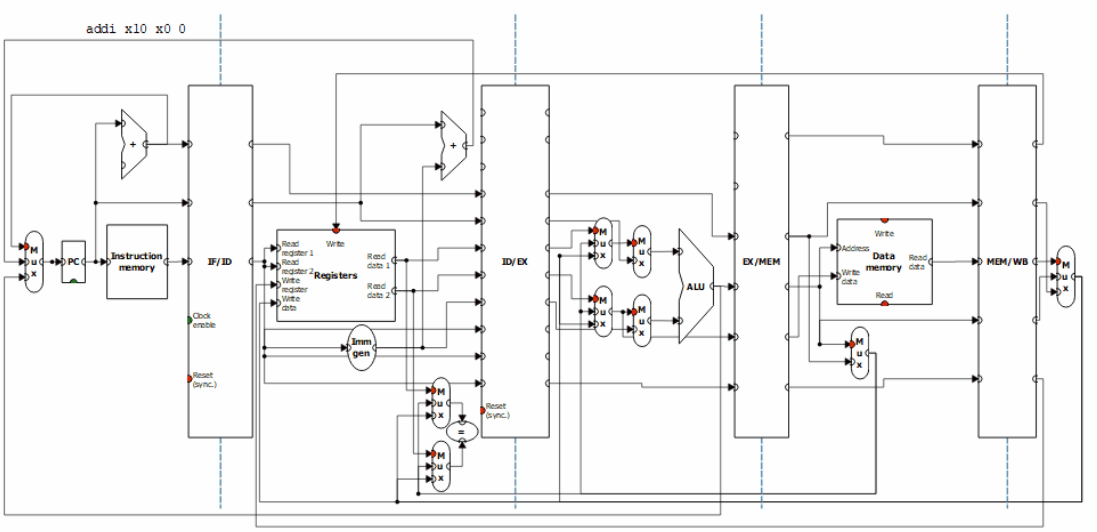

Figure 6: Ripes RISC V 5-Stage Pipeline [14]

Another example, RISCV-Mini was developed as a predecessor to UC-Berkeley's Rocket Chip Generator (shown in Figure 7). This particular implementation features a 3-stage pipeline more akin to that of ARM chips [15].

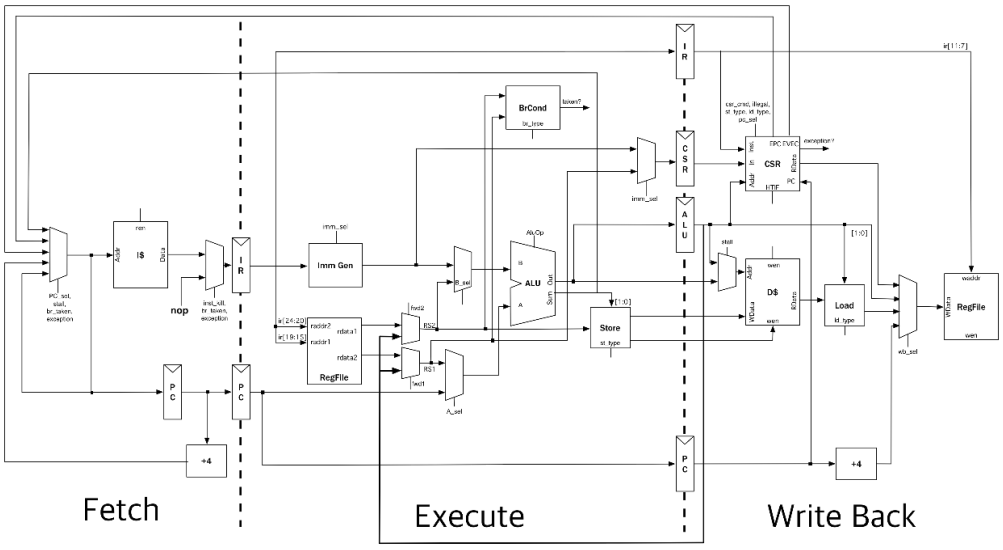

Figure 7: RISC V-Mini 3-stage pipeline [15]

3 Flexible Chip Design

3.1 Rocket Chip Generator

Because RISC V allows for flexible hardware design, a novel application of this is “made-to-order” silicon. A barebones RISC V core that supports the base instruction set can easily be extended with additional hardware to include any of the ISA extensions, such as multiply and divide (M), atomics (A), single-precision (F) and double-precision (D) floating point. A core can choose to support 32-, 64-, or 128-bit instructions. Traditionally, hardware manufacturers can only offer one of several premade options, but the openness and flexibility of RISC V changes that.

One project taking advantage of this is the University of California-Berkeley’s Rocket Chip Generator [16]. Written in Chisel (a Scala-based hardware design language), The Rocket Chip generator customizes 6 different subsystems separately; processor core, processor cache, coprocessor(s), tile, tile link, and peripherals. Tile and TileLink are effective bridges to help the various systems communicate. The core can be one of several designs, such as the Rocket core 5-stage in-order scalar core or the lighter Z-scale Core targeted at embedded applications with a 3-stage, single-issue in-order pipeline.

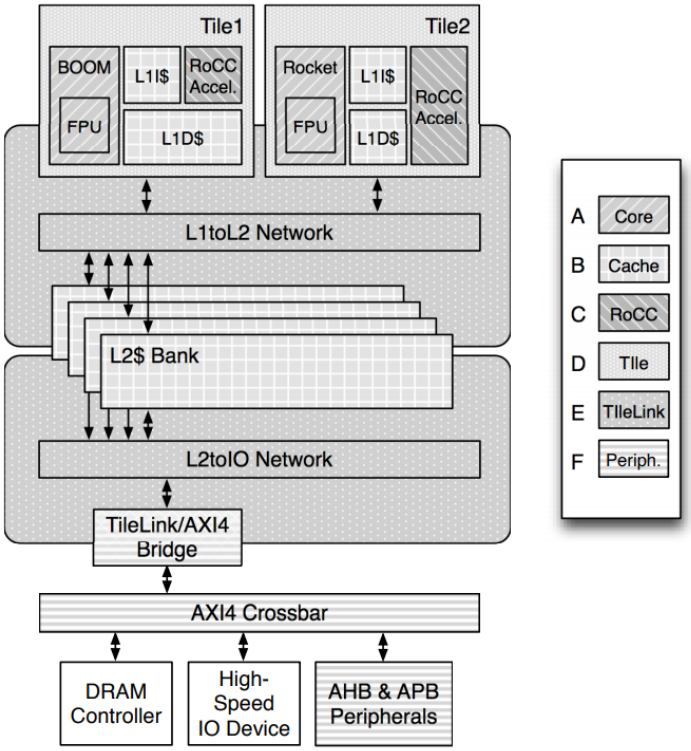

Figure 8: Rocket Chip Generator base design and subsystem layout

Of particular interest is the Berkeley Out-of-Order (BOOM) Core. This is an out-of-order execution processor designed for the Rocket Chip Generator at UC-Berkeley [17]. It features both frontend and backend hardware logic. The frontend supplies instructions as needed, but most importantly, features a branch predictor. This branch predictor works by maintaining a branch instruction to target address cache and a taken/not-taken history. When a branch instruction is presented, if it appears in the cache and shows taken, then its cached address will be fed into the pipeline. The backend issue window holds up to 16 instructions that have been provided. A backend scheduler orders the instructions and chooses the next operable instruction, prioritizing older instructions, and sends it out through one of three windows. These three windows can collectively accommodate up two ALU operations and one memory operation per cycle.

3.2 SiFive

SiFive is a commercial entity founded by Andrew Waterman, one of the creators of RISC V, in order to bring RISC V computers to market. Their first product, the 64-bit, quad-core U54-MC Coreplex is the first RISC V chip capable of running a full version of Linux and is intended for artificial intelligence, machine learning, networking, gateways and smart IoT devices. The chip features a two-level cache with

separate level one regions for instructions and data. SiFive reports the chip scored a 2.75 CoreMark/MHz using the Coremark benchmark [18]. This puts it in line with existing market processors such as Intel Atom N450, Intel Core 2 Duo (Mobile U7600), Microchip's ARM-based PIC32 MX795, and Nvidia Tegra 250, all of which are commercially successful products, albeit on the lower end of the performance spectrum [19].

With this and their other chips, SiFive supports a web-based core designer, allowing developers to order up exactly the core they need. Serving as an example of a corporate entity utilizing the open RISC V intellectual property, SiFive's own products (such as their core designer software) are proprietary and few details about their process have been released. It is safe to assume that it follows much the same methodology as Berkeley's Rocket Chip Designer.

4 Performance

While RISC V is still relatively new to the scene, it is already showing considerable performance gains. While it will doubtless be a while before the standard consumer desktop computer can use a RISC V core, it is far outshining the competition in several specialized applications.

Matthews and Shannon made use of a RISC V system built on an FPGA designed to support Linux shared memory subsystems. Their design implementing the RISC V ISA was able to use 33% less of the available FPGA hardware (slices) and could be clocked 39% faster [20].

Schiavone, et al. performed an analysis of three separate Internet-of-Things(IoT)-oriented RISC V cores [21]. They compared three variants of Shiavone's own Riscy architecture: "Riscy" which implements a 4-stage pipeline, "Zero Riscy" which is area-optimized and implements a two-stage pipeline, and "Micro-Riscy" which removes support for multiplication and division operations and contains the reduced RISC V RVE specification for only 16 registers. When benchmarked with Coremark, the Zero- and Micro-Riscy performed comparably with ARM Cortex-M0/M0+ scoring 2.33 and 2.46 Coremark/MHz, whereas Riscy is comparable with Cortex-M4 scoring 3.40 Coremark/MHz [21]. Such benchmarks are promising for RISC V and stand to prove that it is indeed a viable architecture in industry.

Benchmarking RISC V on applications leaves a lot to be desired. Most of the software written for RISC V are custom embedded solutions that are not directly portable to other architectures and thus, noncomparable. SiFive is still the only company to bring forth a working Linux system running on RISC V. They have yet to publish any definitive benchmarks beyond the initial CoreMark chip score, but it can

reasonably be assumed that their efforts are very much still a work in progress and not yet an immediate threat to existing Linux systems.

5 Conclusions

RISC V remains a stark outlier in the world of hardware development. Still very much a newcomer, RISC V holds very lofty goals in breaking into an entrenched market. Technically, the instruction set architecture is one of the most forward-thinking and comprehensive among its competitors, encompassing a wide range of computing techniques such as vector computing, low-power emphasis, flexibility on floating-point and complex arithmetic hardware, and a forward-thinking memory model. RISC V leaves options up to the hardware designer to implement alternatives to pipeline style, additional coprocessors, or caching strategy as needed.

Relying on both the commercial and academic communities, RISC V has already shown promise in both areas. RISC V has been the subject of numerous research projects and its ability to add on custom instruction set extensions has already been widely utilized. Companies like SiFive are taking advantage of the commercial opportunity. Big-name companies such as Western Digital have signed on with the RISC V Foundation to start producing RISC V cores in their product lines. In fact, Western Digital has open-sourced their RTL hardware design for anyone to critique or use for themselves [22]. This is a testament to the commercial viability of this endeavor and will certainly help ensure its continued success in the market. Perhaps within the next few years, we'll start seeing consumer RISC V based products. Five years down the road, your new cell phone may be running on this very architecture.

6 References

[1] M. P. Singh and M. K. Jain, "Evolution of Processor Architecture in Mobile Phones," *International Journal of Computer Applications,* vol. 90, no. 4, pp. 34-39, March 2014.

[2] R. York, "Embedded Segment Market Update," in *China Technical Seminar Series*, July 2015.

[3] K. Asanović and D. A. Patterson, "Instruction Sets Should Be Free: The Case For RISC-V," August 6, 2014.

[4] M. D. Hill, D. Christie, D. Patterson, J. J. Yi, D. Chiou and R. Sendag, "Proprietary Verses Open Instruction Sets," *IEEE Micro Magazine,* no. July/August, pp. 58-68, 2016.

[5] J. E. Bessen, "Open Source Software: Free Provision of Complex Public Goods," 2005.

[6] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual Volume I: User-Level ISA," SiFive Inc. and CS Division, EECS Department, University of California, Berkeley, Berkeley, 2017.

[7] A. S. Waterman, "Improving Energy Efficiency and Reducing Code Size with," Master's thesis, University of California, Berkeley, 2011.

[8] Y. S. C. O. A. W. A. a. A. K. Lee, "The Hwacha Vector-Fetch Architecture Manual, Version 3.8.1," EECS Department, University of California, Berkeley, December 2015.

[9] Y. W. A. A. R. C. H. S. C. S. V. a. A. K. Lee, "A 45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators," in *40th European Solid-State Circuits Conference (ESSCIRC-40)*, Venice, Italy, September 2014.

[10] M. A. a. M. Sharifkhani, "Extending RISC- V ISA for Accelerating the H.265/HEVC Deblocking Filter," Mashhad, Iran, 2018.

[11] M. Dubois, S. Chrlstoph and F. Briggs, "Memory Access Buffering In Multiprocessors," *SIGARCH Comput. Archit. News,* vol. 14, no. 2, pp. 434-442, 1986.

[12] K. e. a. Gharachorloo, "Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors," *SIGARCH Comput. Archit. News,* vol. 18, no. 2SI, pp. 15-26, 1990.

[13] D. Mosberger, "Memory Consistency Models," Department of Computer Science, The University of Arizona, Tucson, 1993.

[14] M. B. Petersen, "Ripes, a graphical 5-stage processor pipeline simulator and assembly code editor built for the RISC-V instruction set architecture," *GitHub repository,* 2019.

[15] D. Kim , "riscv-mini," *GitHub repository,* 2019.

[16] K. Asanovi, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim and J. Koenig, "The Rocket Chip Generator," University of California at Berkeley, Berkeley, April 15, 2016.

[17] C. Celio, P. Chiu, B. Nikolic, D. Patterson and K. Asanović, "BOOM v2: an open-source out-of-order RISC-V core," University of California at Berkeley, Berkeley, September 26, 2017.

[18] L. Clavin, "SiFive Launches First RISC-V Based CPU Core with Linux Support," SiFive, San Mateo, 2017.

[19] "Scores," EMBC, Embedded Microprocessor Benchmark Consortium, 2019. [Online]. Available: https://www.eembc.org/coremark/scores.php. [Accessed 27 March 2019].

[20] E. Matthews and L. Shannon, "TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, 2017.

[21] P. D. Schiavone and et. al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Thessaloniki, 2017.

[22] W. Digital, "SweRV RISC-V CoreTM from Western Digital," Github Repository, 2019.