RISC-V Reward: Building Out-of-Order Processors in a Computer Architecture Design Course with an Open-Source ISA

Stephen A. Zekany*
University of Michigan
szekany@umich.edu

Jielun Tan*
University of Michigan
jieltan@umich.edu

James A. Connolly
University of Michigan
conjam@umich.edu

Ronald G. Dreslinski
University of Michigan
rdreslin@umich.edu

ABSTRACT

We describe our experience teaching an undergraduate capstone (and elective graduate course) in computer architecture with a semester-long project in which teams of five students design and implement an out-of-order (OoO) pipelined processor core using the open-source RISC-V instruction set. The course content includes OoO scheduling algorithms for instructions to exploit instruction-level parallelism (ILP), example microarchitectures, caching, prefetching, and virtual memory. The labs and projects help students gain proficiency with the SystemVerilog language.

Students use the concepts learned in class to design processors with the goals of achieving correctness and high performance for a suite of test programs representative of different data structures and algorithms. Using RISC-V enables students to validate and benchmark their designs by compiling test programs using GCC with a custom linker. By collaborating as a team, students learn how to write and debug a large code base over the two-month project.

We explain the project content and process in detail, identify the challenges involved for students and the necessary instructor support, and share statistics and student feedback about the project. We have open-sourced our lab and project materials to enable others to teach similar courses.

CCS CONCEPTS

• **Social and professional topics → Computer engineering education**.

KEYWORDS

teaching computer architecture, computer hardware course, undergraduate design project, computer engineering team project

ACM Reference Format:
Stephen A. Zekany, Jielun Tan, James A. Connolly, and Ronald G. Dreslinski. 2021. RISC-V Reward: Building Out-of-Order Processors in a Computer Architecture Design Course with an Open-Source ISA. In *The 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432472

*Both authors contributed equally to this research.

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03.
https://doi.org/10.1145/3408877.3432472

1 INTRODUCTION

Global demand for microprocessors continues to reach new peaks on a yearly basis [2]. Simultaneously, the continued need for high performance in the data center and in mobile and embedded devices is driving increasing complexity and novel designs in computer architecture, especially as Moore's law is slowing [8, 14–16]. In the past thirty years, state-of-the-art CPUs have progressed from in-order pipelined cores built with roughly a million transistors (e.g. Motorola 68040) to multicore, out-of-order (OoO), deeply-pipelined processors with billions of transistors (e.g. Apple A14). Features such as multi-level caching, cache coherence protocols, and simultaneous multi-threading increase performance but also increase the complexity of the design and validation processes.

However, computer architecture education has not kept pace in a world where even tiny embedded devices generally contain pipelined CPUs [1]. We believe that understanding modern computer architecture is not only a necessary grounding for students interested in hardware design careers, but also highly relevant for those learning to build efficient and reliable software. Our courses should not only teach students how processors are built today and discuss the inherent design trade-offs, but give students hands-on experience with an out-of-order pipeline and multi-level caching.

In this paper, we describe our experience updating and teaching an undergraduate capstone design course (also offered as an elective graduate course) with the primary goals of enabling students to develop a deep understanding of how modern processors function and learning to work as a team on a challenging hardware design project. The semester-long course includes background in modern computer architecture, case studies of modern hardware, and a hands-on, design-from-scratch project building an OoO processor that supports a subset of the RISC-V [19] instruction set. The lectures cover a review of basic in-order pipelining through the evolution of various OoO microarchitectures to the present, including multiple ways to exploit instruction-level parallelism with OoO execution (as shown in Figure 1). Labs enable students to develop proficiency with SystemVerilog, an industry-standard hardware description language (HDL). Individual projects enhance students' understanding of various aspects of hardware design while leading to the final team project of OoO processor design.

We updated the course to use the open-source RISC-V instruction set architecture (ISA) (replacing the Alpha ISA [5]) for two reasons: first, to provide students experience with an emerging, interesting, and relevant architecture, and second, to make it easier for students to write test cases, as an ongoing challenge is finding better methods to help students debug their projects. While we provide several dozen test cases hand-written in assembly, with the RISC-V ecosystem students can now write their own C code and compile it using GCC.

The specific contributions of this paper are as follows:

- We describe the necessary structure, schedule, and support to instruct students building synthesizable, out-of-order pipelined processors from scratch.
- We integrate the open-source RISC-V ISA into the design course and describe the advantages of such an approach.
- We survey students on their experience with the updated course design and reflect on the experience as instructors.

The paper is organized as follows: we first share background about the content of the class (Section 3). We describe the final group project in detail (Section 4) and the logistics necessary to support it (Section 5). Finally we share post-project student survey results and discuss the project experience (Section 6).

In the interest of enabling other instructors to offer a similar course, we have collected current versions of our course assignments and starter code into a public GitHub repository [6]. Access to a private repository with instructor solutions and support code is available by contacting the authors.

2 RELATED WORK

Recently, a number of RISC-V implementations of single-cycle and pipelined processors for educational use have been developed. For example, the Davis In-Order (DINO) CPU [11], a Chisel-based five-stage RISC-V pipeline implementing RV32I. DINO includes debugging tools and functional tests, allowing students to understand the underlying architecture instead of implementing helper tools. Similarly, WebRISC-V [9] and the BRISC-V Platform [7] are web-based simulators for RISC-V assembly, with included compilation and debugging tools. These teaching tools are helpful for designing assignments in an introductory class, enabling the instructor to design assignments around a known microarchitecture and system.

Our course project differs from these in that we teach students to design the core of an out-of-order CPU (see Figure 3 and Section 4). Students may have taken a course utilizing these tools as a prerequisite. In fact we have students work with a five-stage RISC-V processor similar to DINO, but the bulk of our class is focused on teaching students to develop a (more complex) out-of-order core, which is the foundation of modern high performance processors.

3 COURSE DESCRIPTION

The class is designed as an advanced undergraduate (and introductory graduate-level) course in computer architecture. The goal is to give students a solid, detailed understanding of how hardware, including the central processor and memory, is designed and interacts with software (see Figure 2), and to teach the numerous trade-offs in design and implementation. A central part of the course is the detailed design of a processor using SystemVerilog, performed in teams of four to five students as a term project.

3.1 Lecture and Lab

The first half of the lecture schedule includes a review of pipelined processors, historic out-of-order implementation, branch prediction [12, 18, 21], and case studies of the Intel P6 [17] and MIPS R10K [20] designs. The second half includes memory and caches [10],

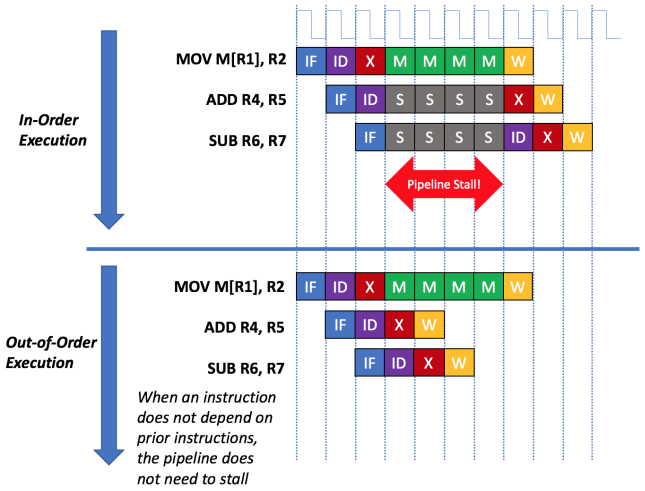

Figure 1: In-order (top) vs out-of-order execution (bottom). An out-of-order pipeline schedules instructions based on the availability of execution units rather than program order. Instructions can finish execution in any order and need not stall unless waiting for a result from a prior instruction.

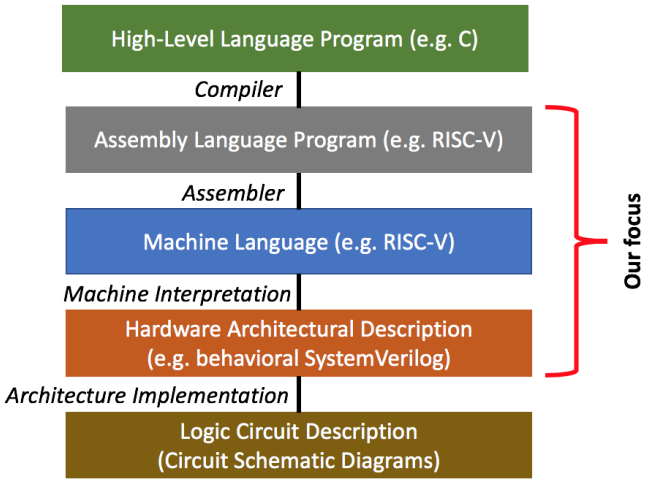

Figure 2: A hierarchy of computing stack from software to hardware. We focus on the interaction of low-level software (assembly and machine code) with the top level of hardware.

power usage, multiprocessors, and finally special topics in computer architecture such as multithreading, new or unusual ISAs, static optimization, and prefetching [13].

There are seven lab sessions, each containing around 45 minutes of instruction. Six labs have assignments, due a week later. The labs teach practical skills such as SystemVerilog, the industry-standard Synopsys toolchain, shell scripting, and version control.

3.2 Individual Projects

Students review basic digital logic and computer architecture by completing three individual projects. These projects develop students' skill at writing combinational and sequential logic, as well as troubleshooting and modifying an existing code base.

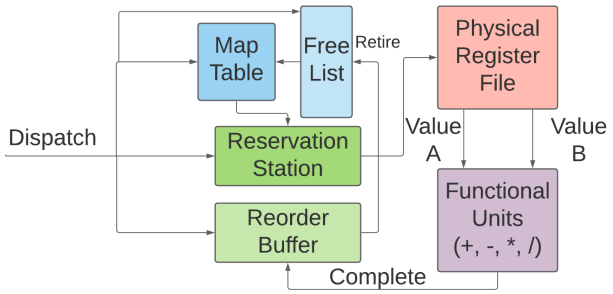

Figure 3: Example implementation of a MIPS R10K pipeline. Architected registers are renamed to a physical register. Instructions are issued from the reservation station to functional units, and results are reported to the reorder buffer.

Easier Features	More Difficult Features
Gshare Branch Predictor	Superscalar (2-N ways)
Set-Associative Cache	Multithreading (SMT)
Return Address Stack	Early Branch Recovery
Stride Prefetcher	Writeback Cache

Table 1: Examples of commonly implemented advanced features. Some features such as a superscalar design must be integrated into the entire pipeline, while smaller features such as a stride prefetcher can be decoupled from the pipeline and often take only a day or two to complete.

3.2.1 Combinational Logic (Priority Selector). This project requires students to write priority selectors using combinational logic and gives an introduction to hierarchical module design by using two-bit priority selectors to build larger priority selectors.

3.2.2 Finite State Machine (Integer Square Root). We give students a sample multiplier with a four-stage pipeline design which computes sums of partial products across multiple clock cycles. Students are asked to reconfigure the multiplier's pipeline stages, benchmark performance, and design a hardware module to compute the integer square root of a 64-bit value.

3.2.3 In-Order Processor. The last individual project is fixing pipeline hazards in a five-stage RV32IM processor. (RV32IM is the integer and multiplication extension of RISC-V, although we do not support divide instructions.) We provide students with a five-stage pipeline and a top-level pipeline integration module, along with a testbench for reading test programs in machine code and printing output as the simulation executes. The pipeline works correctly for code without hazards, but students must add support for instruction, control, and structural hazards.

4 PROCESSOR DESIGN PROJECT

The final project occurs during the last eight weeks of class. Students form groups of four or five to build a correct implementation of a high-performance out-of-order processor which supports RV32IM. Performance is measured by time for each test case to execute, as a function of the clock period and average instructions executed per cycle (IPC). The project is complex enough that having a group of five is helpful even with increased communication overhead.

4.1 Project Details

The project has two components: the baseline and advanced features. The baseline consists of an OoO pipeline following either Intel P6- or MIPS R10K-style microarchitecture, along with instruction and data caches, a branch predictor, and a load-store unit. Figure 3 shows an example of a MIPS R10K style OoO pipeline with dispatch/issue logic and reorder buffer (ROB). Before the project officially begins, instructors hold meetings with each group to discuss which advanced features they propose to implement. Table 1 shows a list of advanced features that groups can add to their design. During the course of the project, we have two mandatory and one optional milestones to measure students' progress and ensure they remain on-track.

4.1.1 Starter Code. Students can freely reuse code from the in-order processor project, of which the most useful modules are the instruction decoder and the arithmetic logic unit (ALU). We also include a simple (low performance) instruction and data cache. To facilitate students' designs, we provide several reference utility modules such as priority selectors and pipelined multipliers, the former of which can be used throughout the project wherever arbitration is needed. The in-order processor also serves as a ground-truth simulator, generating a correct memory map after the program execution and a log of all instructions executed and register writes.

4.1.2 Writing Modules. All groups begin by completing one fully-debugged module at the first milestone. Students are encouraged to code as a whole group for this phase of the project to agree on coding style and practices. We recommend each group construct a high level architecture and decide on the interfaces used throughout the design. The first module serves as a reference point for the interfaces between different units. We test and grade the first module and testbench by introducing small bugs and seeing whether the testbench will catch them (e.g., flipping an "OR" to "AND", or subtracting instead of adding, etc). This puts a strong emphasis on unit testing and incentivizes students to use tools to measure coverage. After the first module, we recommend groups have at least two people working on a module so that if someone is absent later on, there is at least one other person who can debug the module.

4.1.3 Pipeline Integration. By the second milestone meeting, we expect groups to have a functional OoO pipeline core capable of fetching and executing instructions (but no data memory operation yet). We recommend students begin integrating modules a week before the milestone and finish advanced features unrelated to memory. Poor coding style often leads to ambiguous behavior, which may cause bugs. As SystemVerilog simulators tend to be tolerant of ambiguous code, these bugs may be difficult to find post-simulation. To mitigate this, we are meticulous about teaching correct coding style from the beginning of the semester. We suggest all groups reserve one week for the final integration with the memory system.

4.1.4 Debug and Synthesis. We recommend students unit-test, ensure adequate test coverage, and synthesize each of the individual modules as pipeline integration is easier when they do more unit-testing earlier. Students have access to a test suite of more than 35 test cases and can optionally write their own in C or assembly. Students can also contribute a new test case back to the class test

suite. Once a group has a working pipeline, we recommend they log the history of register writes tagged with the corresponding instructions, as this is the only way to verify correctness without being able to modify memory. Also, groups can identify exactly which instruction is incorrectly executed if it performs a register write. This log also serves as a list of committed instructions, which can be compared to the log generated by the in-order core to see if control-flow instructions such as branches and jumps have taken the correct direction. After the second milestone, when groups have a fully functioning pipeline that can modify memory, groups check the memory map their processor produces after program execution with the provided in-order processor.

Logical synthesis translates the HDL into a netlist of standard cells, which can be either macro blocks such as adders or simple logic gates. This is a critical step in hardware design since most designs are written by behavior and not their hardware construction, but ultimately must be capable of being built into actual hardware. A typical problem that students face is using uninitialized values that are not reset or using values without checking validity, resulting in garbage values propagating down the pipeline.

The other aspect of synthesis is performance. Groups are benchmarked by their total run time of the test programs, which is the product of the number of cycles executed and clock period. Synthesis includes static timing of the design and reports critical paths, area, and other potential problems such as circular logic. To reduce the latency of the critical path, groups can add more pipelining in the logical chain, reduce the size of data structures or sometimes completely give up certain functionalities. They learn the important trade-off between cycles per instruction (CPI) vs. clock speed, a problem that every logic designer faces.

4.2 Why RISC-V?

Before RISC-V the final project used the Alpha ISA. The obvious advantages of migrating to RISC-V are relevance and adoption by industry; for example current commercial products running on RISC-V instructions exist. Furthermore, the open-source aspect makes RISC-V available to everyone in the world without having to worry about licensing and export control. For our course, aside from implementing the necessary infrastructure and writing new supporting materials, we encountered no downsides to switching to RISC-V. Students have a chance to apply more knowledge from class in the real world. However the important advantages of using RISC-V in the classroom are more subtle, as we detail below.

4.2.1 Better test programs. RISC-V supports the full GNU toolchain including a compiler and binary utilities such as objdump, which disassembles executables. Previously, we had only assembly-based test programs that were hand-designed mostly to test for corner cases. While these are useful to find bugs, they are poor metrics for measuring performance because they are short and do not reflect real workloads. We have a few assembly test programs that represent common algorithms such as quicksort, but these are still limited in number of instructions. We found that groups became discouraged from implementing certain features if the benefits cannot be reflected within a short execution time, such as advanced branch predictors requiring significant time to warm up.

With GCC, we can develop large test programs with reduced effort and workloads are more representative of real applications. We still create assembly programs for edge cases, but now we also have much larger C programs that serve as better indicators of performance. This also makes it easier for students to write their own tests and more likely to contribute a test case back to the class. Another advantage is the ability to change optimization level during compilations. Before the revision, we had 20 public test cases. Now there are over 35, of which the C programs can be compiled into several versions with different optimization levels to catch bugs.

4.2.2 Simplicity. RISC-V is a clean-sheet design suitable for educational use, and is unencumbered by the need for legacy support. On the other hand, Alpha has not been actively developed for more than 15 years. The students must support the RV32IM instruction set except the system calls (emulation of divide instructions is optional), totaling about 40 instructions. While a subset of any commercial ISA may achieve similar results, it is difficult to provide compiler support for only the subset. Since RISC-V is highly modularized, that support is built-in. Fewer than 10 pages of the ISA documentation supports all the instructions for the class.

4.2.3 Understanding of the computing stack. Although not the main purpose of the class, in the end students can see that the majority of the C library functions can be represented by permutations of about 40 instructions. It is valuable for students to able to draw the connection between software and hardware. Seeing how data structures map to memory and commonly used algorithms execute on hardware they have built themselves is a unique experience.

5 LOGISTICS

5.0.1 Schedule. Our university runs on a semester schedule with 14 weeks of class time (not including breaks and exams). The first half of the class is structured with the three individual projects and regular homework assignments. The OoO design project is the last eight weeks of the class, spanning more than half the semester. By the midterm, the students will have seen all lectures teaching the OoO pipeline concepts. There are several milestones for the final project, and at each the instructors meet with every group individually. We find milestone meetings are enormously helpful for keeping students on-track and helping them resolve problems early. The first milestone, at two weeks, requires every group to turn in one fully debugged module. The second milestone, at 4 weeks, is for students to complete the OoO pipeline except for the load-store unit and the data cache. An optional third milestone is 7-10 days before the project due date.

5.0.2 Grading. The group design project is 35% of students' final grade. The three individual projects are 7%. Typically the midterm and final exam make up 40-48% of the class, with the remaining 7%-15% split between homework and labs. All projects are autograded, and the first two also have a hand-graded portion to reinforce proper coding style, as we find this is important once students begin to work in groups. The grading test suite for the final project includes private test cases on top of the public test cases given to students, along with any student submitted test cases that are of good quality (limit 1 per group). The final project is graded on correctness, performance against the class, and analysis of features.

5.0.3 Staffing. The course staff consists of the instructor and two graduate TAs or one graduate TA and two undergraduate TAs. This is more teaching staff than typically allotted for a class of this size (a ratio of roughly 1:20), but our experience is the TAs are still very busy. In the first half of the class, TAs teach the labs, manage individual projects, and provide support for SystemVerilog and other tools in office hours, including debugging and troubleshooting. In the second half, when students are working on their designs, TAs take on more of an advisory role, since every group's code base is very large and their designs can be quite different.

5.1 CAD Tools and Cost

There are three major tools involved in this class, two of which are for SystemVerilog simulation and design-constraint testing and the third which handles compilation for generating machine code. The first two are proprietary and the last is open-source.

5.1.1 SystemVerilog Simulation and Synthesis. Synopsys VCS (Verilog Compiler Simulator) is used to perform simulations for behavior and structural SystemVerilog. VCS is a common industry tool and while it is proprietary, educational licenses can be obtained via Synopsys' academic programs [3]. Should there be a concern over obtaining licenses, there are free to use, open-source alternatives for SystemVerilog simulation such as Verilator and Cocotb, which support most of the SystemVerilog functionalities while only missing a few features important to industry but not used in this class such as Universal Verification Methodology (UVM) libraries. A third option is to obtain simulation tools from FPGA manufacturers such as Altera and Xilinx, who also have academic programs.

5.1.2 Synthesis. We use Synopsys Design Compiler for SystemVerilog synthesis to build structural netlists (also described in SystemVerilog) of logic gates from behavioral SystemVerilog. This tool can also be obtained via Synopsys' educational programs. Open-source alternatives exists such as Yosys, and academic commercial alternatives are sold by both Altera and Xilinx.

5.1.3 RISC-V GNU Toolchain. The GNU suite of toolchains are used for compilation and linking. For the most recent semester we used GCC 9.2 with flags to support the correct subset of RISC-V instructions. We developed a custom linker script to support placing text and data in the specific memory regions that complies with the memory model provided to students as part of the final project. Additionally, we need the RISC-V version of elf2hex to convert the ELF (executable) produced after linking into a human readable hex file, which is then read by the testbench.

6 STATISTICS AND DISCUSSION

We switched the course to use RISC-V in Fall 2019. For the most recent 10 terms the course was offered, including the semester we taught RISC-V, 41% of students (N=282) responded to the university's course evaluation [4], yielding the following statistics:

- 95% had a strong desire to take the course
- 98% thought the course advanced their understanding of the subject matter
- 91% expressed an increased interest in the subject because of this course

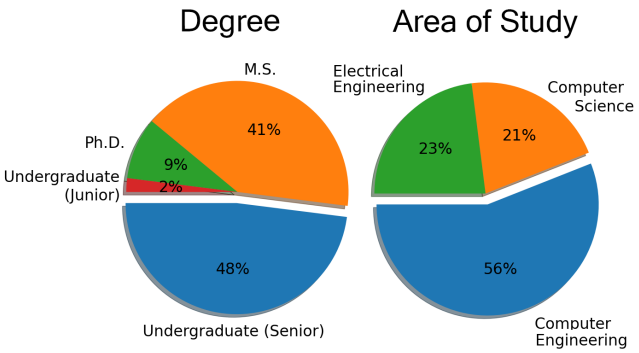

Figure 4: Degree program of students (left) and area of study among undergraduate students (right) for the last 10 semesters. Computer engineering undergraduate students represent a plurality, but the class has representation from all three areas of study in our EECS department and across graduate programs as well.

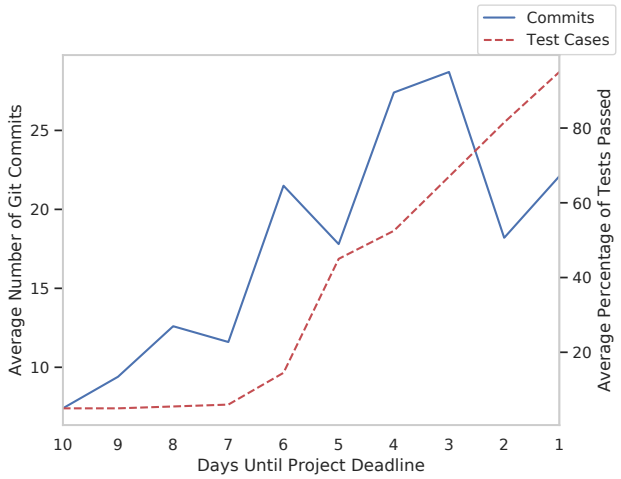

Figure 5: The average number of of Git commits made and test cases passed by all groups in the ten days leading up to the final project deadline.

Figure 4 shows the degree programs and areas of study among undergraduates who take the class. While computer engineering undergraduates is the largest single group, we find representation of undergraduates from both computer science and electrical engineering, as well as both M.S. and Ph.D. students.

6.1 Project Data and Statistics

6.1.1 Git and Autograder Statistics. In the first semester we taught RISC-V (Fall 2019), we found that a typical group's project consists of approximately 3,500-5,000 lines of code (not including test code). Since the last week of the project is, for many groups, a crunch period to get everything working, we also looked at how many Git commits were made per day by each group and how many of the test cases their project passed in the ten days leading up to the project due date. We found that the number of commits averaged 7-29 for each group during this period (Figure 5) and many groups do not pass the majority of test cases until about 3-4 days prior to

#	Question	Response Range
1	Before this class, how comfortable were you at reading and writing Verilog?	1 (very uncomfortable) - 5 (very comfortable)
2	How comfortable are you at reading and writing Verilog now?	1 (very uncomfortable) - 5 (very comfortable)
3	This project improved my understanding of computer architecture concepts (e.g. out-of-order pipelines, memory systems, performance measurements, etc).	1 (strongly disagree) - 5 (strongly agree)
4	After this project, I have a better understanding of how software interacts with hardware.	1 (strongly disagree) - 5 (strongly agree)
5	Rate the difficulty of the module design and unit testing phase of the project.	1 (not difficult) - 5 (very difficult)
6	Rate the difficulty of the pipeline integration and debugging phase of the project.	1 (not difficult) - 5 (very difficult)
7	Rate the difficulty of the pipeline synthesis and testing/debugging phase of the project.	1 (not difficult) - 5 (very difficult)
8	Did you use the RISC-V compiler to compile your own test cases?	0 (no) or 1 (yes)
9	How valuable was the compiler in allowing you to debug your pipeline with test cases you wrote in C?	1 (not at all valuable) - 5 (very valuable)
10	This class made me more interested / less interested / did not change my interest in computer architecture design and/or research.	1 (less interested), 2 (did not change), or 3 (more interested)

Table 2: Survey Questions

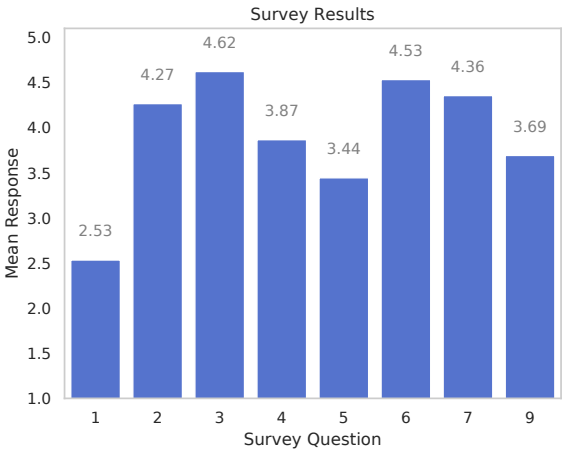

Figure 6: Responses for survey questions in Table 2 except Q8 (for which 56% of students indicated they used the compiler) and Q10 (for which the mean response was 2.63/3).

the deadline. Based on discussions with several groups we believe this is because they had not yet matched the format required for the autograder, not because their project did not work.

6.1.2 Student Survey. In Fall 2019, we surveyed students after the project to obtain their opinion of the project's level of difficulty, the value of the compiler in writing their own test cases, and how it advanced their own skills and understanding of computer architecture. A total of 42 students responded to the survey and their responses are shown in Figure 6. Notably, students believed the project improved their SystemVerilog skills and their understanding of computer architecture concepts including the hardware/software interface. Students believed the pipeline integration and synthesis phases were more difficult than the module design phase, likely due to the number of bugs exposed during these phases. We also found only about half the class used the compiler, but those who did found it useful to write their own test cases for debugging.

6.2 Discussion

In a typical recent semester, about 5% of students fail or withdraw, but we try to intervene early when students do poorly on the individual projects, miss meetings with their group, or turn in low quality milestone code. Students who fail the class generally fail because they perform very poorly on both the individual projects and the group project. On the other hand, a group almost never performs poorly enough on the project to fail the class, because all groups end up with at least a semi-functional OoO processor. None of this is to say our course is perfect, nor that we have anticipated every possible situation. We are constantly making improvements based on student and instructor feedback.

7 CONCLUSION

We have described our experience teaching a design course in which students build an OoO processor supporting a subset of the RISC-V ISA. Changing ISAs was a worthwhile investment due to the advantages RISC-V offers compared to Alpha despite the upfront work. First, we no longer worry about the end of support for Alpha. Second, the free-to-use RISC-V is simple and modular, which is easier for students to learn and debug the reference hardware designs, and build their own. Finally, an ongoing challenge has been to provide students with better methods to debug their projects, and migrating to RISC-V gives students access to the GCC compiler to write test cases in C. Since it was easier for groups to write programs that exploit their hardware's features, about half the groups submitted a test case to be added to the class suite.

In this paper we set out to describe our course in detail. We hope this description is useful to others, and we encourage you to make use of or modify our materials if you find them useful!

Acknowledgement

We are grateful to Austin Rovinski for beta testing the RISC-V project, and Siying Feng and Xueyang Liu for teaching. We thank the instructors who contributed to the course: Mark Brehob, Tom Wenisch, Jon Beaumont, and Will Cunningham. Finally we thank our students without whom this project would not be possible!

REFERENCES

[1] 2020-08-27. *Arm M4 Processor.* https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4

[2] 2020-08-27. *Global Microprocessor Market Will Reach USD 8,894 Million By 2025.* https://www.globenewswire.com/news-release/2019/05/10/1821796/0/en/Global-Microprocessor-Market-Will-Reach-USD-8-894-Million-By-2025-Zion-Market-Research.html

[3] 2020-08-27. *Synopsys Academic Programs.* https://www.synopsys.com/community/academic-programs.html

[4] 2020-08-27. *University of Michigan Atlas Course Profiling Tool (EECS 470).* https://atlas.ai.umich.edu/course/EECS%20470/

[5] 2020-08-27. *Wikipedia: DEC Alpha.* https://en.wikipedia.org/wiki/DEC_Alpha

[6] 2020-12-01. *GitHub Repository: OpenCompArchCourse.* https://github.com/jieltan/OpenCompArchCourse

[7] Rashmi Agrawal, Sahan Bandara, Alan Ehret, Mihailo Isakov, Miguel Mark, and Michel A Kinsy. 2019. The BRISC-V Platform: A Practical Teaching Approach for Computer Architecture. In *Proceedings of the Workshop on Computer Architecture Education*. 1–8.

[8] Lieven Eeckhout. 2017. Is Moore's Law Slowing Down? What's Next? *IEEE Micro* 4 (2017), 4–5.

[9] Roberto Giorgi and Gianfranco Mariotti. 2019. WebRISC-V: a Web-Based Education-Oriented RISC-V Pipeline Simulation Environment. In *Proceedings of the Workshop on Computer Architecture Education*. 1–6.

[10] Norman P. Jouppi. 1990. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. *SIGARCH Comput. Archit. News* 18, 2SI (May 1990), 364–373. https://doi.org/10.1145/325096.325162

[11] Jason Lowe-Power and Christopher Nitta. 2019. The Davis In-Order (DINO) CPU: A Teaching-focused RISC-V CPU Design. In *Proceedings of the Workshop on Computer Architecture Education*. 1–8.

[12] Scott Mcfarling. 1993. *Combining Branch Predictors.* Technical Report.

[13] K. J. Nesbit and J. E. Smith. 2004. Data Cache Prefetching Using a Global History Buffer. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*. 96–96. https://doi.org/10.1109/HPCA.2004.10030

[14] Subbarao Palacharla, Norman P Jouppi, and James E Smith. 1996. *Quantifying the complexity of superscalar processors.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

[15] Alberto Pirati, Jan van Schoot, Kars Troost, Rob van Ballegoij, Peter Krabbendam, Judon Stoeldraijer, Erik Loopstra, Jos Benschop, Jo Finders, Hans Meiling, et al. 2017. The future of EUV lithography: enabling Moore's Law in the next decade. In *Extreme Ultraviolet (EUV) Lithography VIII*, Vol. 10143. International Society for Optics and Photonics, 101430G.

[16] Parthasarathy Ranganathan. 2017. End of Moore's Law: Or, a Computer Architect's Mid-life Crisis?. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 1–1.

[17] J. Schutz and R. Wallace. 1998. A 450 MHz IA32 P6 family microprocessor. In *1998 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, ISSCC. First Edition (Cat. No.98CH36156)*. 236–237. https://doi.org/10.1109/ISSCC.1998.672450

[18] André Seznec. 2011. A New Case for the TAGE Branch Predictor. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (Porto Alegre, Brazil) *(MICRO-44)*. Association for Computing Machinery, New York, NY, USA, 117–127. https://doi.org/10.1145/2155620.2155635

[19] Andrew Waterman and RISC-V Foundation Krste Asanovic. 2019. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20190608-Base-Ratified",.

[20] K. C. Yeager. 1996. The Mips R10000 superscalar microprocessor. *IEEE Micro* 16, 2 (April 1996), 28–41. https://doi.org/10.1109/40.491460

[21] Tse-Yu Yeh and Yale N. Patt. 1991. Two-Level Adaptive Training Branch Prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture* (Albuquerque, New Mexico, Puerto Rico) *(MICRO 24)*. Association for Computing Machinery, New York, NY, USA, 51–61. https://doi.org/10.1145/123465.123475