International Journal of Computer Architecture Education, 2025, 14:1
ISSN: 2316-9915 • doi: 10.5753/ijcae.2025.6543

RESEARCH PAPER

Teaching Computer Architecture through an Integrated Top-Down RISC-V Processor Design Approach

Guilherme Esmeraldo [Instituto Federal de Educação, Ciência e Tecnologia do Ceará | *guilhermealvaro@ifce.edu.br*]
Edson Lisboa [Instituto Federal de Educação, Ciência e Tecnologia de Sergipe | *edson.lisboa@academico.ifs.edu.br*]
Victor Medeiros [Universidade Federal de Pernambuco | *vwcm@cin.ufpe.br*]
Edna Barros [Universidade Federal de Pernambuco | *ensb@cin.ufpe.br*]

Instituto Federal de Educação, Ciência e Tecnologia do Ceará campus Crato, Rod. CE 292 Km 15, s/n, Gisélia Pinheiro, Crato, Ceará, 63115-500, Brazil.

Abstract.
Computer organization and architecture teaching has historically been supported by didactic simulators, notably associated with the MIPS architecture, whose simplicity favored the understanding of fundamental concepts. However, the rise of the RISC-V architecture, conceived as an open, modular, and extensible model, has reshaped the educational landscape, demanding new pedagogical approaches. Beyond being fundamental for the initial understanding of concepts such as pipelining, memory hierarchy, and instruction execution, simulation also provides a means to explore multiple abstraction levels and validate designs against a formal reference model. This dual role reinforces conceptual learning and fosters a more practical and applied educational experience. In this context, integration with simulation tools that provide support at multiple abstraction levels significantly enhances the exploration of fundamental aspects in the teaching–learning process of computer organization and architecture, while also assisting in the design of complex digital systems. This article presents a didactic–methodological approach that articulates different abstraction levels, ranging from functional and RTL simulation to validation and testing, considering a reference model integrated with the tools. Such an approach fosters interactive experimentation and customization of the RISC-V ISA and brings the teaching process closer to professional practice in computer engineering. A comparative analysis of existing simulators and a description of the proposed methodology highlight that combining simulations at different abstraction levels strengthens student training, offering a robust, modern, and contemporary learning ecosystem aligned with computing demands. Finally, the article discusses the results of applying this teaching approach during one semester of the Computer Engineering program at UFPE.

Keywords: Computer Archtecture Learning, Integrated Top-Down Design Approach, RISC-V, CompSim Simulator

Received: 08 September 2025 • **Accepted:** 28 September 2025 • **Published:** 21 January 2026

1 Introduction

Since the genesis of computer engineering, teaching computer organization and architecture has been one of the essential pillars for training professionals capable of understanding and designing digital systems at multiple levels of abstraction. Despite its relevance, it is an area that has historically presented didactic challenges due to the gap between theoretical formulation and its practical realization in real architectures. In this context, simulators have been consolidated over the past decades as a widely disseminated didactic strategy, enabling students to explore, in an accessible and interactive way, the internal functioning of processors, low-level instruction execution, pipeline organization, and memory hierarchy principles.

Classical tools based on the MIPS architecture, such as SPIM Larus [1990], MARS Vollmar and McIver [2005], RARS Stanford [2019], and Jupiter Moraes [2020], have played a crucial role in this process, allowing classroom practice with register visualization, program debugging, and ISA exploration in a controlled environment.

However, although fundamental, such tools remain restricted to the conceptual space of simulation, focusing on the functional model, limiting themselves to reproducing abstract models that, while useful for initial understanding, do not provide students with the concrete experience of dealing with a real computing system. In other words, simulation is necessary but insufficient for consolidating deep and consistent learning when it only addresses the verification of high-level functionalities of the studied architecture.

The advent of the RISC-V architecture Waterman *et al.* [2017], open, modular, and extensible in nature, has fostered new pedagogical possibilities by allowing exploration in simulated environments and the analysis of non-functional aspects important in real hardware implementations. At the same time, modern simulators have emerged — such as Ripes Thor [2019], BRISC-V Mishra *et al.* [2019], WebRISC-V Fisler [2020], and CompSim Esmeraldo *et al.* [2023] — that expand the scope of didactic visualization and, in some cases, offer support for FPGA synthesis, bringing learning closer to the concrete space of physical implementation. This feature is crucial, as it allows students to experience simulated behavior and the performance, limitations, and specificities of architectures effectively implemented in reconfigurable hardware.

Additionally, simulation tools that allow different abstraction models and consider non-functional aspects, such as RTL level and timing, significantly complement the teaching–learning process by bringing activities closer to real hardware implementation. In this sense, traditional low-level simulation tools such as ModelSim (Intel/Altera) Ramachandran [2007] have played a central role in the materialization of digital projects, enabling students to implement, test, and validate processors close to the real hardware level. Although widely disseminated, such environments present complex and often challenging interfaces for pedagogical use in the early stages of learning. This reinforces the need for solutions that reconcile the didactic simplicity of simulators with the practical depth of synthesis tools, reducing the fragmentation of the teaching–learning process.

Nevertheless, there remains a significant gap in the current ecosystem: few tools integrate, in a cohesive manner, the didactic clarity of simulation with the practical depth required at the lower levels to enable real hardware synthesis. Thus, fragmented environments persist, forcing students to move between multiple platforms, fragmenting the teaching–learning process and hindering the consolidation of concepts.

In light of this scenario, this work presents a didactic–methodological approach for teaching computer organization and architecture centered on the RISC-V ISA. The approach adopts a top-down perspective in which students develop and test assembly programs in CompSim, then progressively advance toward understanding and implementing the underlying microarchitecture in SystemVerilog. CompSim is extended to act as an integrative tool, supporting the generation of artifacts such as MIFs (Memory Initialization Files) Intel Corporation [2023], reference profiles, and the standardized `tb_top.sv` testbench enables seamless comparison between functional simulation and RTL execution in ModelSim. This workflow reduces fragmentation, allowing students to experience the trajectory from ISA to microarchitecture continuously and structured.

Furthermore, the article reports on applying this approach in the 2025.1 semester of the Computer Engineering program at UFPE. It discusses the outcomes obtained and highlights its effectiveness in reinforcing engagement and conceptual understanding through incremental practice and integrated validation. Finally, this work paves the way for future integration that will enable timing-constrained simulation and testing on a real FPGA platform, while consistently maintaining the same validation strategy across all abstraction levels.

The text is organized as follows. The next section presents the state of the art in teaching methodologies for computer architecture and design. In Section 3, the method adopted in the computer architecture course at UFPE that motivated the proposal of this work will be presented. In Section 4, we describe the proposed methodology in detail. In Section 5, preliminary results on improving learning are presented, and in Section 6, we present conclusions and future work.

2 State-of-The-Art

The teaching–learning process in Computer Organization and Architecture has historically been supported by the use of didactic simulators, which allow the abstraction of the complexity inherent to physical hardware while simultaneously offering students a controlled environment for experimentation. The MIPS architecture has traditionally been consolidated as a pedagogical reference due to its structural simplicity, instruction set orthogonality, and broad adoption in classical textbooks. Environments such as SPIM Larus [1990], MARS Vollmar and McIver [2005], RARS Stanford [2019], and Jupiter Moraes [2020] have become central tools for academic practice, providing support for program editing, execution, and debugging, with features for visualizing registers, memory, and execution flow. Although effective from a didactic perspective, such tools present significant limitations regarding ISA extensibility and support for contemporary architectures. In recent decades, the emergence of the RISC-V architecture, conceived as an open, modular, and extensible ISA, has reshaped the academic and industrial landscape Waterman *et al.* [2017]. Unlike MIPS, whose licensing is restricted to proprietary contexts, RISC-V enables free experimentation, fostering the development of innovative simulators. Within this context, tools such as Ripes Thor [2019], BRISC-V Mishra *et al.* [2019], WebRISC-V Fisler [2020], emulsiV Hoover [2020], and, notably, CompSim Esmeraldo *et al.* [2023], have emerged, explicitly conceived for pedagogical purposes, with support for pipeline visualization, memory hierarchy, and real-time interactivity mechanisms. Such simulators align with the need for more dynamic didactic environments, ensuring greater adherence between theory and practice. In parallel, professional and research-oriented simulators such as Spike Waterman and Asanović [2017], the official simulator of the RISC-V Foundation, riscOVPsim Ltd. [2019], Whisper Inc. [2019], rv8 Hall [2018], and gem5 Binkert *et al.* [2011] have gained ground. These tools are characterized by high temporal accuracy and support for multiple ISA extensions, enabling detailed microarchitectural analysis, cache modeling, and integration with hardware prototyping environments. While highly relevant for scientific investigation, their configuration complexity and lack of pedagogical interfaces make them less accessible for introductory teaching. On the other hand, multi-ISA simulators such as QEMU Bellard [2005] and gem5 Binkert *et al.* [2011] have been consolidated as versatile alternatives capable of emulating heterogeneous architectures (x86, ARM, RISC-V, among others) and supporting custom extensions. Although valuable for advanced experimentation and research prototyping, their steep learning curve and absence of intuitive didactic resources distance them from the context of basic academic instruction. The comparative analysis summarized in Table 1 thus highlights the coexistence of two fundamental axes: on one side, simulators designed for didactics, and on the other, tools aimed at high-performance research. However, a gap remains between these extremes, particularly regarding reconciling pedagogical interactivity, architectural completeness, and ISA customization capability. This observation justifies and motivates the proposition of innovative approaches, such as CompSim Esmeraldo *et al.* [2023], which positions itself as a hybrid alternative, capable of combining didactic simplicity with the flexibility and extensibility demanded by emerging architectures.

Table 1. Comparative analysis of simulation tools for Computer Organization and Architecture education

Tool	Arch.	Main Purpose	Educational Features	ISA Extensibility	Ease of Use
SPIM	MIPS	Introductory teaching	Program execution, register and memory visualization	Fixed ISA (MIPS)	Low
MARS	MIPS	Introductory teaching	Integrated IDE, debugging, graphical visualization	Fixed ISA (MIPS)	Low
RARS	MIPS	Classroom practice	Integration with external projects (JavaFX), online use	Fixed ISA (MIPS)	Low
Jupiter	MIPS	Teaching	Lightweight JavaScript simulator, browser-based	Fixed ISA (MIPS)	Low
Ripes	RISC-V	Teaching architecture	Interactive visualization of pipeline and memory	Extensible (RISC-V)	Medium
BRISC-V	RISC-V	Design exploration	CPU exploration toolbox, supports architectural changes	Extensible (RISC-V)	Medium
WebRISC-V	RISC-V	Online teaching	Browser-based execution, no installation required	Basic ISA (RISC-V)	Low
emulsiV	RISC-V	Lightweight practice	Simple simulator for RV32I instructions	Restricted ISA (RISC-V)	Low
CompSim	RISC-V	Hybrid (teaching and practice)	Integrated graphical environment, Pipeline visualization, memory hierarchy, RTL/FPGA support	Extensible, educational focus, Extensible (multi-ISA)	Medium-Low
Spike	RISC-V	Reference simulator	High accuracy, official RISC-V Foundation tool	Full RISC-V support	High
riscOVPsim	RISC-V	Research and prototyping	Timing accuracy, hardware prototyping integration	Multiple RISC-V extensions	High
Whisper	RISC-V	Research (SiFive)	Debugging and detailed analysis	Extensible	High
rv8	RISC-V	Performance and debugging	Fast simulation, debugging support	Extension support	High
gem5	Multi-ISA	Advanced research	Detailed modeling, caches, multi-level abstraction	Highly extensible	Very High
QEMU	Multi-ISA	Emulation and virtualization	Focus on virtualization, not education	Extensible (multi-ISA)	High

3 Methodological Framework for a Top-Down RISC-V Processor Design Approach

Our proposal is based on a top-down approach, in which students begin their learning journey in computer architecture by understanding the RISC-V Instruction Set Architecture (ISA), exploring it in the CompSim simulation environment through the writing of assembly programs. They then advance to implementing a microarchitecture at the Register Transfer Level (RTL), using SystemVerilog, which is validated through simulation with the ModelSim tool.

The pedagogical structure integrates theoretical and practical activities, distributed over the semester in a total workload of 90 hours, with 60 hours allocated to theoretical classes and 30 hours to laboratory activities. The plan foresees that, each week, four hours of theory are complemented by two hours of practice, allowing concepts discussed in class to be immediately applied and deepened in laboratory activities. It is worth noting that the division between theory and practice is not rigid: specific topics required for carrying out the practical activities may be introduced directly within the laboratory context.

In the first stage of the course, the focus is on mastering the ISA, specifically the basic RV32I subset. Students develop assembly programs proposed throughout the first weeks, which progressively explore different instructions of the set. The activities are structured to suggest and, in some cases, restrict subsets of instructions, ensuring that students become familiar with the detailed functioning of each operation. To mitigate the risk that superficial use of generative artificial intelligence tools negatively impacts learning, strategies are adopted such as requiring written explanations about the functioning of each instruction used; applying short oral tests so that students demonstrate their individual understanding during monitoring activities; and comparing different solutions, encouraging students to justify their choice of instructions and control structures.

Once students have consolidated their understanding of the RV32I instruction set, they move on to the implementation stage. At this point, an initial RTL-level pipeline version implemented in SystemVerilog is made available via a GitHub repository. This version supports only a minimal subset of instructions — BEQ, LW, SW, ADD, and AND — to provide a functional but incomplete base that serves as a starting point for incremental development.

From this point on, students carry out weekly deliverables, each one responsible for implementing a new group of RV32I instructions, following the established order:

- **Week 1 – Arithmetic, Logic, and Shifts:**
  - R-Format: add, sub, sll, slt, sltu, xor, srl, sra, or, and;
- **Week 2 – Memory Access:**
  - L-Format: lb, lh, lw, lbu, lhu;
  - S-Format: sb, sh, sw;
- **Week 3 – Arithmetic, Logic, and Shifts with Immediates:**
  - I-Format: addi, slti, sltiu, xori, ori, andi, slli, srli, srai;
- **Week 4 – Branches and Jumps:**
  - B-Format: beq, bne, blt, bge, bltu, bgeu;
  - J-Format: jal, jalr;
- **Week 5 – Upper Immediate Instructions and Halt:**
  - U-Format: lui, auipc;
  - the pseudo-instruction halt.

This incremental flow allows students to gradually advance in the complexity of the instructions, starting with basic arithmetic and logic operations and progressing to memory instructions, conditional branches, and function calls. In this way, students develop a deeper understanding of the architectural implications of each instruction category, strengthening their skills in implementing, debugging, and validating a microarchitecture.

The entire functional simulation process is carried out using the ModelSim tool. Each deliverable is accompanied by a set of memory initialization files (MIF) generated by CompSim, and reference files are also generated by CompSim, as explained in the next section. These artifacts enable students to test their implementations in ModelSim, comparing the RTL simulation results with the expected results obtained in the CompSim simulator. This practice provides a standardized verification mechanism, guiding students' analysis in cases of divergence and making the debugging process more systematic and focused. Usually, the formats generated in the different learning stages are not standardized, which makes the validation process more difficult for students. The technique of comparing standardized output files allows students, through a simple file comparison, to identify failure points that can then be debugged through more detailed waveform analysis, for example. Furthermore, the adopted approach enables an integrated verification flow, extending the same procedures to subsequent stages, such as timing simulations and, later, implementation on reconfigurable hardware (FPGA).

4 Integrated CompSim Support for the Top-Down Approach

Hardware development is challenging, mainly due to the fragmentation of the tools used throughout the design flow. For students, the difficulty is even greater, since in addition to understanding the concepts of computer architecture, they must also deal with the heterogeneity of platform specifications and learn how to operate each. The design flow includes: hardware description in languages such as VHDL or SystemVerilog, simulators for functional validation, and specific environments for synthesis and later implementation on an FPGA. Each stage has distinct interfaces and formats, significantly increasing the learning load.

Our proposal seeks to mitigate this complexity by centralizing the processor design workflow in CompSim, which serves as an integrating environment. Although external calls to specialized tools are still required, CompSim ensures that all results are presented in a standardized format, offering students a continuous environment for the development flow and facilitating interoperability between tools. Figure 1 illustrates this process.

In Step 1, the student codes the application in RISC-V assembly language using CompSim's editor. Once the code is generated, an iterative cycle begins between Step 2 (high-level simulation inside CompSim) and Step 2.1 (program refinement and validation), until the execution matches the expected behavior.

After this stage, Step 3 generates essential files for the next step, in which the processor is implemented in SystemVerilog: (i) MIF files for initializing instruction and data memories; (ii) reference profiles of memory and registers, which serve as the basis for validating the RTL simulation results (golden files); and (iii) the `tb_top.sv` file, which acts as the top-level entity of the SystemVerilog testbench, is responsible for instantiating the microarchitecture developed in Step 4 and ensuring standardization in the generation of memory and register profiles.

CompSim also provides direct support for running ModelSim, which compiles and simulates the SystemVerilog RTL microarchitecture. The output files obtained in ModelSim are generated in a standardized format, enabling their comparison with the reference profiles. This process is illustrated in Steps 5 and 5.1, representing the iterative cycle of simulation, validation, and correction of the microarchitecture.

Finally, the proposed flow is extensible: in future versions, the exact mechanisms may be applied to timing simulations and, subsequently, to implementation on reconfigurable hardware. In this latter case, the memory and register profiles could be obtained via dumping through the JTAG interface, ensuring consistency of the verification process throughout all project stages.

4.1 RISC-V RV32I Model for CompSim

CompSim Esmeraldo *et al.* [2023] is an educational simulator designed to support the teaching of Computer Organization and Architecture by integrating assembly-level programming and processors' internal operation. It provides a dedicated assembler and a simulation environment for the RISC-V architecture, enabling students and researchers to develop programs, observe the execution of instructions, and analyze data flow across registers, memory, and functional units. Conceived as a pedagogical tool, CompSim emphasizes accessibility and interactivity while maintaining technical accuracy, thereby bridging the gap between theoretical concepts and practical experimentation in modern architectures.

The processor implemented in CompSim adheres to the RV32I specification, covering 37 core instructions distributed across the R, I, S, B, U, and J formats, in addition to the pseudo-instruction halt, which signals the end of simulation. This set encompasses arithmetic, logical, shift, memory access, conditional branch, and jump instructions, offering the essential foundation for understanding the RISC-V architecture. Privileged instructions such as ecall, ebreak, and CSR-related operations were intentionally omitted, as the simulator was designed primarily for introductory educational purposes focused on the user-level ISA.

The correctness of the implementation was ensured through the RISC-V Architecture Compliance Test SIG RISC-V International [2023], made available by the RISC-V Foundation. This suite contains thousands of test cases that validate the proper generation of bytecodes and the expected functional behavior of the RV32I instruction set. The CompSim processor passed 15,411 tests, confirming compliance with the official specification and validating its reliability as a teaching and experimentation tool. Such rigorous conformance guarantees that students can trust the outcomes of their experiments, as they accurately reflect the behavior of real RISC-V processors.

CompSim was extended to include multiple simulation models targeting different educational and analytical needs. The single-cycle model executes one instruction per clock cycle, supporting an initial understanding of the hardware–instruction relationship. The dual-cycle model refines this approach by splitting execution into two stages: instruction fetch (retrieving the instruction from memory) and instruction execution (processing the fetched instruction), offering students a clearer view of the separation between memory access and computation. The cycle-accurate model introduces further temporal fidelity by dividing execution into five classical stages (Instruction Fetch, Instruction Decode, Execution, Memory Access, and Write Back). A pipeline-enabled cycle-accurate model enhances realism by allowing parallel instruction execution, making it possible to study hazards

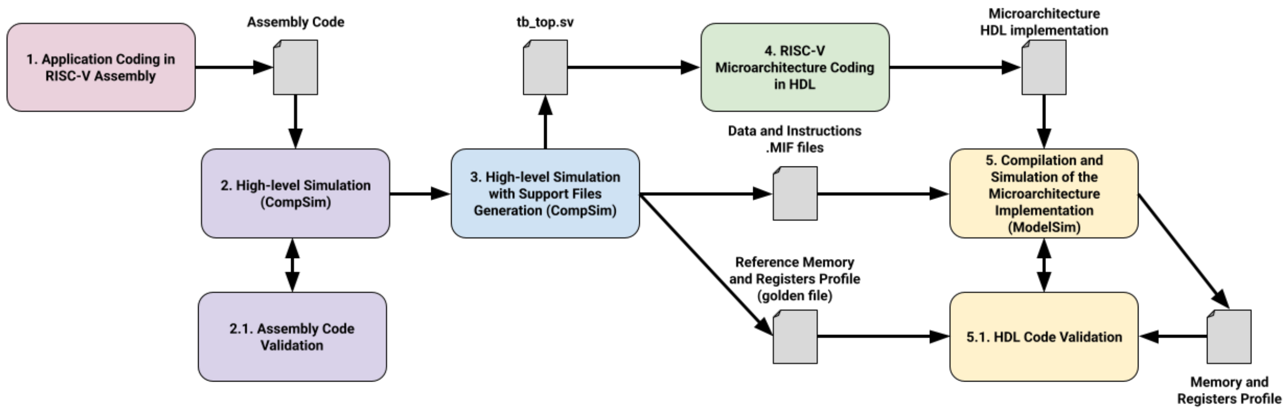
1. Application Coding in RISC-V Assembly
Assembly Code
2. High-level Simulation (CompSim)
2.1. Assembly Code Validation
3. High-level Simulation with Support Files Generation (CompSim)
tb_top.sv
4. RISC-V Microarchitecture Coding in HDL
Microarchitecture HDL implementation
Data and Instructions .MIF files
Reference Memory and Registers Profile (golden file)
5. Compilation and Simulation of the Microarchitecture Implementation (ModelSim)
5.1. HDL Code Validation
Memory and Registers Profile

Figure 1. Integrated CompSim Top-Down Approach Flow

```
  addi x1,x0,16
  addi x2,x0,32
  halt
```

```
DEPTH = 65536;        -- The size of memory in words
WIDTH = 8;            -- The size of data in bits
ADDRESS_RADIX = DEC;  -- The radix for address values
DATA_RADIX = BIN;     -- The radix for data values
CONTENT               -- Start of (address: data pairs)
BEGIN

000: 10010011; -- addi x1,x0,16
001: 00000000;
002: 00000000;
003: 00000001;

004: 00010011; -- addi x2,x0,32
005: 00000001;
006: 00000000;
007: 00000010;

008: 11111111; -- halt
009: 00000000;
010: 00000000;
011: 00000000;

END;
```

Figure 2. Example of Assembly code and corresponding MIF file generated by CompSim.

and dependencies. Finally, the behavioral model abstracts away timing considerations, executing all instructions continuously and providing a faster functional validation option. This diversity of models makes CompSim a flexible platform suitable for introductory education and advanced architectural experimentation.

In addition to instruction execution, the CompSim assembler provides direct support for generating Memory Initialization Files (MIFs) Intel Corporation [2023], which contain the binary representation of both instructions and data to be preloaded into RAM. These files are crucial in hardware design flows, as they allow memory blocks in FPGA-based implementations to be initialized with the program executed by the RISC-V processor, ensuring consistency between simulation and hardware deployment. The generation of MIF files is carried out by a Python script that performs a function analogous to that of an assembler: it translates RISC-V assembly code into machine code, but in this case the output is a MIF file that can be directly used in simulations. Figure 2 illustrates the structure of a simple MIF file generated through this process.

```
RESET AT 5000
t=75000:     x3  <= 0x00000000
t=85000:     x4  <= 0x00000000
t=95000:     x5  <= 0x00000000
t=145000:    x10 <= 0x00000000
t=1355000:   x3  <= 0x00000000
t=1365000:   x4  <= 0x00000000
t=1375000:   x5  <= 0x00000000
t=1425000:   x10 <= 0x00000000
t=2635000:   x3  <= 0x00000000
t=2645000:   x4  <= 0x00000000
...
```

Figure 3. Excerpt of a register-write golden file produced by CompSim/-ModelSim.

Furthermore, after each simulation run, CompSim enables the generation of simulation profiles (golden files) IEEE and Accellera Systems Initiative [2020], which capture the expected states of memory and processor registers. Figure 3 illustrates the golden file corresponding to the write operations performed on the processor registers during simulation. This artifact records the time-stamped updates of register values, serving as a reference for validation and comparison with the expected behavior of the RISC-V implementation.

The `tb_top.sv` file plays a central role in the verification flow, acting as the standardized top-level SystemVerilog testbench that instantiates the developed microarchitecture. In addition, it is responsible for exposing the interfaces of the register file and memory — including read, write, data, and addressing signals — so that they can be appropriately monitored and the corresponding events recorded into standardized golden files. These files are later used for comparison after simulation, serving as the basis for validating the RISC-V implementation. Figure 4 illustrates the `tb_top.sv` file, showing the interface of the register file and memory made available at the top level, the instantiation of the architecture developed during the course, and the process of recording register write operations into the golden file.

Together, these artifacts provide an integrated verification framework, streamlining the comparison between high-level simulation and RTL execution results, and supporting a more robust teaching and debugging process.

5 Results of the Proposed Approach

The proposed approach was employed in the Computer Engineering program at UFPE during the 2025.1 semester,

```
module tb_top;
    // Clock and reset (context)
    logic        tb_clk, reset;

    // Register file interface
    logic [4:0]  reg_num;
    logic [31:0] reg_data;
    logic        reg_write_sig;

    // Memory interface
    logic        wr, rd;
    logic [8:0]  addr;
    logic [31:0] wr_data;
    logic [31:0] rd_data;
    ...
    // Microarchitecture instantiation
    riscv riscV (
        .clk(tb_clk),
        .reset(reset),
        // Register file monitoring
        .reg_num(reg_num),
        .reg_data(reg_data),
        .reg_write_sig(reg_write_sig),
        // Memory monitoring
        .wr(wr),
        .rd(rd),
        .addr(addr),
        .wr_data(wr_data),
        .rd_data(rd_data)
    );
    ...
    always @(posedge tb_clk) begin : REGISTER
        if (reg_write_sig)
            $display($time," REG Write:
                reg[%d] value[%d] | [%d] |
                [%b]\n", reg_num, reg_data,
                $signed(reg_data), reg_data);
    end : REGISTER
    ...
endmodule
```

Figure 4. Excerpt of `tb_top.sv` showing the top-level interfaces of the register file and memory, and the instantiation of the developed RISC-V microarchitecture.

in the courses **Computer Organization and Architecture** (60h, theory) and **Computer Organization and Architecture Laboratory** (30h, practice), both offered in the third semester—the experiment aimed to evaluate the effectiveness of the proposed approach in improving student performance in these courses.

Adopting a top-down teaching approach, which gradually introduces the complexity of hardware architecture development and is supported by an integrated tool, proved to be a distinctive factor in the teaching and learning process. By allowing students to begin with a clear understanding of the functioning of an ISA and the interaction between hardware and software in a modern computing system, while at the same time centralizing simulation and artifact generation in CompSim (despite limitations such as the need to execute ModelSim externally), it was possible to provide greater clarity in the development flow and consistency in analyzing the results obtained by the students. This standardization reduced the fragmentation of the process and provided learners with a more guided environment for incremental debugging and verification of their solutions.

Another relevant aspect was the adoption of weekly partial deliverables, which effectively promoted student engagement. This strategy allowed implementation difficulties to be addressed gradually, avoiding the cognitive overload common when all modules are required at the final stages. The incremental nature of the proposal encouraged continuous participation and reinforced learning through practice.

The course was structured in two complementary stages. In the first, the focus was on understanding what an ISA is, including the discussion of concepts such as CISC, RISC, and their differences, culminating in practical activities where students developed assembly programs for the RV32I ISA of RISC-V and validated them on the CompSim platform. These programs were designed as didactic examples that progressively explore the core features of the ISA, including arithmetic and logical operations, conditional and unconditional branching, as well as interaction with memory and simple input/output devices. Through these exercises, students practiced implementing loops, decision structures, and data exchange routines, consolidating their understanding of low-level programming and processor behavior. Subsequently, students began to understand how this ISA is implemented in hardware through the microarchitecture, delving deeper into this study in the theoretical classes. This stage proved fundamental to understanding the hardware-software interface of a modern computing system. From this point, the exploration of the provided base microarchitecture began, based on the book **Computer Organization and Design RISC-V Edition** Patterson and Hennessy [2020], which served as a solid starting point for the proposed implementations. The first activities consisted of analyzing the functional units that compose the supplied SystemVerilog implementation, drawing parallels with what had been discussed in the theoretical classes. Students were encouraged to identify these functional units in the code, their current limitations, the strategies used for pipeline implementation, and the handling of hazards studied in class, and to propose initial ideas for expanding them to support the instructions specified in the RV32I ISA. At this stage, theoretical aspects of the SystemVerilog language were also addressed during the lab sessions, to facilitate students' understanding of the provided code and consolidate the integration between theory and practice.

During the semester, 40 students were enrolled simultaneously in both courses, with only one taking the laboratory course alone, as they had already completed the theory course. The students were encouraged to form teams of up to four members, forming 11 teams in total. The practical assignments were delivered weekly and contributed to strong classroom attendance: the average was five absences per week, approximately 12% of the total students. The failure rate did not differ from previous years, comparable to the best results observed in the last eight semesters, as illustrated in Figure 5.

It is important to emphasize that there were not two separate courses in the eight previous semesters: there was only a 75-hour course in which theory and practice were addressed within the same context. The final grade was composed of both theoretical exams and practical projects. Historically, it was observed that individual exam scores were lower than group project scores, which may indicate students' difficulty in consolidating concepts uniformly, with this effect being masked by collective assignments. In the 2025.1 semester, the theory course was assessed exclusively through individual grades, while the project grade was used only in the laboratory course. Figure 6 presents the approval and failure rates for the theory and laboratory courses in the 2025.1 semester, corroborating the previously observed pattern that perfor-

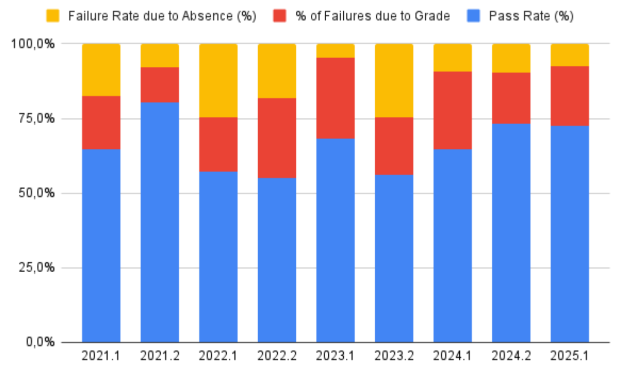

Figure 5. Approval and Failure Rates over 9 Semesters: Comparing 8 Unified Editions with the 2025.1 Theory-Only Edition

mance in group-based practical activities tends to be superior to individual theoretical assessments. Nevertheless, it is possible to observe that, even without including the project grade in the composition of the theory course, the success rate was similar to the best results from previous semesters. Only three students had divergent outcomes between the theory and laboratory courses, being approved in one and failing the other. Furthermore, the higher number of failures due to absences in the laboratory course compared to the theory course suggests that students who performed poorly on theoretical exams likely chose not to complete the group project.

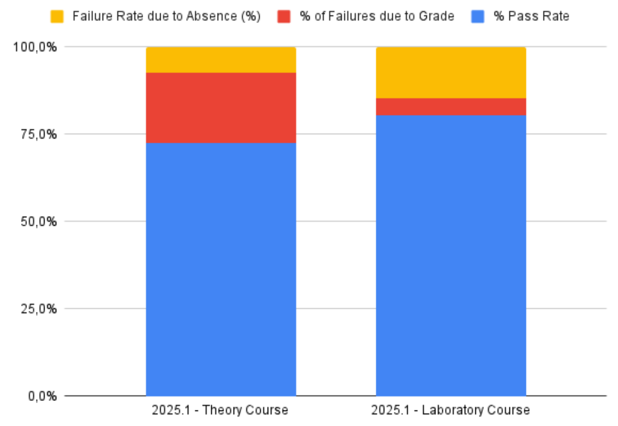

Figure 6. Approval and Failure Rates in 2025.1: Comparing the Theory Course and the Laboratory Course

The data are still limited in reaching definitive conclusions. However, it is already possible to state that the application of the approach did not negatively affect student performance and apparently strengthened class attendance and engagement through the gradual hardware development carried out in the laboratory sessions. There are still points for improvement and requirements that have not yet been implemented, which will be incorporated and evaluated in the coming semesters.

The metrics to be monitored include: approval rates, failure rates due to grades and absences, student attendance throughout the course, and the variation in scores on theoretical exams, which provide a more accurate measure of students' understanding of the content covered. These data will provide a more solid basis for assessing the impact of the proposed approach on learning and academic performance.

6 Conclusions and Future Works

The results of the proposed approach indicate that it did not negatively affect student performance. On the contrary, it contributed to increased participation, as evidenced by the strong attendance in the laboratory sessions. The gradual development of hardware, integrated with theoretical discussions, fostered student engagement and allowed knowledge to be built incrementally, strengthening their understanding of the hardware–software interface in modern computing systems.

As future work, we intend to advance the integration of the tools employed in the hardware development flow within the CompSim environment, extend the integrated platform and learning flow to include timing-constrained simulations, and integrate the platform with FPGA-based execution. We also plan to expand the implementation to other ISAs of the RISC-V specification, such as RV32M, and to define and capture new metrics to monitor student performance throughout the courses more accurately. Additionally, we are considering the adoption of alternative open-source simulation tools to complement or replace proprietary solutions like ModelSim, aiming to enhance accessibility, reproducibility, and flexibility in academic environments.

Declarations

Authors' Contributions

GE, EL, VM e EB led the conceptualization, methodology, and writing of the original draft. GE and EL contributed to software development and validation. VM contributed to data curation, preparation of figures, and visualization. EB participated in investigation and provided critical review and editing. All authors read and approved the final version of the manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets (and/or softwares) generated and/or analysed during the current study will be made upon request.

Further relevant information

Generative AI tools were used exclusively for text editing tasks, including corrections, improvements, and translation of sections of the manuscript. No generative AI tools were employed in the conception of ideas, methodology, data analysis, or interpretation of results.

References

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the USENIX Annual Technical Conference (FREENIX Track)*, pages 41–46. Available at: `https://www.usenix.org/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf`.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7. DOI: 10.1145/2024716.2024718.

Esmeraldo, G. A. R. M., Feitosa, R. G. F., Barros, E. N. d. S., Proto, E. C. P. d. S., Mello, H. M. d., Lisboa, E. B., Bispo Jr., E. L., and Campos, G. A. L. d. (2023). Uma abordagem para ensino-aprendizado de projetos de sistemas computacionais com utilização do simulador compsim com suporte à arquitetura risc-v. *Revista Brasileira de Informática na Educação*, 31:271–288. DOI: 10.5753/rbie.2023.2951.

Fisler, G. (2020). Webrisc-v: Browser-based risc-v simulator for education. Available at: `https://webriscv.github.io/`.

Hall, C. (2018). rv8: Risc-v simulator for performance and debugging. Available at: `https://rv8.io`.

Hoover, S. R. (2020). emulsiv: A lightweight risc-v instruction set simulator. Available at: `https://github.com/arcana261/emulsiV`.

IEEE and Accellera Systems Initiative (2020). IEEE standard for universal verification methodology (UVM). Available at: `https://standards.ieee.org/ieee/1800.2/7567/`.

Inc., S. (2019). Whisper: Risc-v instruction set simulator. Available at: `https://github.com/sifive/whisper`.

Intel Corporation (2023). *Quartus Prime Handbook: Memory Initialization File (.mif) Format*. Intel FPGA. Available at: `https://www.intel.com/content/www/us/en/docs/programmable/683130`.

Larus, J. R. (1990). Spim s20: A mips r2000 simulator. Technical report, University of Wisconsin-Madison. Available at: `https://course.khoury.northeastern.edu/cs4410/spim_documentation.pdf`.

Ltd., I. S. (2019). riscvovpsim: Reference simulator for risc-v. Available at: `https://github.com/riscv/riscv-ovpsim`.

Mishra, P., Sirowy, B., and Govindarajan, S. (2019). Brisc-v: An educational risc-v cpu design space exploration toolbox. In *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE)*, pages 45–50. DOI: 10.1145/3289602.3293991.

Moraes, A. (2020). Jupiter: Risc-v assembly simulator. Available at: `https://github.com/andremm/jupiter`.

Patterson, D. and Hennessy, J. (2020). *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann. Book.

Ramachandran, S. (2007). Digital vlsi systems design: A design manual for implementation of projects on fpgas and asics using verilog. DOI: 10.1007/978-1-4020-5829-5.

RISC-V International (2023). Risc-v architecture tests. Available at: `https://github.com/riscv-non-isa/riscv-arch-test` Accessed: 2025-09-07.

Stanford, D. K. (2019). Rars: Risc-v assembler and runtime simulator. Available at: `https://github.com/TheThirdOne/rars`.

Thor, M. (2019). Ripes: A visual computer architecture simulator and assembly ide. Available at: `https://github.com/mortbopet/Ripes`.

Vollmar, K. and McIver, P. (2005). Mars: An education-oriented mips assembly language simulator. *Journal of Computing Sciences in Colleges*, 21(1):50–56. DOI: 10.1145/1124706.1121415.

Waterman, A. and Asanović, K. (2017). The risc-v instruction set simulator (spike). Available at: `https://github.com/riscv-software-src/riscv-isa-sim`.

Waterman, A., Lee, Y., Patterson, D., and Asanović, K. (2017). The risc-v instruction set manual, volume i: User-level isa, version 2.2. Technical report, RISC-V Foundation. Available at: `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.pdf`.