WIMS : A Modern Web-Based MIPS Simulator for Improved Learning in Computer Architecture and Operating Systems

1st Reinaldo Assis
Instituto de Computação
Universidade Federal Alagoas (UFAL)
Maceió, Brazil
rma2@ic.ufal.br

2nd Bruno Nogueira
Instituto de Computação
Universidade Federal Alagoas (UFAL)
Maceió, Brazil
bruno@ic.ufal.br

***Abstract*—The MIPS processor is well-regarded in teaching Computer Architecture and Operating Systems due to its simpler architecture and efficient design. Simulators are a highly effective method for learning MIPS. Current simulators, like MARS, face issues such as lack of data path visualization, outdated interfaces, and installation requirements, which impede student learning. This work aims to develop an accessible online MIPS simulator featuring a modern code editor, data path visualization, step-by-step execution, and memory visualization. The simulator runs in a web browser, eliminating installation barriers and enhancing usability. This tool aims to improve the educational experience and deepen students' understanding of MIPS architecture and assembly programming.**

***Index Terms*—MIPS Processor, Computer Architecture, Assembly Programming, Educational Simulators, Data Path Visualization**

I. Introduction

The MIPS processor has long been a widely used tool in the teaching of Computer Architecture and Operating Systems due to its simplicity and stability [1]. MIPS assembly programming is an important skill for students to understand the fundamental operations of hardware and the execution of computer programs. Additionally, proficiency in low-level computer operations is crucial for developing high-performance software architectures. Consequently, effective educational tools are needed to facilitate the learning process of these concepts.

Despite its importance, the current educational landscape lacks efficient, user-friendly online simulators for MIPS assembly programming. Most existing simulators, such as MARS [2], require installation, which can be a barrier to entry, and they often lack critical features like data path visualization, which is essential for understanding the processor's operation. Furthermore, these simulators typically use outdated code editors, limiting their usability and effectiveness in modern educational settings.

The current work, WIMS (Web-based Interactive MIPS Simulator), aims to address these shortcomings by developing an online simulator for the MIPS processor. The proposed simulator is designed to be easily accessible via the web, thereby eliminating the need for software installation and lowering the barrier to entry. WIMS incorporates a modern code editor, featuring syntax highlighting, code completion, and other functionalities that significantly enhance the coding experience. This modern interface helps students minimize small mistakes and improve their learning efficiency.

One of the standout features of the proposed simulator is its data path visualization capability. By providing real-time visual feedback of the processor's data path, students will gain a deeper understanding of how instructions are executed within the MIPS architecture. Additionally, the simulator supports step-by-step execution, memory visualization, and screen simulation, offering a comprehensive suite of tools that are currently lacking in most MIPS simulators.

Moreover, WIMS is also open source, ensuring transparency and community-driven improvements. The source code, accessible at the GitHub repository[1], provides comprehensive documentation and codebase for users and contributors. Additionally, the simulator is hosted online at WIMS website[2], offering a seamless and readily accessible platform for students and educators to engage with MIPS assembly programming without the need for local installations.

The target audience for this simulator is primarily students. The design focus is on ease of use and accessibility, ensuring that the tool remains straightforward and effective for educational purposes without overcomplicating it with unnecessary features. Educators can utilize this simulator as an instructional tool in classrooms and for assigning projects, thereby enhancing the overall learning experience.

In summary, this work endeavors to develop a sophisticated, user-friendly online MIPS simulator that addresses the limitations of existing tools. By integrating modern coding tools and comprehensive visualization features, the simulator aims to provide a suitable way MIPS architecture and assembly programming are taught, ultimately fostering a deeper understanding and greater proficiency among students.

[1]https://www.github.com/ReinaldoAssis/mips-sim
[2]http://reinaldoassis.github.io/ufal-wims

II. Ralated Work

The MIPS processor has several simulators widely used in educational settings to help students understand computer architecture and assembly language programming. Among these, a few notable ones include MARS, SPIM, EduMIPS64, and Visual MIPS.

MARS (MIPS Assembler and Runtime Simulator) [2], developed by Missouri State University, is a well-known IDE for MIPS assembly language programming. It features a GUI with point-and-click control, an integrated editor, and variable-speed single-step execution. Despite its popularity, MARS lacks a more complete data path visualization, limiting students' ability to comprehend the processor's internal workings.

SPIM [3] is another popular MIPS simulator that executes assembly language programs for the MIPS32 processor. It includes a simple debugger and minimal operating system services, making it useful for instructional purposes. However, SPIM does not offer data path visualization.

EduMIPS64 [4] is a cross-platform MIPS64 CPU simulator designed to aid students in learning MIPS assembly. It provides a visual representation of the CPU, allowing users to see how instructions are processed. Despite its visual approach, EduMIPS64 still lacks comprehensive data path visualization and modern code editing features.

Visual MIPS is an online emulator developed using F# and FABLE, available for web, desktop, and mobile platforms. It aims to simplify learning MIPS assembly language by offering a user-friendly interface and real-time feedback on program execution. However, its usability is limited due to the absence of more complex features such as detailed data path visualization and advanced debugging tools.

WebMIPS [5] is a five-stage pipeline online simulator for the MIPS32 architecture, it features a complete datapath visualization and step by step execution. Although it's possible to access registers and memory data, the simulator does not offer any more features and as such lacks more advanced tools.

ViSiMIPS [6] is a simulation tool tailored to teaching pipelining concepts in computer architecture. It focuses on dynamically visualizing the stages of instruction execution in a MIPS32 pipelined processor. Its graphical interface aids students in comprehending the register-transfer-level processes involved in pipelining, which are typically challenging to grasp. The simulator also includes representations of MIPS instructions and their progression through pipeline stages.

DEVSJAVA MIPS32 [7] is a MIPS32 processor simulator developed using the DEVSJAVA [8] framework. It supports single-cycle, multi-cycle, and pipelined implementations as described in the textbook Computer Organization and Design by Patterson and Hennessy [9]. The simulator emphasizes dynamic visualization at the RT-level, allowing users to view the processor's internal operations and collect performance metrics such as cycle count, execution time, and instruction count. Additionally, it provides platform independence and the flexibility to explore processor designs at different abstraction levels, making it a versatile simulator.

MIPSFPGA [10] extends the educational potential of MIPS processor teaching by integrating graphical simulation with FPGA prototyping. It provides an environment to visualize the data path of MIPS processors with and without pipelining while supporting incremental learning through editable graphical projects. MIPSFPGA allows users to simulate, debug, and even implement processor designs on FPGA platforms.

To address these limitations, our work proposes a web-based MIPS simulator that integrates advanced features including a modern code editor with syntax highlighting and code completion, data path visualization, step-by-step execution, and memory visualization. Designed to be easily accessible online, even on mobile devices, this simulator eliminates the need for software installation and provides a more interactive and engaging learning experience for students. A comprehensive comparison of the features offered by various MIPS simulators, including our proposed tool, is presented in Table I.

TABLE I: Comparison of MIPS Simulators

Feature	WIMS	MARS	SPIM	EduMIPS64	Visual MIPS
Web-based	Yes	No	No	No	Yes
Modern Code Editor	Yes	No	No	No	No
Data Path Visualization	Yes	Limited	No	Limited	No
Step-by-Step Execution	Yes	Yes	Yes	Yes	Yes
Memory Visualization	Yes	Yes	Yes	Yes	Yes
Real-Time Screen Rendering	Yes	Yes	No	No	No
Open Source	Yes	Yes	Yes	Yes	Yes
Mobile Friendly	Yes	No	No	No	Yes

III. Simulator Architecture

The architecture of WIMS, as illustrated in Figure 1, comprises three primary components: the assembler, the worker, and the simulator. The assembler takes the code input from the front end and produces a binary of the assembly program. The user can then either execute the program or visualize the binary through the hex view page. When the user initiates the execution of the program (or steps through it), commands are dispatched to the worker module. The worker's role is to generate web workers [11] to operate the simulator in a pseudo-threaded environment, which is crucial to prevent the user interface from becoming unresponsive during the simulation. The simulator, operating using the MIPS monocycle 32-bit architecture [9], subsequently communicates its state to the worker, and the result can be accessed in the front end.

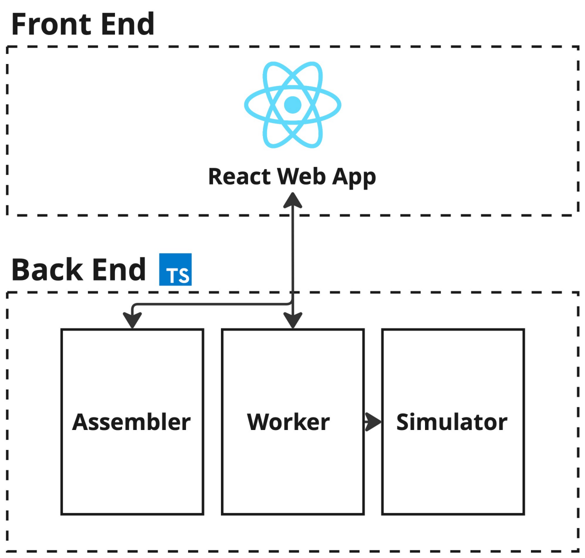
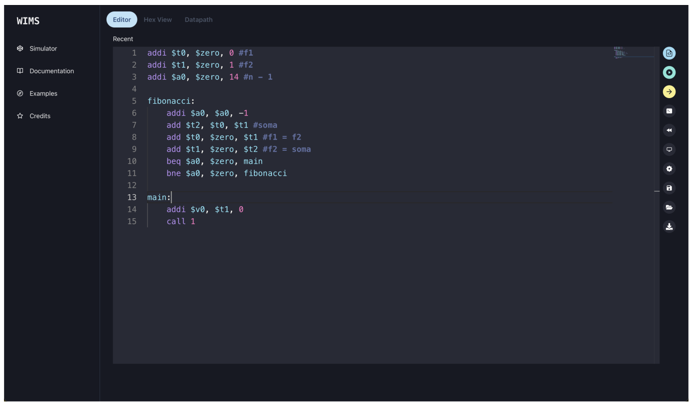


Figure 1: WIMS Architecture.

The development of the user interface aimed to create a straightforward, intuitive, and modern experience tailored for students. The chosen technology stack played a crucial role in achieving these objectives. React and TypeScript [12] were selected for their popularity, widespread use, and robust support within the open-source community. React's component-based architecture and TypeScript's static type checking significantly enhanced the development process by reducing bugs and ensuring a modern, cohesive design for the simulator.

A. Modern Code Editor Features



```
addi $t0, $zero, 0 #f1
addi $t1, $zero, 1 #f2
addi $a0, $zero, 14 #n - 1

fibonacci:
    addi $a0, $a0, -1
    add $t2, $t0, $t1 #soma
    add $t0, $zero, $t1 #f1 = f2
    add $t1, $zero, $t2 #f2 = soma
    beq $a0, $zero, main
    bne $a0, $zero, fibonacci

main:
    addi $v0, $t1, 0
    call 1
```

Figure 2: WIMS code editor

The simulator integrates the Monaco editor (as displayed in Figure 2), an open-source code editor from Visual Studio, recognized for its rich feature set. One of the key enhancements is the custom syntax highlighting, tailored specifically for MIPS assembly language. This feature improves code readability and helps students quickly identify different elements of the code, thereby enhancing the overall learning experience.

Another significant feature is code completion, which assists users by suggesting possible code completions as they type. This functionality reduces the likelihood of syntax errors and boosts coding efficiency, allowing students to focus more on learning the concepts rather than dealing with syntax issues.

Additionally, the simulator includes step-by-step line visualization. This feature allows students to visualize the execution of their code line-by-line, providing a clear understanding of the program flow and aiding in debugging. By observing the detailed execution process, students can gain deeper insights into how their code interacts with the simulated MIPS architecture.

These features collectively create a sophisticated coding environment that enhances the educational experience by minimizing errors and supporting efficient learning. The integration of these modern tools ensures that the simulator remains both user-friendly and effective for educational purposes.

B. Technical Challenges

One of the most challenging aspects of developing the MIPS simulator was the complexity involved in visualizing the data path. Initially, the approach was to create a custom framework for dynamically rendering components within a grid-like space, using a modified version of the A* pathfinding algorithm to connect these components. However, this method encountered numerous bugs, primarily due to the complexities of writing directly to the `<canvas>` element for high-performance rendering and handling different screen sizes efficiently.

Transitioning from a dynamic to a static SVG-based approach was a pivotal decision. The realization came when more time was being spent on debugging the custom library than on developing the core simulator functionalities. By manually drawing the architecture in SVG and carefully naming each wire and component, dynamic color changes in code were possible. Although this resulted in a fixed architecture, the simplicity and clarity of the code significantly improved, making it more maintainable.

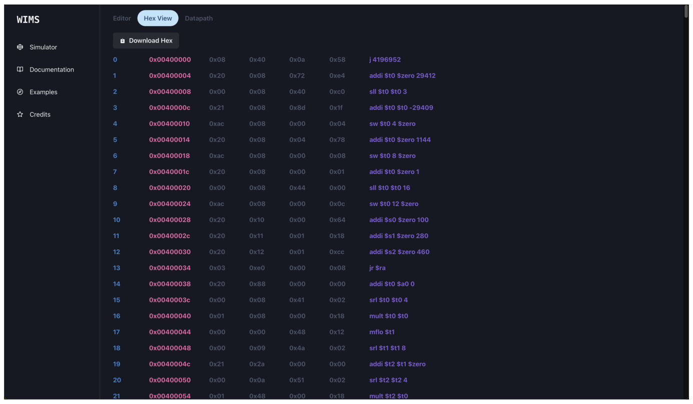
C. Performance Optimization

Optimizing the performance of the MIPS simulator was essential to ensure a smooth and efficient user experience. Several strategies were employed to achieve significant performance improvements, each addressing different aspects of the simulation process. The primary methods implemented are detailed below:

- **Web Workers for Pseudo Multi-Threading**: Implementing web workers enabled pseudo multi-threading within the browser, preventing the browser from freezing during CPU simulation. This approach resulted in a performance boost of approximately 164x. The main challenge in this method was managing inter-thread communication, which was efficiently handled by two workers responsible for managing the communication between the client and the simulator.

- **On-Demand Memory Mapping**: Instead of pre-allocating all memory, the simulator employs a strategy where memory addresses are mapped as needed. This method conserves resources by only allocating memory when a program writes to a specific address. For instance, if the program writes to address 1000, the simulator maps this address and its value, which is initially set as undefined. This approach significantly improves performance and resource efficiency, especially given the variability of devices in web environments.
- **Real-Time Screen Rendering**: Integrating real-time screen rendering was one of the most complex features to implement, as it needed to keep pace with the simulator's high velocity without introducing noticeable lag. To achieve this, the HTML canvas component is utilized directly, eschewing external libraries to minimize overhead. The screen is memory-mapped, allowing users to write an RGB color to a pixel. To update the screen, a system call instruction is provided. To fully leverage the pseudo-thread strategy, a batch processing system was developed, enabling the processor to update the screen all at once, thereby avoiding delays. We believe this to be a core feature, as it enables students to visualize the output of their programs in a clear and intuitive manner.

IV. WIMS Operation

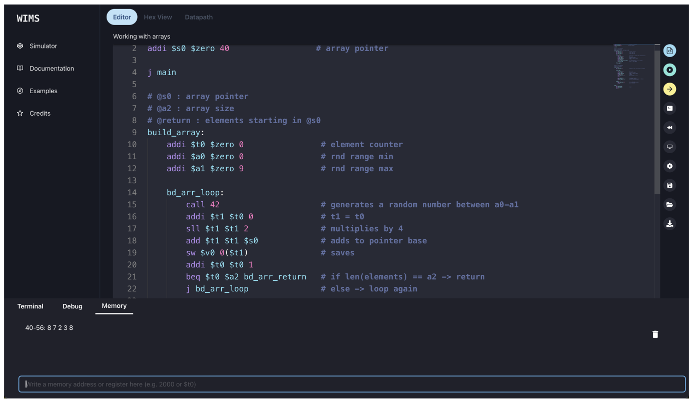


```
0   0x00400000  0x08  0x40  0x0a  0x58  j 4196952
1   0x00400004  0x20  0x08  0x72  0xe4  addi $t0 $zero 29412
2   0x00400008  0x00  0x08  0x40  0xc0  sll $t0 $t0 3
3   0x0040000c  0x21  0x08  0x8d  0x1f  addi $t0 $t0 -29409
4   0x00400010  0xac  0x08  0x00  0x04  sw $t0 4 $zero
5   0x00400014  0x20  0x08  0x04  0x78  addi $t0 $zero 1144
6   0x00400018  0xac  0x08  0x00  0x08  sw $t0 8 $zero
7   0x0040001c  0x20  0x08  0x00  0x01  addi $t0 $zero 1
8   0x00400020  0x00  0x08  0x44  0x00  sll $t0 $t0 16
9   0x00400024  0xac  0x08  0x00  0x0c  sw $t0 12 $zero
10  0x00400028  0x20  0x10  0x00  0x64  addi $s0 $zero 100
11  0x0040002c  0x20  0x11  0x01  0x18  addi $s1 $zero 280
12  0x00400030  0x20  0x12  0x01  0xcc  addi $s2 $zero 460
13  0x00400034  0x03  0xe0  0x00  0x08  jr $ra
14  0x00400038  0x20  0x88  0x00  0x00  addi $t0 $a0 0
15  0x0040003c  0x00  0x08  0x41  0x02  srl $t0 $t0 4
16  0x00400040  0x01  0x08  0x00  0x18  mult $t0 $t0
17  0x00400044  0x00  0x00  0x48  0x12  mflo $t1
18  0x00400048  0x00  0x09  0x4a  0x02  srl $t1 $t1 8
19  0x0040004c  0x21  0x2a  0x00  0x00  addi $t2 $t1 $zero
20  0x00400050  0x00  0x0a  0x51  0x02  srl $t2 $t2 4
21  0x00400054  0x01  0x48  0x00  0x18  mult $t2 $t0
```

Figure 3: Hex view page.

The typical workflow for users begins with composing an assembly program using the integrated code editor. Once the program is written, it can be assembled and executed either all at once or step-by-step utilizing the tools provided in the side pane. Moreover, users can visualize the binary output in the hex view page as depicted in Figure 3.

WIMS provides both a graphical screen and a text terminal. Users can utilize system calls to write outputs to the text terminal. Alongside this, the simulator includes a debug tab and a memory tab, which are essential for thorough debugging. For instance, users can print a specific memory region to the memory terminal by entering the start and end addresses in the format [start address]-[end address], as illustrated in Figure 4.

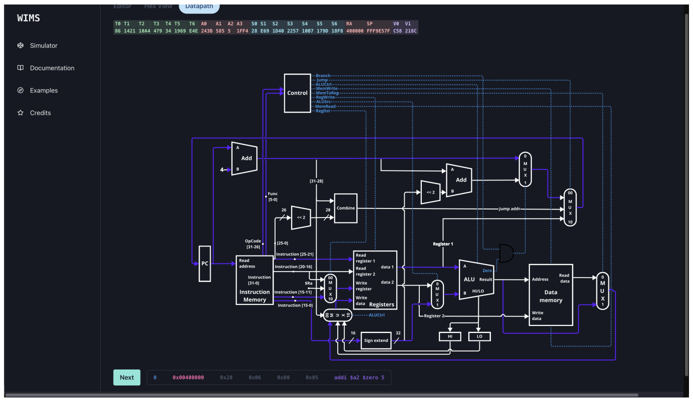


```
addi $s0 $zero 40                # array pointer

j main

# @s0 : array pointer
# @a2 : array size
# @return : elements starting in @s0
build_array:
    addi $t0 $zero 0                # element counter
    addi $a0 $zero 0                # rnd range min
    addi $a1 $zero 9                # rnd range max

    bd_arr_loop:
        call 42                     # generates a random number between a0-a1
        addi $t1 $t0 0              # t1 = t0
        sll $t1 $t1 2               # multiplies by 4
        add $t1 $t1 $s0             # adds to pointer base
        sw $v0 0($t1)               # saves
        addi $t0 $t0 1
        beq $t0 $a2 bd_arr_return   # if len(elements) == a2 -> return
        j bd_arr_loop               # else -> loop again
```

```
40-56: 8 7 2 3 8
```

Figure 4: WIMS memory terminal printing array in memory.

Additionally, users can step through an assembly program within the data path view. This feature allows them to observe the values stored in registers, the path utilized by each instruction, and view the instruction in both human-readable form and hexadecimal. By clicking the "next" button, as shown in Figure 5, users can advance through the assembled program step-by-step, gaining a detailed understanding of the instruction flow and the underlying data path operations.



```
T0 T1   T2   T3  T4 T5   T6  A0   A1  A2 A3   S0 S1  S2   S3   S4   S5   S6   RA     SP       V0  V1
86 1421 10A4 479 34 1969 E4E 243B 585 5  1FF4 28 E69 1D40 2257 10B7 179D 18F8 400000 FFF9E57F C58 218C
```

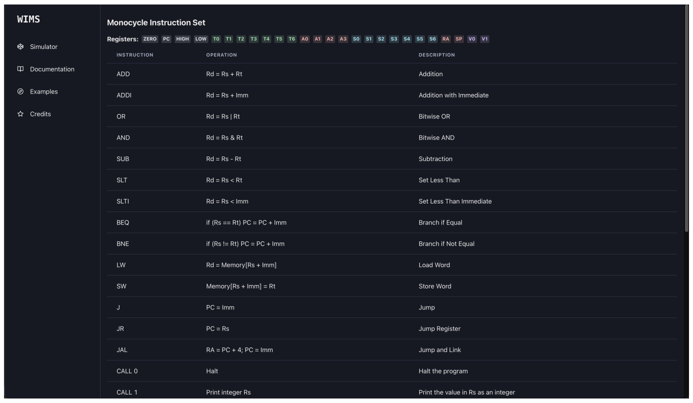
```
0   0x00400000  0x20  0x06  0x00  0x05  addi $a2 $zero 5
```

Figure 5: WIMS data path view.

Moreover, as shown in Figure 6, the WIMS website features an instruction set page and an example page, serving as valuable references for students writing assembly programs. These resources provide detailed information on the MIPS instruction set and offer practical examples to aid in the learning process. This combination of tools and resources ensures that WIMS is a powerful and accessible educational platform for mastering MIPS assembly language.

V. Results and Discussion

The effectiveness and efficiency of the developed MIPS simulator were evaluated through various complex programming tasks. One notable example is a complex 3D renderer coded in assembly using the simulator's editor. This program, consisting of approximately 1000 lines of code, leverages the simulator's screen output to display a spinning cube in real time, as shown in Figure 7.



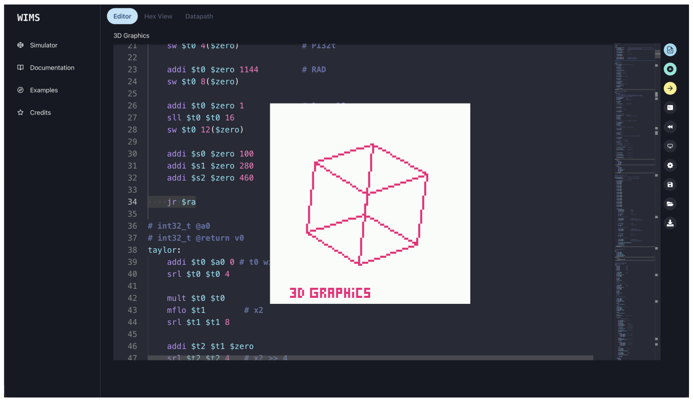
Monocycle Instruction Set

Registers: ZERO PC HIGH LOW T0 T1 T2 T3 T4 T5 T6 A0 A1 A2 A3 S0 S1 S2 S3 S4 S5 S6 RA SP V0 V1

INSTRUCTION	OPERATION	DESCRIPTION
ADD	Rd = Rs + Rt	Addition
ADDI	Rd = Rs + Imm	Addition with Immediate
OR	Rd = Rs | Rt	Bitwise OR
AND	Rd = Rs & Rt	Bitwise AND
SUB	Rd = Rs - Rt	Subtraction
SLT	Rd = Rs < Rt	Set Less Than
SLTI	Rd = Rs < Imm	Set Less Than Immediate
BEQ	if (Rs == Rt) PC = PC + Imm	Branch if Equal
BNE	if (Rs != Rt) PC = PC + Imm	Branch if Not Equal
LW	Rd = Memory[Rs + Imm]	Load Word
SW	Memory[Rs + Imm] = Rt	Store Word
J	PC = Imm	Jump
JR	PC = Rs	Jump Register
JAL	RA = PC + 4; PC = Imm	Jump and Link
CALL 0	Halt	Halt the program
CALL 1	Print integer Rs	Print the value in Rs as an integer

Figure 6: WIMS instruction set.

The 3D renderer demonstrates several key capabilities of the simulator. Firstly, the program successfully calculates rotation matrices and trigonometric functions such as sine and cosine, which are essential for rendering a 3D object. The ability to handle such computationally intensive tasks is a testament to the simulator's performance optimization strategies, particularly the use of web workers for pseudo multi-threading and on-demand memory mapping.



```
    sw $t0 4($zero)          # PI32t

    addi $t0 $zero 1144      # RAD
    sw $t0 8($zero)

    addi $t0 $zero 1
    sll $t0 $t0 16
    sw $t0 12($zero)

    addi $s0 $zero 100
    addi $s1 $zero 280
    addi $s2 $zero 460

    jr $ra

# int32_t @a0
# int32_t @return v0
taylor:
    addi $t0 $a0 0 # t0 w
    srl $t0 $t0 4

    mult $t0 $t0
    mflo $t1        # x2
    srl $t1 $t1 8

    addi $t2 $t1 $zero
    srl $t2 $t2 4   # x2 >> 4
```

Figure 7: 3D Renderer running in real-time.

Secondly, the real-time performance of the spinning cube highlights the efficiency of the simulator. Despite the complexity and size of the program, the simulator maintains smooth execution, ensuring that the cube spins without lag or noticeable delays. This real-time rendering capability underscores the simulator's suitability for educational purposes, providing students with immediate visual feedback on their assembly programs.

Furthermore, the program's success illustrates the robustness of the simulator's code editor. The Monaco editor's features, such as custom syntax highlighting, code completion, and step-by-step line visualization, significantly contributed to the development process. These tools minimized errors and facilitated the efficient writing and debugging of the extensive assembly code required for the 3D renderer.

Another benchmark program was developed to compute an approximation of π. Initially, this program took 9826 ms to complete the computation. After implementing specific performance optimization strategies, including the use of web workers for pseudo multi-threading and on-demand memory mapping, the computation time was reduced to just 60 ms. The benchmarks were conducted in the same web environment and on the same computer, demonstrating an approximately 164-fold improvement in speed. Experiment replication can be achieved by running the simulator locally, following commit codes.

VI. Conclusion

This work presented a modern, web-based MIPS simulator representing a significant advancement in the field of computer architecture and operating systems education. This work effectively addresses the limitations of existing tools, such as installation barriers, outdated interfaces, and lack of critical features like data path visualization. By integrating a modern code editor and providing extensive debugging tools, the simulator enhances the learning experience for students, making it easier to understand and debug MIPS assembly programs.

The successful implementation of benchmark programs, such as a 3D renderer and π approximation, demonstrates the robustness and effectiveness of the simulator. These programs highlight the simulator's capability to handle complex tasks and provide immediate visual feedback, which is crucial for educational purposes. The significant reduction in computation time, achieved through performance optimizations, further underscores the simulator's efficiency and suitability for educational settings.

Looking ahead, several steps are planned to further enhance the simulator. These include conducting tests in classroom environments to gather comprehensive feedback and assess its educational impact. Additionally, data will be collected to evaluate the simulator's effectiveness and usability, allowing for informed improvements. Comparative studies will also be undertaken to understand students' preferences for this simulator in relation to those cited in related work, ensuring it meets the expected educational standards. Furthermore, future development will focus on incorporating memory-mapped I/O support, expanding the simulator's capabilities and providing a more complete learning tool.

Acknowledgments

The authors express their gratitude to the National Council for Scientific and Technological Development (CNPq) for their support, and to the Research Support Foundation of the State of Alagoas (FAPEAL).

References

[1] R. Britton, *MIPS: assembly language programming*. Prentice Hall, 2004. [Online]. Available: https://books.google.com.br/books?id=LB6y6TwcFWEC

[2] K. Vollmar and P. Sanderson, "Mars: an education-oriented mips assembly language simulator," in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 239–243. [Online]. Available: https://doi.org/10.1145/1121341.1121415

[3] J. R. Larus, “Spim s20: A mips r2000 simulator,” University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 1990.

[4] D. Patti, A. Spadaccini, M. Palesi, F. Fazzino, and V. Catania, “Supporting undergraduate computer architecture students using a visual mips64 cpu simulator,” *IEEE Transactions on Education*, vol. 55, no. 3, pp. 406–411, 2012.

[5] I. Branovic, R. Giorgi, and E. Martinelli, “Webmips: a new web-based mips simulation environment for computer architecture education,” in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, 2004, pp. 19–es.

[6] M. T. Kabir, M. T. Bari, and A. L. Haque, “Visimips: Visual simulator of mips32 pipelined processor,” in *2011 6th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2011, pp. 788–793.

[7] H. Sarjoughian, Y. Chen, and K. Burger, “A component-based visual simulator for mips32 processors,” in *2008 38th Annual Frontiers in Education Conference*, 2008, pp. F3B–9–F3B–14.

[8] H. S. Sarjoughian and B. Zeigler, “Devsjava: Basis for a devs-based collaborative m&s environment,” *Simulation Series*, vol. 30, pp. 29–36, 1998.

[9] J. L. H. David A. Patterson, *Computer Organization And Design*. Morgan Kaufmann, 2014.

[10] J. C. Penha, G. Fontes, and R. Ferreira, “Mipsfpga-um simulador mips incremental com validaçao em fpga,” *International Journal in Computer Architecture Education (IJCAE)*, vol. 5, no. 1, pp. 19–25, 2016.

[11] I. Green, *Web workers: Multithreaded programs in javascript*. ” O’Reilly Media, Inc.”, 2012.

[12] R. H. Jansen, V. Vane, and I. G. De Wolff, *TypeScript: Modern JavaScript Development*. Packt Publishing Ltd, 2016.