eduARM: Web Platform to Support the Teaching and Learning of the ARM Architecture

Maria Inês Alves[1], António Duarte Araújo[2] and Bruno Lima[3]
[1]*Faculty of Engineering, University of Porto, Porto, Portugal*
[2]*INESC TEC and Department of Electrical and Computer Engineering, Faculty of Engineering, University of Porto, Porto, Portugal*
[3]*INESC TEC and Department of Informatics Engineering, Faculty of Engineering, University of Porto, Porto, Portugal*

Keywords: Computer Architecture Teaching, ARM Architecture, Simulation, Learning Resources.

Abstract: Computer architecture is a prevalent topic of study in Informatics and Electrical Engineering courses, though students' overall grasp of this subject's concepts is many times hampered, mainly due to the lack of educational tools that can intuitively represent the internal behaviour of a CPU. With the evolution of the ARM architecture and its adoption in higher education institutions, the demand for this sort of tool has increased. Educational tools, specifically developed for the ARMv8 processor, are scarce and inadequate for what is necessary in an academic context. In order to contribute towards solving this problem, *eduARM*, a practical and interactive web platform that simulates how a ARMv8 CPU functions, was developed and is presented through this paper. Since this tool's main purpose is to aid computer architecture students, contributing to an improvement in their learning experience, it comprises varied concepts of computer architecture and organization in a simple and intuitive manner, such as the internal structure of a CPU, in both its unicycle and pipelined versions, and the effects of executing a set of instructions. As to better understand its value, the developed tool was then validated through a case study with the participation of computer architecture students.

1 INTRODUCTION

Computer Architecture (CA) is a fundamental subject in higher education technology courses, namely Informatics and Electrical Engineering. Among other topics, this course unit allows students to learn about the major internal subsystems of a computer, the general architecture of its platform, and assembly programming.

The Central Processing Unit (CPU) itself and its internal behaviour is a key study item, and one that students tend to exhibit difficulties in understanding. This problem is a result of the scarcity of intuitive educational tools that can graphically represent the CPU and the impact of executing a set of instructions (Nova et al., 2013). A most ideal platform would encourage CA students to learn through interactive experimentation, allowing them, for example, to thoroughly consult the CPU's datapath, in both its unicycle and pipelined versions, for each instruction in a set written in assembly language.

Another problem is raised with the evolution and popularity of the ARM architecture, specifically ARMv8. This is currently the most popular instruction set architecture (ISA) in the industry (Patterson and Hennessy, 2016), that is widely used in smartphones, laptops, and other embedded systems. Consequently, this caused certain higher education institutions to adapt their syllabus to include the teaching of this architecture, replacing other common architectures such as MIPS or RISC-V.

Similarly to what was already identified as a problem in the teaching of the most common processors (Nova et al., 2013), suitable platforms for teaching ARMv8 are scarce, which increases the need for developing an adequate software tool, capable of being used by both teachers and students during classes.

The *eduARM* web platform serves as an intuitive and interactive approach to learning the CPU and its behaviour, supporting both its unicycle and pipelined versions. The platform allows students to freely explore the CPU datapath in each stage of execution and understand how a specific instruction impacts its components' values, write simple assembly programs and debug them by inspecting the simulator's results, and check component latencies and the critical path of an

Alves, M., Araújo, A. and Lima, B.
eduARM: Web Platform to Support the Teaching and Learning of the ARM Architecture.
DOI: 10.5220/0011712800003470
In *Proceedings of the 15th International Conference on Computer Supported Education (CSEDU 2023) - Volume 2*, pages 341-348
ISBN: 978-989-758-641-5; ISSN: 2184-5026

instruction.

Following the present section, this document includes a review and discussion of related work on tools for CA education in Section 2, followed by an overview of the implementation process behind the *eduARM* platform in Section 3, and a description of its user interface in Section 4. The validation process is detailed in Section 5, and, to finalize, conclusions are disclosed in Section 6.

Although valuable projects in this domain were found, showcased in the State of the Art, none are capable of completely responding to this problem. Nonetheless, these tools' good practices and features presented a solid starting point for the development of the *eduARM* web platform, an intuitive and interactive approach to teaching and learning of the ARMv8 architecture.

2 STATE OF THE ART

Throughout this section, several tools dedicated to CA education are presented, their various features are discussed, as well as their positive and negative aspects. Finally, a comparison is made between the presented tools, highlighting which features were considered the most important for the platform and what contribution its development can give to the field.

This section focuses solely on open-source tools dedicated to RISC architectures, namely MIPS, RISC-V and ARM, as these make up a much less complicated processor compared to CISC architectures (Gupta and Sharma, 2021), and are thus commonly adopted in CA courses.

SPIM is a self-contained MIPS simulator that runs assembly language programs, considered the most widely known and used MIPS simulator (Larus, 1990), both for education and the industry. Although *SPIM* can be used for debugging assembly programs, in order for the tool to thoroughly support the teaching of CA, CPU datapath visualization is missing.

WebMIPS is an educational web-based MIPS simulation environment written in the ASP language. This tool is a five stage MIPS pipeline simulator (Branovic et al., 2004) and solely supports the MIPS architecture and the pipeline version of the CPU, which poses a great disadvantage. The platform can thus be seen as outdated and easily surpassed by other similar, more recent tools.

MARS, the Mips Assembly and Runtime Simulator (Vollmar and Sanderson, 2006), was designed as an alternative to *SPIM*, tackling most of its shortcomings and greatly outperforming it. Despite being a robust and useful tool for assembly debugging, *MARS* has no CPU datapath visualization, no support for pipelined architectures and is not available on the Web.

DrMIPS is an open-source graphical simulator of the MIPS processor, specifically designed for teaching and learning CA (Nova et al., 2013). While this project provides students with a robust and highly intuitive tool that includes fundamental principles lectured in CA, it exclusively supports the MIPS architecture, preventing its adoption in more recent higher education courses. Nevertheless, its focus on education and multitude of functionalities and visualization make *DrMIPS* a valuable platform whose vision served as a model for the tool presented in this paper.

RARS, or RISC-V Assembler and Runtime Simulator (Landers, 2017), is a direct port of the MIPS simulator *MARS*. *RARS*, much like its MIPS counterpart, focuses intensively on assembly debugging, containing essentially the same features and also lacking CPU datapath visualization and support for pipelined architectures.

The *BRISC-V Simulator* is a browser-based assembly programming simulator, which, together with *BRISC-V Explorer*, makes up the *BRISC-V Platform* (Agrawal et al., 2019). Similarly to several of the already presented educational platforms, *BRISC-V* is more focused on assembly debugging and does not include a display of the CPU datapath, as well as lacking support for a pipeline version of the CPU.

The *BRISC-V Explorer* is an educational tool for exploring CPU design, allowing the creation of single or multi-core RISC-V processors. As the *BRISC-V Explorer* provides a platform for CPU architecture design, this can be helpful for students to understand its components on a deeper level. Since this tool is more focused on architectural design, it is better suited for more advanced CA classes (Agrawal et al., 2019).

VisUAL is an ARM emulator developed as a cross-platform tool for ARM education, particularly ARMv7 (Arif, 2015). Although this tool is not as focused on the CPU and its internal behaviour, and more on assembly debugging, it includes multiple educational features that are useful for students and can be taken into consideration while designing a new tool. *VisUAL* only supports ARMv7, which was already replaced by ARMv8 in certain institutions, not being suitable for aiding those students any longer.

The *Graphical Micro-Architecture Simulator*, also called simply *LEGv8 Simulator*, is a web based ARMv8-A ISA simulator, which, albeit still in BETA version, delivers a complete and interactive environment for CPU visualization, both its unicycle and pipelined versions (ARM, 2021). Despite being a very complete platform, this simulator lacks certain

important features for education, such as a data memory display, visualization of input and output values in each component, and a more complete register file. Component latencies and the critical path are also not included in the platform.

WepSIM is a modular and intuitive online educational simulator of the CPU, supporting both MIPS and RISC-V architectures (García-Carballeira et al., 2019). Being available on the web, the platform is highly portable and can be run from any web browser or from a text-based command line. *WepSIM*, albeit a complete platform, can be too dense or complex, and does not support a pipeline version of the CPU nor implementation of latencies and the critical path.

CREATOR is a generic web-based simulator for assembly programming developed by the same authors as *WepSIM* (Camarmas-Alonso et al., 2021). Unlike its older counterpart, *CREATOR* does not include a display of the CPU datapath, as it is a tool more focused on assembly programming instead of CPU visualization and configuration.

In order to efficiently analyse these existing tools and what a new platform should include, a few key requirements were established. The educational platform should be available on the Web, through the browser, removing the need for downloads and making it accessible to anyone regardless of operating systems or platform specifications. The tool should allow its users to visualize and examine the CPU datapath and provide an area for assembly programming, so students can write their own code, debug it, and explore its effects on the CPU. Both the unicycle and pipelined versions of the processor should be supported, as well as latencies and the critical path, for students to analyse the CPU's performance. Finally, the platform should support ARMv8, considering educational tools for this architecture are scarce, as explained previously in Section 1.

To evaluate whether these tools would meet all the requirements established for a complete and suitable platform to teach the ARM architecture, table 1 is provided.

Despite each tool having its strengths and useful features, none can satisfy all the requisites proposed. More specifically, even though all of them can teach assembly programming, only a few also focus on the CPU and its datapath. Most lack support for both the unicycle and pipeline versions of the processor, as well as latency and critical path implementation.

Seeing as none of the works presented can completely respond to the problem identified, even more so with most of them lacking support for ARMv8, a platform that can aggregate the positive features of these tools and provide what they are lacking would be the ideal solution. It would also present a source of innovation in the field, which is what this project aims to achieve, and whose implementation is thoroughly explained in the next section.

3 IMPLEMENTATION

3.1 Requirements Specification

The development of this educational platform must take several CA concepts into account, handling the multiple topics students learn in classes. After analysing the state of the art and the functionalities of existing educational tools, the requirements for a new platform were specified as follows:

- Must be simple and intuitive, so students can easily interact with the platform and understand its concepts, avoiding unnecessary elements or visual clutter
- Represent the CPU datapath in both its unicycle and pipelined versions, which the user can interchange accordingly
- Provide an interface for assembly coding, so students can write their own instructions and explore what happens in the CPU with their execution, showing detailed information on all data and control signals, as well as display how these instructions are encoded in machine code
- Execute a set of instructions provided step-by-step, allowing the user to forward or go back to a certain instruction and understand what its effects on the CPU are
- Allow visualization of registers and data memory contents, as well as latencies and the CPU critical path
- Be available on the Web, hence providing easy access to anyone, regardless of their operating system or computer specifications

Each of the requirements above was taken into consideration while planning for the platform's development, ensuring that the finished product would accomplish all objectives and adequately respond to CA students' needs.

3.2 Development Process

After the platform's requirements specification, the first step of development was to ensure it would run on the web. A simple web app was created, and, within it, two distinct layers were conceived - the frontend, encapsulating all user interface-related work and

Table 1: Comparison of requirements met by the tools.

	Web	CPU datapath	Assembly	Unicycle & pipeline	Latency & critical path	ARMv8
SPIM	No	No	Yes	No	No	No
WebMIPS	Yes	Yes	Yes	No	No	No
MARS	No	No	Yes	No	No	No
DrMIPS	No	Yes	Yes	Yes	Yes	No
RARS	No	No	Yes	No	No	No
BRISC-V	Yes	Yes	Yes	Yes	No	No
VisUAL	No	No	Yes	No	No	No
LEGv8Sim	Yes	Yes	Yes	Yes	No	Yes
WepSIM	Yes	Yes	Yes	No	Yes	No
CREATOR	Yes	No	Yes	No	Yes	No

client-side operations, and the backend, handling the server and all CPU simulation logic. Connecting these segments allows the web platform to run in its entirety, with each request sent by a user's actions being received and handled server-side.

3.2.1 Simulation Logic

Having established the foundation for both layers, the program's backend was the first to be thoroughly implemented. To accurately simulate the CPU, each of its components was simulated individually. In order to achieve this, a JavaScript class was implemented for every component, storing inputs, outputs, latency, and its own distinct execution behaviour, which will run when that component is scheduled to operate.

The order in which every component operates is defined through a JSON file, where all CPU elements are represented through objects. This configuration file is then used in another JavaScript class, called "CPU", responsible for establishing the connections between the various defined components and ultimately executing a program. This class encapsulates all simulation logic, containing a method, "execute", that given an instruction's machine code, will compute the execution method of each component sequentially, in the order defined by the JSON file. The results of a CPU component's operation are thus propagated to its connected components, and, if the instruction changes registers or writes to memory, the CPU's register file and data memory are updated.

For students to understand exactly how much time the unicycle processor takes to execute a set of instructions, latencies were implemented into the platform. Additionally, the platform highlights the critical path of each instruction, defined as the sequence of components that, altogether, take the longest to execute.

The same implementation method was applied to the CPU's pipeline version, which required the addition of pipeline registers, four in total, that act as an intermediary between two pipeline stages, storing values between cycles. Besides this, forwarding was implemented into the platform, with the addition of a forwarding unit capable of handling dependences between instructions, as well as a hazard detection unit that solves more complicated hazards that require pipeline stalling.

As the CPU simulation logic implemented is significantly heavy on operations, running its calculations client-side would overload the user's machine and thus be extremely inefficient. Having a dedicated web server that runs the simulation and handles all requests made on the frontend was the approach selected to solve this problem, with the creation of an application programming interface (API) able to communicate with the frontend layer and answer its requests, receiving, altering, and sending back data.

4 USER INTERFACE

The application's user interface was designed with the objective of providing students with a user friendly and interactive environment for studying. In order to achieve this, the interface was kept mostly simple, avoiding any visual clutter and the addition of too much information, which could end up confusing students and ultimately doing more harm than good. The finished product's homepage is shown in Figure 1.

The platform's most prominent element is the CPU datapath, on its left side, with its components and corresponding connections. In this visual representation, control lines and related components are colored blue. On top of the diagram, two more tabs besides the CPU one can be seen: "Assembly" and "Machine Code". The former is dedicated to writing assembly code, while the latter displays each instruction's machine code after compiling a program.

On the screen's rightmost area, the content of each

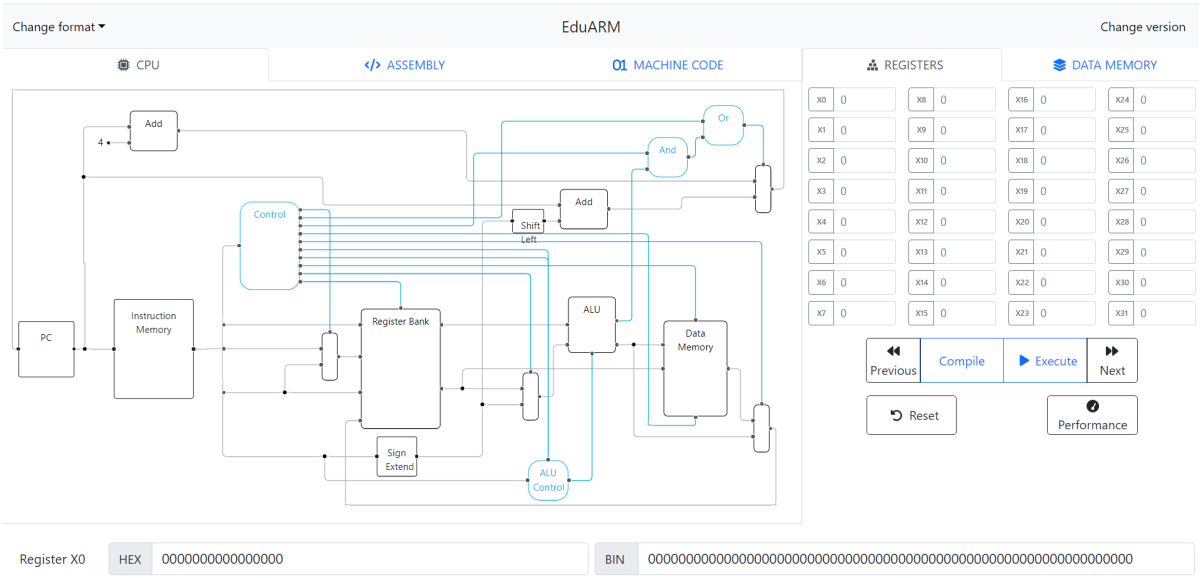

Figure 1: Homepage of *eduARM*.

register, from X0 to X31, is displayed. By changing a register's value, its hexadecimal and binary representations will also be displayed in the two fields at the bottom of the page, always on their 64-bit format. The content of these can also be filled in by the user, and the corresponding conversions will be made to the register file.

After a user writes the code and changes the necessary registers, the program is ready to be executed. The user can press "Compile" and the machine code of each of the program's instructions will appear in the "Machine Code" page, along with their position on the instruction memory and their assembly code. In the case of the platform identifying any compilation errors, these will appear as an alert on top of the page, and the program will not proceed unless these are fixed.

Having successfully compiled the program, the user can then proceed by clicking the "Execute" button, and the datapath will showcase, for each instruction, the internal behaviour of the CPU. Any changes to data memory, in the case of executing memory access instructions, can be checked in the "Data Memory" tab, next to "Registers".

After execution, the datapath will display the state of the last instruction written. The user can freely cycle through instructions by using the "Previous" and "Next" buttons next to "Execute". Below this area, the current instruction's machine code and its different fields are presented, also updating when cycling through instructions. The instruction's assembly code is also displayed in this area and highlighted on the code written in the "Assembly" tab.

Connections that are considered relevant for each instruction, that is, that carry any sort of relevant data, are painted black. The various CPU components can be hovered over in order to see their IDs, latencies, inputs and outputs' current values. These values can also be converted to either their binary or hexadecimal formats by selecting the desired representation with the "Change format" button on the top left side of the screen.

Additionally, two buttons entitled "Reset" and "Performance" are located below the execution controls. The former will reset the program for the user, setting all registers and memory values to zero and reverting the datapath to its original state. "Performance" grants access to the CPU's critical path, whose connections and components will be highlighted red.

By connecting both the simulation logic and the user interface, the platform is able to execute simple assembly programs and provide the corresponding datapath visualization. This includes the results of the assembly operations on the registers and the data memory, as well as the input and output values of each CPU component in that state.

The button "Change version" in the top right corner of the page can be used to change between the unicycle and pipelined versions of the CPU. While the unicycle is the one chosen as default, the user can press this button to switch to the pipeline version, presented in figure 2.

Five pipeline stages are represented with colors on top of the datapath. Unlike the unicycle version, a CPU state will represent the five stages in a clock

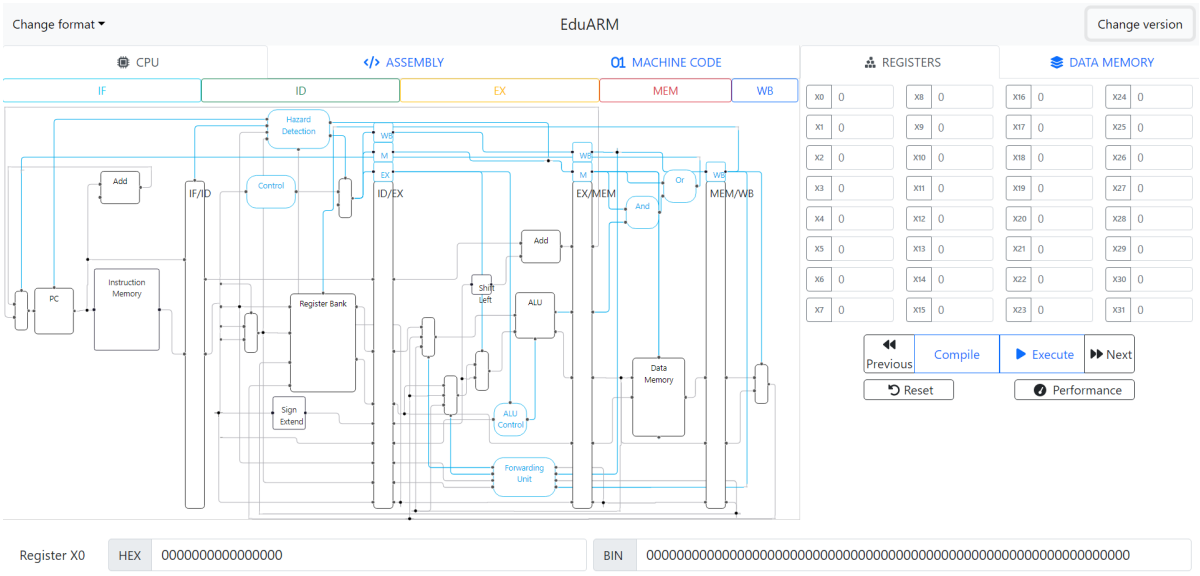

Figure 2: Pipeline datapath of *eduARM*.

cycle, which could have five instructions at the same time. Because of this, the number of steps is now five times higher, and the ”Previous” and ”Next” buttons do not iterate through instructions, but rather through clock cycles. Which instruction is in each stage can be seen in the area below the execution buttons.

Regarding forwarding and hazard detection, when either unit identifies a problem, their datapath component will be highlighted purple. If a stall occurs, the processor will not do any relevant operations and a ”bubble” is added to the instruction flow.

This section closes the process behind the implementation of the platform, which was then tested by a group of students through a case study, whose results and discussion are presented in the next chapter.

5 VALIDATION

The developed platform, in order to completely fulfill its intended objectives, must be validated by its intended audience. This audience, in the case of *eduARM*, is comprised of Informatics and Electrical Engineering students who are learning CA. It is only possible to understand the true value of the platform when it is tested by students.

The method deemed the most appropriate was validating the platform through a survey, sent by e-mail to former CA students of the Faculty of Engineering of the University of Porto (FEUP). A total of 15 students participated in this study, with 9 of them also providing qualitative feedback and reporting bugs.

The survey sent to students opens with a small contextualization of the work and explanation of the study's objectives, being followed by a total of 17 questions grouped into three different sections.

The first group of questions relates to the background and general vision of the students on the CA course. The purpose of this survey section was attempting to understand what difficulties students felt during the course, and how challenging the CPU and its datapath are to comprehend.

Its first question attempts to understand how students perceive the CA course by asking them to measure its difficulty. The majority of participants (53,4%) claim the course has either a difficulty level of 6 and 7 out of 10, as shown in Figure 3, which implies the course unit is considered mildly challenging.

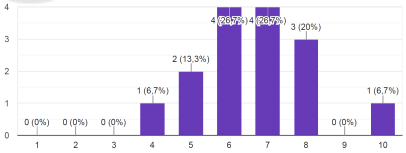

Figure 3: Results of Question 1.

The second question asks students to evaluate how difficult the CPU's concepts and behaviour are to understand. As seen in Figure 4, results obtained were diverse, similarly to question 1, with most participants (80%) placing this subject's difficulty level between 6 and 8, thus implying the majority of CA students found the CPU rather difficult to understand.

The final question in the form's first section asks students whether they believe having access to a platform capable of simulating the CPU's behaviour

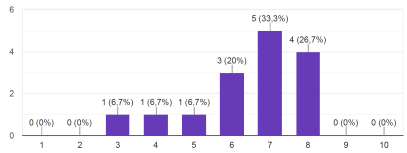

Figure 4: Results of Question 2.

would have helped them better understand these topics, with 100% choosing the option "Yes".

The next survey section includes an exercise with eight questions about the execution of a provided assembly program, that should be solved using the platform. The results obtained in this part are not as significant, since they mostly evaluate whether the student chose the right answer or not, which could depend on a variety of factors. Rather, the questions presented in the exercise are merely intended to allow the students to fully explore the platform's features.

The form closes with an area for students to offer their overall feedback, both quantitative and qualitative.

The first question of this feedback section, question 12, asks students to quality the platform regarding its usability and how intuitive it is. On a scale of 1 to 5, results were considered positive when equal to 4 or above, which was the case of Question 12, with 80% of the participants giving a 4 or the maximum score of 5 to the interface, as seen in Figure 5.

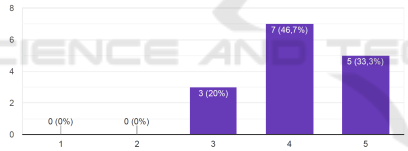

Figure 5: Results of Question 12.

The next question's purpose is to understand exactly how useful the platform is for CA classes, asking students to measure how much they think having access to the tool would have improved their comprehension of the lectured topics. The scores obtained were, once again, very positive, as 80% of the students chose the highest option, as shown in Figure 6.

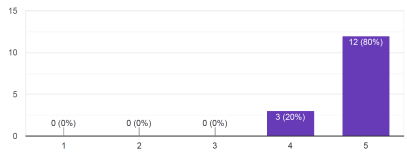

Figure 6: Results of Question 13.

Afterward, question 14 aims to analyse how this platform would make learning more enjoyable for students. Most students (93,4% chose either option 4 or 5) seem to believe the platform would greatly improve their learning experience, as illustrated through Figure 7.

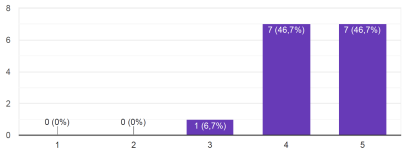

Figure 7: Results of Question 14.

The next question asks students to measure how useful the platform would be for their studying outside the class. The majority of the participants agree that having access to this platform would benefit their studying, with 86,7% selecting either option 4 or 5, as seen in Figure 8.

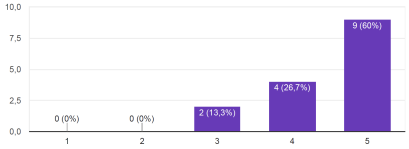

Figure 8: Results of Question 15.

Question 16 inquires how much the students think having access to the platform would have improved their final CA grade, when they frequented the course. Each student's response will naturally be influenced by how low or high their final grade was - nevertheless, more than half (60%) of the answers were positive, with scores of either 4 or 5, as shown in Figure 9.

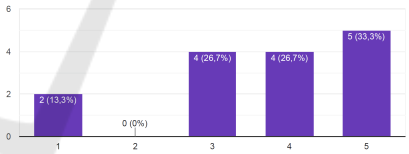

Figure 9: Results of Question 16.

The final question of the survey is very straightforward, simply asking the participants whether they would use the tested platform, were it available when they had the CA course unit. Results were extremely positive, with 100% of the students selecting option "Yes".

5.1 Threats to Validity

Since the form was sent during summer break, and the study was already limited to a specific group of students, adherence was low. Nonetheless, student feedback was highly positive, with the participants finding the platform adequate and useful for CA classes. These results could be further improved if student ad-

herence to the survey was higher, whose low values were a consequence of the timing chosen for validation.

Another important aspect that would reinforce the platform's validation is the inclusion of the pipelined CPU version. Due to this version not yet being completed at the time of user testing, the survey only contemplates the unicycle version of the CPU, which limits the validity of the platform, as the pipelined version is a key part of the project and essential for understanding the CPU's more complex behaviour. Nevertheless, testing the unicycle brought forth valuable feedback on how to improve the interface, which helps towards refining the platform to become its best possible version.

6 CONCLUSIONS

The platform described throughout this paper aspires to be of use to both computer architecture teachers and students in the academic community, as well as presenting a source of innovation in the field of ARMv8 educational tools capable of simulating the CPU.

Considering the use of simulators is believed to greatly improve students' understanding, this work expects to aid in the comprehension of subjects that would otherwise be difficult to understand through more conventional teaching methods. Thus, the developed platform offers an interactive approach to learning the CPU's components and overall behaviour, for both its unicycle and pipelined versions.

The successful adoption of this platform in CA classes would further emphasize that the goals of this project were accomplished, and that CA students have access to an intuitive and adequate platform capable of helping them further comprehend the topics lectured in classes and, ultimately, positively contributing to their academic success.

REFERENCES

Agrawal, R., Bandara, S., Ehret, A., Isakov, M., Mark, M., and Kinsy, M. A. (2019). The brisc-v platform: A practical teaching approach for computer architecture. In *Proceedings of the Workshop on Computer Architecture Education*, pages 1–8.

Arif, S. (2015). Visual - a highly visual arm emulator. https://salmanarif.bitbucket.io/visual/index.html.

ARM (2021). Graphical micro-architecture simulator. https://www.arm.com/zh-TW/resources/education/education-kits/legv8.

Branovic, I., Giorgi, R., and Martinelli, E. (2004). 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*, pages 19–es.

Camarmas-Alonso, D., García-Carballeira, F., Del-Pozo-Puñal, E., and Mateos, A. C. (2021). A new generic simulator for the teaching of assembly programming. In *2021 XLVII Latin American Computing Conference (CLEI)*, pages 1–9. IEEE.

García-Carballeira, F., Calderón-Mateos, A., Alonso-Monsalve, S., and Prieto-Cepeda, J. (2019). Wepsim: an online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming. *IEEE Transactions on Learning Technologies*, 13(1):211–218.

Gupta, K. and Sharma, T. (2021). Changing trends in computer architecture : A comprehensive analysis of arm and x86 processors. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 619–631.

Landers, B. (2017). Rars - risc-v assembler and runtime simulator. https://github.com/TheThirdOne/rars.

Larus, J. R. (1990). Spim s20: A mips r2000 simulator. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Nova, B., Ferreira, J. C., and Araújo, A. (2013). Tool to support computer architecture teaching and learning. In *2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)*, pages 1–8. IEEE.

Patterson, D. A. and Hennessy, J. L. (2016). *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann.

Vollmar, K. and Sanderson, P. (2006). Mars: an education-oriented mips assembly language simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 239–243.