An Interactive Visualization Tool for Computer Organization and Design Course

Mateo Garcia*, Quamar Niyaz*, Xiaoli Yang†, Ahmad Y. Javaid‡, Sidike Paheding†
*ECE Department, Purdue University Northwest, Hammond, IN, United States
†CSE Department, Fairfield University, Fairfield, CT, United States
‡EECS Department, The University of Toledo, Toledo, OH, United States
{garci531, qniyaz}@pnw.edu, {spaheding, xyang}@fairfiled.edu, ahmad.javaid@utoledo.edu

2024 IEEE International Conference on Electro Information Technology (eIT) | 979-8-3503-3064-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/eIT60633.2024.10609897

***Abstract*—Computer Organization and Design is a pivotal course for learning an instruction set architecture (ISA), the microarchitecture which implements it, and determining a system's overall performance. However, many concepts in this course have been found daunting to understand. To illustrate those concepts, an interactive desktop application is being developed using the *Unity* game engine for the purpose of enhancing student learning. This tool covers material that is traditionally covered in the course and adds security concepts to make students aware of the security issues associated with micro-architecture.**

***Index Terms*—Computer Organization and Design, visualization-based learning tool, and cybersecurity**

I. Introduction

Computer Organization and Design (COD) is one of the most pivotal courses in a Computer Engineering/Science student's curriculum. Most students will get their first introduction to the intricacies of computer design and implementation in this course. The course covers essential low-level computing topics such as hardware functionality, instruction set architecture (ISA), and CPU design/performance. Amongst other things, the knowledge of how computer software and hardware interact, how high-level programming language is translated into low-level machine language, and how a computer interprets the machine language, all serve as foundational knowledge that computing students will continue to build on. Despite the importance of COD course, it is commonly considered to be one of the most difficult courses in the Computer Engineering/Science curriculum. This is because most students will only have an abstract idea of how computers operate. The transition from high-level knowledge to the "nuts and bolts" of how computers work is a difficult one to make. As such, many students struggle with learning the topics covered in this course.

Along with the traditional microarchitecture concepts, microarchitecture security in the COD course must be a new addition. The rising danger of cybercrime cannot be overlooked and the global shift towards the digital medium has only magnified the attack potency of cybercriminals. Global cybercrime costs will soar to an estimated $10.5 trillion (annually) by the year 2025 [1]. There is significant shortage of cybersecurity workforce that can be attributed to the lack of exposure to cybersecurity in standard computing curriculum. An increased effort should be put forth towards raising student awareness of this matter. It is especially important to include a security component in the COD course because while certain attacks exploit software vulnerabilities (e.g., buffer overflow) there exist attacks which exploit the very microarchitecture which computers are based on. These types of attacks are even harder to defend against because they exploit vulnerabilities in the fundamental design of a computer. It is imperative that computing students are knowledgeable of these vulnerabilities as they are the next generation of developers whose work will determine the security of future digital assets. Introducing students to security concepts in the COD course may even motivate them to pursue a career as a full-fledged cybersecurity professional, which can be considered a "win" given the aforementioned shortage.

Most computing courses follow the STEM methodology of utilizing "hands-on" assignments, exams, and a culminating project to achieve student learning outcomes. While this has been a tried-and-true approach, many students still experience hindered learning. Visualization is the missing ingredient in traditional instruction methods and is a proven technique to facilitate student learning. Studies have shown the use of visualization has a positive impact on student motivation for learning in Computer Engineering and Science courses [2], [3]. The benefits of visualization methods has been demonstrated even outside pure computing curriculum [4]. Positive opinions about visualization are not exclusive to students but are shared with instructors alike.

After identifying the struggles of students enrolled in the COD course, the need for an effective supplementary approach to present the course materials became apparent. [2]–[4] validate visualization as a legitimate method to improve student learning and prove it effective when combined with some level of interactivity. With this motivation, this work presents an interactive visualization-based learning tool that is being developed specifically for the COD course. The tool contains visualization-based learning modules that will cover the concepts traditionally taught in a COD course and introduce micro-architecture security concepts.

The remainder of the paper is structured as follows. Section II will discuss related work related to the development of educational tools for teaching computing concepts. Section III will briefly discuss the topics that will be covered in the proposed tool. Section IV is dedicated to the design and implementation of the visualization tool. Section V will conclude the paper with future work and final remarks.

II. Related Work

In this section, we outline a few recent works carried out for the development of educational tool for COD or similar courses. An educational tool, DrMIPS, was developed in [5] to support students in a computer architecture course. This tool simulates the well-known MIPS processor and allows users to create or load assembly programs and simulate execution. In a step-by-step visualization the user can observe the contents of the processor's registers and data memory. The tool includes both single-cycle and pipelined processors. Another work focuses on the impact the EduMIPS64 simulator had on a different computer architecture courses at the University of Catania [6]. This simulator served as the primary means of learning topics, completing homework assignments, and taking the final exam. EduMIPS64 can simulate instruction execution in a pipeline processor by drawing different pipeline stages in sub-windows. Thesis work done at the University of Illinois introduces the Datapath Simulator [7], which is described as "a tool for visually teaching the fundamentals of hardware architecture." The Datapath Simulator is a web-based program written in JavaScript and uses the React library to display the content in the user's browser. The simulator is built around the MIPS architecture and supports execution of the MIPS32 ISA on simulated circuits.

A web-based tool, eduARM [8], focuses on the increasingly popular ARM ISA. It was designed to simulate the functionality of the unicycle and pipelined ARMv8 CPU. Users can create their own assembly code and observe what occurs in the processor during execution. WebRISC-V [9] is another web-based application that focuses on the RISC-V architecture however it exclusively simulates pipelined processors. The front-end is implemented with HTML and JavaScript while the backend is implemented with PHP.

III. Overview of Covered Topics

The related works have demonstrated great success in visualization-based instruction of COD concepts. This project offers a new contribution to the genre with the development of a Unity desktop application as a supplemental educational resource for the *ECE 37100* (COD) course at Purdue University Northwest. The tool intentionally visualizes concepts with a level of abstraction. This approach was taken to make the tool more approachable (which increases the likelihood of students actually using it). This design approach does not detriment the tool because the tool is a supplemental resource by nature. It is not intended to replace the instructor, the course textbook or other course resources in any capacity. In correlation to the course, the tool focuses on the ARM Instruction Set Architecture (ISA) and the associated microarchitecture. Current topics include the ARM ISA (arithmetic and memory access instructions) and CPU pipelining. Strong reference was made to the COD textbook, *Digital Design and Computer Architecture* [10] during the development of the two modules covering these topics. As previously mentioned, the visualization tool will also contain learning modules for microarchitecture security concepts. A module for the Spectre [11] attack is currently in development and current plans also seek to develop a module for Meltdown [12]. Besides the ones aforementioned, the concepts covered in the tool will continue to expand in future works.

A. ARM Instruction Set Architecture

Arithmetic Operations: The arithmetic logic unit (ALU), as the name suggests, performs arithmetic operations. The ALU uses instructions such as ADD, SUB, logical AND, and logical ORR. The ALU strictly operates between registers. The basic syntax of these instructions is `<operation> Rd, Rn, Operand2`. Rd is the destination register where the result of the ALU operation will be stored. Rn is the first source register where data is retrieved from and Operand2 is optionally used. For example, the instruction `ADD R1, R2, R3` would be understood as content of registers R2 and R3 will be added and stored in register R1.

Memory Access: LDR and STR are the instructions used for memory access. These instructions have the basic format of `<operation> Rd,[<address in Rn>]`. Despite similar syntax, LDR and STR differ in how they function. The instruction `LDR R1, [R2]` would be understood as the content from the memory location specified by register R2 will be stored in register R1. On the other hand, `STR R1, [R2]` would be understood as the content of register R1 will be stored at the memory location specified by register R2. Memory access instructions prove especially challenging for students to understand. This is because unlike ALU operations, which only operate between registers, memory access instructions operate between registers and memory. Visualizing data transfer between registers and memory will prove to be especially beneficial for students.

B. Pipelining

Pipelining significantly enhances the processor's performance in comparison to its single-cycle implementation, which can only perform one instruction per clock cycle. Pipelining allows for the parallelization of instruction execution and greatly improves throughput. To achieve this an instruction traverses through different pipelining stages, e.g. Fetch, Decode, Execute, Memory, and Writeback in a five-stage pipeline. In the Fetch stage, the instruction is read from instruction memory by the processor. In the Decode stage, the processor reads the operands from the register file and prepares the control signals. In the Execute stage, the ALU operation is performed. In the Memory stage the processor reads/writes data memory. Finally, in the Writeback stage the result is written to a register (when necessary). Note that while multiple instructions can be simultaneously executed, different instructions will be in different pipelining stages.

Pipeline hazards come at the cost of the increased performance. There are three types of hazards including data, control, and structure hazards. A data hazard occurs when a sequence of instructions requires operands to be accessed in a certain order. A distinction is made for the Read-After-Write (RAW) data hazard which occurs when an instruction reads an operand before a previous instruction was able to write to it. Control hazards occur when the CPU arrives at a branching instruction and is forced to make a branch decision before the decision is ready to be made. In the structure hazard, there is a conflict to access resources. The visualization module will cover the hazards and the methods used to handle them. In the pipelining module, we specifically focus on the use of `NOP` instructions, data forwarding, stalling, and flushing.

The use of `NOP` instruction to resolve data hazards is a software approach that forces the CPU to remain idle until the data is ready to be read. Students are made aware of the fact that this approach complicates programming and severely degrades performance. With Forwarding, the pending data is forwarded from the Memory or Writeback stage to a dependent instruction in the Execute stage. Stalling involves halting operation until the pending data is ready. This approach is superior to NOP because a new instruction can enter the pipeline and the CPU is not kept at idle. Flushing is a control hazard management method in which the CPU makes a branch prediction instead of idling until the branch decision is ready. The CPU loads sequential instructions into the pipeline based on its branch prediction. In the case of an incorrect branch prediction, all the executed branch instructions would be discarded.

C. Microarchitecture Security

Recently, several micro-architectural attacks were discovered. For example, the Spectre attack [11], discovered by Google's Project Zero in 2018, takes advantage of the speculative execution feature (intended as a performance optimization) to execute malicious code. A CPU with the speculative execution feature preemptively performs tasks to have data ready in the case of it being useful in the future. Intel, Apple, ARM, and AMD have produced CPU which are vulnerable to this attack. The Meltdown attack affects Intel and Apple processors and, similar to the Spectre attack, it exploits speculative execution. Unlike Spectre, which can only gain unauthorized access to data within the same program, the Meltdown attack [12] can expose data from separate programs. It is important that these attacks should be discussed in a COD course. With this focus, we intend to include a visualization of such attacks in our tool.

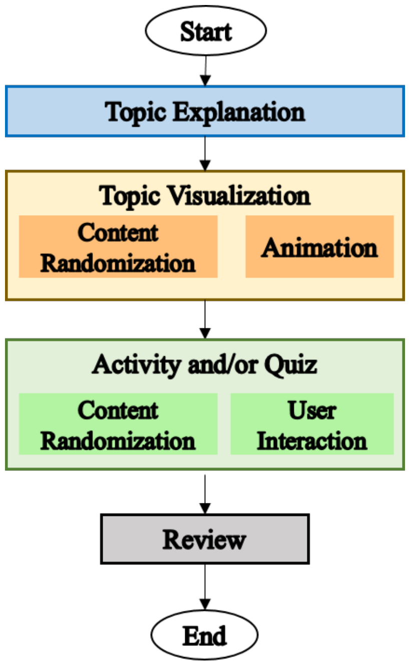
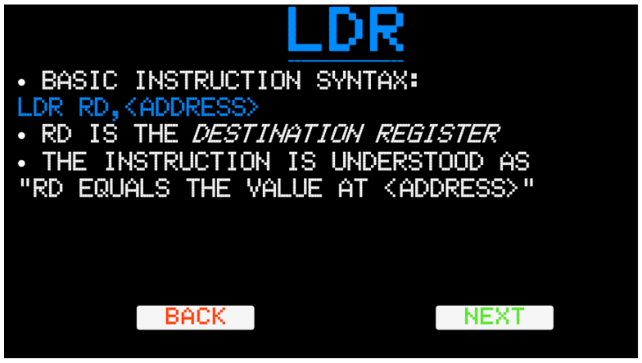


Fig. 1: Flow chart of modules interaction in the tool

IV. Design and Implementation

The tool was built following a modular design. A modular design is defined by the separation of distinct components. Each module present in this tool has a distinct concept. This design approach is beneficial both from a user experience and development perspective. This design allows the user to focus on a specific concept of their choice, as opposed to having multiple concepts grouped together. The modules are also concise thanks to this design, which shortens play-through time. This ideally not only conveniences the user but also has the benefit of speeding up their learning process. On the same token, the conciseness of the modules also speeds up the development process. A single-concept module can be developed faster in comparison to a multi-concept module (assuming the same amount of content for each concept). The debugging process is simplified by organizing the contents in a modular fashion, which makes it easier to locate the issue. In addition, the total impact of revising the tool is lessened as revising one module uniquely changes that module and does not affect other modules. This decreases the risk typically associated with updating software over time.

All modules follow the same design principles. Modules will first begin with providing conceptual information. To avoid becoming a text-heavy PowerPoint-like presentation, the modules provide concise explanations of concepts. These learning modules serve as a supplemental resource to enhance the utility of the course textbook, not replace it. The conceptual information is followed by a visualization of the concept. The visualization strives to keep things simple. Many students in the course will just be getting acquainted with certain concepts and excessively complex visualizations can potentially be detrimental. As mentioned in [13], the



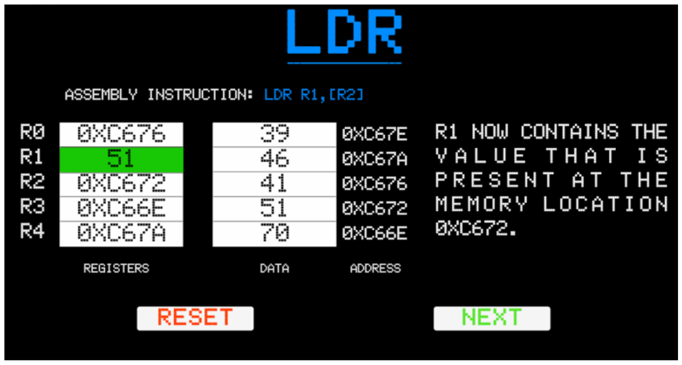
LDR

- BASIC INSTRUCTION SYNTAX:

LDR RD,<ADDRESS>

- RD IS THE *DESTINATION REGISTER*
- THE INSTRUCTION IS UNDERSTOOD AS "RD EQUALS THE VALUE AT <ADDRESS>"

a. Overview of LDR



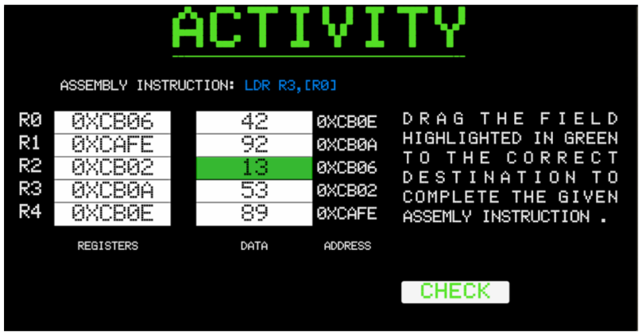
LDR

ASSEMBLY INSTRUCTION: LDR R1,[R2]

	REGISTERS	DATA	ADDRESS
R0	0XC676	39	0XC67E
R1	51	46	0XC67A
R2	0XC672	41	0XC676
R3	0XC66E	51	0XC672
R4	0XC67A	70	0XC66E

R1 NOW CONTAINS THE VALUE THAT IS PRESENT AT THE MEMORY LOCATION 0XC672.

b. Visualization for LDR



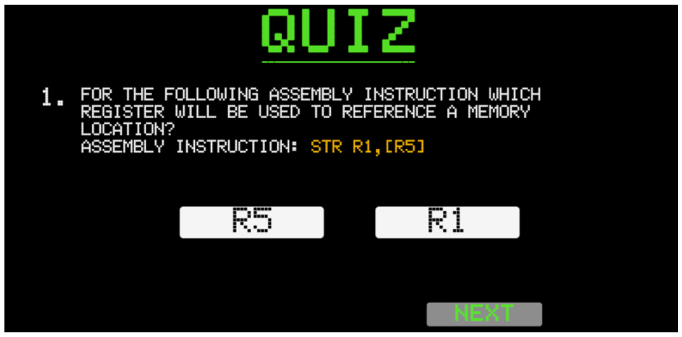
ACTIVITY

ASSEMBLY INSTRUCTION: LDR R3,[R0]

	REGISTERS	DATA	ADDRESS
R0	0XCB06	42	0XCB0E
R1	0XCAFE	92	0XCB0A
R2	0XCB02	13	0XCB06
R3	0XCB0A	53	0XCB02
R4	0XCB0E	89	0XCAFE

DRAG THE FIELD HIGHLIGHTED IN GREEN TO THE CORRECT DESTINATION TO COMPLETE THE GIVEN ASSEMLY INSTRUCTION .

c. Activity for LDR



QUIZ

1. FOR THE FOLLOWING ASSEMBLY INSTRUCTION WHICH REGISTER WILL BE USED TO REFERENCE A MEMORY LOCATION?
ASSEMBLY INSTRUCTION: STR R1,[R5]

d. Quiz for LDR

Fig. 2: Sample visualization for LDR instruction

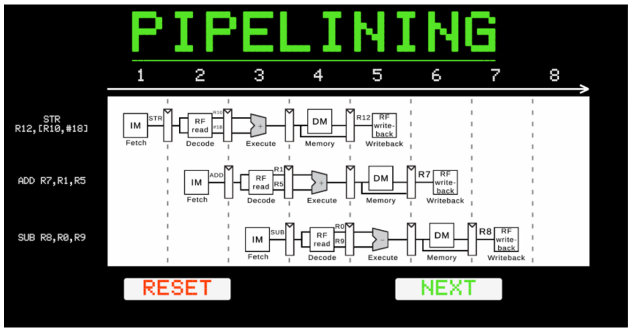
benefit of visualization-based learning is enhanced by interactivity. This comes in the form of activities related to the visualizations, and/or multiple-choice quizzes. The quizzes, taken at the end of each module, are a great method for students to assess their understanding of the concept. The quizzes are graded, and students can review their answers after submission. A great feature of all modules is the randomization of the content within visualizations, activities, and quizzes. This feature ensures a unique experience upon each use of the tool and avoids (or at least delays) user fatigue.

The tool was built using Unity, a powerful cross-platform game engine. It supports the development of 2D, 3D, virtual reality (VR), and augmented reality (AR) based games. Unity's potency as a game engine and ability to distribute to a variety of platforms made it a clear choice for being used to develop this visualization tool. In addition the lead designer of this tool has a strong familiarity with Unity. The Unity game environment is hierarchical in nature. At the very top of this hierarchy is the `Scene`. Every Unity project has at least one Scene, which contains the actual game environment. Multiple Scenes can be used for both organizational and functional purposes to do things such as change character or change the level of the game. Each Scene contains one or more `GameObjects`. GameObjects are the most fundamental entity within Unity and can be a range of things such as characters, props, or cameras. A GameObject can contain one or more children GameObjects and the child will inherit attributes from its parent. A `Component` describes a GameObject's features such as size, position, or color. The `Script` component is the most powerful component a GameObject can possess. The Script component allows developers to create C# programs to gain precise control of GameObjects during run-time.

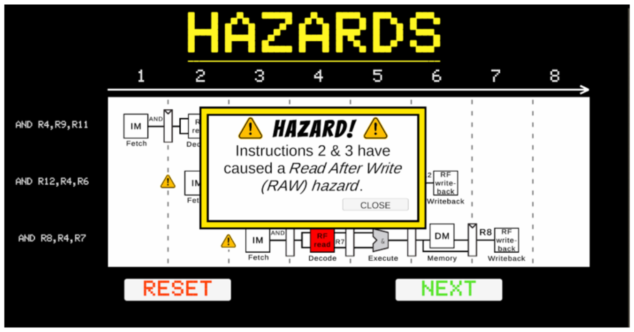
Unity provides integration with the Visual Studio IDE which is a great convenience during GameObject scripting. Visual Studio Community edition was used in this project for C# development. All GameObject scripts were created following traditional software development practices as best as possible. The game environment includes a 3D computer asset [14], purchased from the Unity Asset Store. The use of this asset, among other things, were "stylistic" choices made in this project. Although these choices do not technically improve the tool, the hope is that they would make the tool more appealing to students. To date, learning modules for Memory Access and Pipelining have been completed. Figure 1 shows the interaction of different modules in the visualization tool.

A. Memory Access Module

The memory access module focuses on the LDR and STR instructions, which are used for accessing computer memory. The LDR instruction is covered first with the basic instruction syntax being provided. Afterwards two visualizations are shown with a step-by-step explanation of the LDR instruction execution. The registers, data, and memory follow a layout similar to what is typically shown in the DDCA textbook. Students can visit the visualizations as

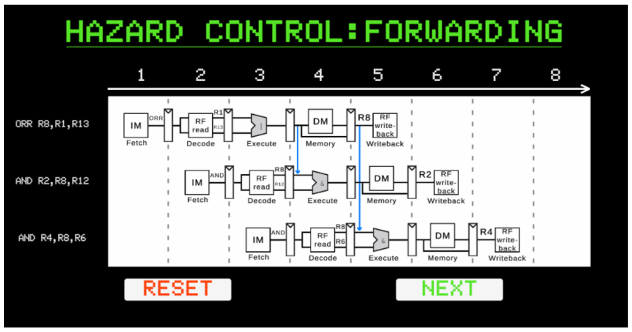


a. Visualization for pipelining

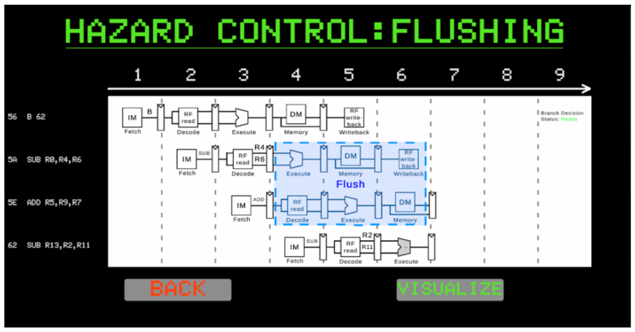


Instructions 2 & 3 have caused a *Read After Write (RAW) hazard*.

b. Hazards in pipelining

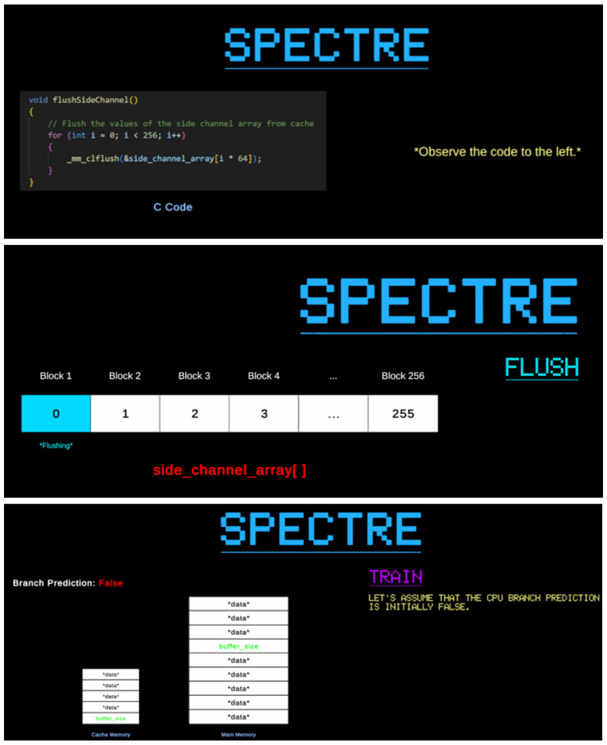


c. Visualization for forwarding



d. Visualization for flushing

Fig. 3: Sample visualization for pipeline



```
void flushSideChannel()
{
    // Flush the values of the side channel array from cache
    for (int i = 0; i < 256; i++)
    {
        _mm_clflush(&side_channel_array[i * 64]);
    }
}
```

LET'S ASSUME THAT THE CPU BRANCH PREDICTION IS INITIALLY FALSE.

Fig. 4: Sample visualization for Spectre

many times as they want. As mentioned before, the content associated with the visualization, such as the assembly instruction and data values, is randomly generated and will be unique upon each play through.

This same process of explanation and visualization is repeated for the STR instruction. After both instructions are explained and visualized, there is an activity where the user can "manually execute" the instruction to reinforce their learning. This activity contains three examples of either an LDR or STR instruction (again due to content randomization). The student must get the correct answer to progress, but they have unlimited attempts to do so. After completing the activity, the user is challenged with a six-question multiple-choice quiz. The questions incorporate randomization to avoid the user simply remembering the correct answers after long enough playtime. Once they have submitted their answers, they are shown their quiz score and have completed the module. The user can either review their submission or return to the main menu. Figure 2 shows sample visualization for LDR instruction.

B. Pipelining Module

The pipelining module has been developed closely referencing Chapter 7 of the DDCA textbook [15]. The pipelining technique and the five pipeline stages are explained in the visualization tool. Afterwards, a visualization of a pipelined CPU executing three randomized instructions is shown. The visualization shows the execution of each instruction on a per clock cycle basis, and states which pipeline stage they are currently in. The users can review the visualization as many times as they want. The module shares the downside of a pipelined CPU's increased performance, which is hazards. The data hazard, specifically the Read After Write (RAW) hazard, is covered first. A visualization of a pipelined CPU executing three instructions is once again shown, however these instructions now present a RAW hazard. The decode stage and the hazardous register are highlighted in red to illustrate where the hazard is occurring. After this the methods used to manage data hazards are explained. These methods include the use of NOP instructions, forwarding, and stalling. The limitations of the NOP and forwarding approach are discussed. Each method is given a visualization to portray what is done to prevent the hazard. The pipeline forwarding visualization includes an arrow animation showing the data being forwarded from the memory and writeback stages to the execute stage of the subsequent instructions.

The pipeline stalling visualization distinguishes which clock cycle the stall is happening and shows a loading animation. The control hazard and the pipeline flushing method which prevents it are the final topics of the module. The pipeline flushing visualization shows a pipelined CPU execute four instructions where the first instruction branches to the last. The visualization shows the status of the branch decision, highlights what is flushed, and includes an animation of the instructions being "discarded" into a trash can. After this visualization the user can take the end of module quiz. A few sample visualization screenshots are shown in Figure 3.

C. Spectre

The Spectre module is currently in development so it will be briefly discussed because details may be subject to change. It is important to acknowledge [11] for serving as the "ground truth" and SEED Labs [16] for providing practical experience on how the attack functions. The module will discuss the discovery of the attack, related terms (such as speculative execution and side-channel attacks), then discuss attack methodology. According to the knowledge acquired from both of the aforementioned resources, the attack can be divided into four steps: *flush*, *train*, *invoke*, and *reload*. In the first step of the attack, the side-channel is flushed from the cache. The second step of the attack is to train the CPU to perform speculative execution. In the third step the attacker invokes a "target function" to steal private data. In the final step of the attack the side-channel is reloaded. The visualization shown for this module in Figure 4 follows a proof of concept C program demonstrating this procedure. It is mentioned that the C program is pseudo-code. Students are encouraged to do additional online research for a full Spectre program. As with the others, this module will be concluded with a short multiple-choice quiz.

V. Conclusion

As mentioned in the literature, visualization is an effective learning technique. This project sought to satisfy the need for a fresh approach to teach computing organization and design. Two interactive and visualization-based learning modules have been developed for the undergraduate Computer Organization and Design course. The tool is to be distributed this Spring 2024 semester and results will be collected to assess the effectiveness of the developed modules. The focus is on developing more modules in the immediate future. The two modules that have already been developed cover traditional microarchitecture, the next module to be developed will shift to micro-architecture security. After this additional modules will continue to be developed and the tool will be released publicly for widespread adoption by instructors in other universities.

Acknowledgement

This work is based upon the work supported by the National Science Foundation (NSF) under Grant No. 2021345. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of NSF.

References

[1] M. H. U. Sharif and M. A. Mohammed, "A literature review of financial losses statistics for cyber security and future trend," *World Journal of Advanced Research and Reviews*, vol. 15, no. 1, pp. 138–156, 2022.

[2] J. Á. Velázquez-Iturbide, I. Hernán-Losada, and M. Paredes-Velasco, "Evaluating the effect of program visualization on student motivation," *IEEE Transactions on Education*, vol. 60, no. 3, pp. 238–245, 2017.

[3] M. K. Quweider and F. Khan, "Visualization as effective instructional and learning tools in the computer science curriculum," in *2017 ASEE Annual Conference & Exposition*, 2017.

[4] P. N. A. Barata, M. Ribeiro Filho, and M. V. A. Nunes, "Consolidating learning in power systems: Virtual reality applied to the study of the operation of electric power transformers," *IEEE Transactions on Education*, vol. 58, no. 4, pp. 255–261, 2015.

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

[6] 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.

[7] C. D. Hazlett, *A MIPS datapath simulator for enhancing visual learning of computer architecture*. PhD thesis, 2020.

[8] M. I. F. Alves, "eduarm: Web platform to support the teaching and learning of the arm architecture," 2022.

[9] R. Giorgi and G. Mariotti, "Webrisc-v: A web-based education-oriented risc-v pipeline simulation environment," in *Proceedings of the workshop on computer architecture education*, pp. 1–6, 2019.

[10] S. Harris and D. Harris, *Digital Design and Computer Architecture: ARM Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2015.

[11] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[12] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[13] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[14] SnowQ (Unity Asset Store Publisher), "3d computer asset." https://assetstore.unity.com/packages/3d/props/electronics/computer-211592, 2022. Purchased: 10/20/2022.

[15] S. Harris and D. Harris, *Digital Design and Computer Architecture: ARM Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2015.

[16] Wenliang Du, "Seed labs – spectre attack lab." https://seedsecuritylabs.org/Labs_16.04/PDF/Spectre_Attack.pdf, 2018. Accessed: March 2024.