Qeios

Peer Review

Review of: "WebRISC-V: A 64-bit RISC-V Pipeline Simulator for Computer Architecture Classes"

Pablo Fuentes[1]

1. Universidad de Cantabria, Spain

The paper describes a graphic web-based tool to simulate the execution and pipeline of a RISC-V processor for educational purposes. The tool is able to show the outcome of each stage in the pipeline and to step backwards, so that a user can better understand the causes of unexpected behavior.

The paper is well-written and well-structured, easy to read, although sparse on details. I think the paper should be significantly extended to better explain the inner behavior and characteristics of the tool. For example, section “Software Architecture” indicates that “the simulator supports the full implementation of two RISC-V modules.” It does not explain what a RISC-V module is, instead redirecting the reader to a short footnote and to the description in the ISA specification; I believe a short explanation would make the paper easier to read. Furthermore, the paper does not indicate the ability to expand those modules, which can be critical since the base instruction set is significantly limited, relying on the use of those modules to provide common capabilities. As a reader (and potential user of the tool), I would like to know if such expansion ability exists, and if so, how complex it would be to develop it.

Another example of lack of details is the focus on the webpage service, with a single sentence describing that the tool can be locally installed as “an instance [...] hosted with a simple procedure on a Linux or Windows server.” However, the paper does not expand on that process, and I have not been able to find any description of the procedure on the page of the tool itself. I believe a web service is significantly easier and more comfortable for students to run, but it can be a significant limitation for evaluation if internet access suffers any type of degradation or if the teacher wants to limit online connectivity.

Regarding the comparison against previous tools, the section gives a good argument to choose the WebRISC-V tool but is heavily leaning on MIPS tools. Given the ongoing low market share of MIPS during the last two decades, I expect a significant number of courses have by now migrated to other ISAs such as

 doi.org/10.32388/2WA5WU

ARM. A comparison against other tools focused on a given ARM pipeline would be useful and make the comparison more adequate. I also find the comparison against other RISC-V tools a bit limited, only citing Ripes. I understand that the significant benefit of WebRISC-V over Ripes is the ability to run on a web service, although this is not explicitly stated. However, other (by now mature) tools supporting the RISC-V ISA have been introduced. WepSIM (https://wepsim.github.io/wepsim/) provides a similar tool to simulate the pipeline of a RISC-V processor and, although the interface makes the process of assessing the stages each instruction goes through, has good, easy-to-run examples and allows for better modulation of the amount of detail shown by the tool. Another good web-based simulator is CPUlator (https://cpulator.01xz.net/); it does not show the pipeline behavior but has significant support for several ISAs and has a well-polished interface with the capability to set breakpoints and watchpoints, and to load ELF binary files (instead of simply relying on a text editor to develop the codes, as WebRISC-V does).

The paper does not stress enough the ability of the tool to step backwards in the code execution, which I consider a prominent feature. I would encourage the authors to consider implementing a post-mortem analysis to trace back to a given instruction, instead of going back step-by-step.

One final point about the paper that I find uneasy is the list of authors. This paper is single-handedly written by one person, but a previous paper introducing the tool was co-written by two authors, with the missing author in this paper being the only contributor and owner of the public repository where the tool is hosted.

Regarding the tool itself, I find it compelling but in need of an overhaul, particularly regarding the interface. It does not provide any error messages when a program has syntax errors, which can be a major deterrent for educational purposes. A set of given easy-to-load examples would also make it more useful for would-be educational users. I think the left panel, where the “Execution Status” is shown, provides kind of a sensory overload while missing a good overview: instructions are shown in assembly code and machine language, the latter broken down by fields, even indicating the type of instruction executed. This prevents showing more than 5 instructions at a time, limiting the understanding of the code functionality. I would suggest simply showing the address and assembly code of the instructions, with a right marker highlighting in color the stage of the pipeline the instruction is at, and showing the rest of the information separately (through a clickable or hover-over window, or in a lower separate view, focusing on a single instruction at a time). This information could also be merged with the execution table, simplifying the information provided to users.

Registers could be shown in hex format instead of binary, which would reduce the number of digits needed and also be easier to grasp. Memory could have a default view, with users able to configure other setups through a menu instead of having to select a specific option to get an initial view. Font usage, particularly font sizes, should be given more thought, since it can be a significant limitation for visually impaired students. Additionally, the upper bar employs a significant amount of space for options that are typically set and forgotten, such as the architecture or the ability to show paths; those would be better placed in an initial menu to access the main interface (with a similar two-step process as the one used by CPUlator) and then available as a menu option. This would free space to relocate the execution controls and allow showing the whole execution status (instructions, data, and registers) simultaneously.

Declarations

Potential competing interests: No potential competing interests to declare.