EmuGator Alpha Build

Fiona Cahalan Nikhil Iyer Liam Rosenfeld

Rohan Simon Christopher Tressler

I. Introduction

EmuGator is a web application developed to serve as an online development environment and educational tool for RISC-V assembly language. In this Alpha Build, the develop team describes EmuGator's progress by sharing usability, build quality, vertical features, the effort put in for this past milestone, and known bugs.

II. Usability

A. Interface

Interface elements are all accessible. This includes the Monaco text editor, "Assemble" and "Next Clock" buttons, text pipeline description, register file, instruction memory, and data memory.

The Monaco text editor has been properly implemented with RISC-V syntax highlighting, breakpoint toggling when clicking in the gutter (standard for most IDEs), highlighting the current instructions running, and the default visuals for the dark-mode of Monaco text editor (also utilized by Microsoft Visual Studio [1]).

The "Assemble" and "Next Clock" buttons use colors that contrast with the background and bold text.

The text pipeline description is a placeholder for the eventual visualized pipeline but is currently a readable information struct.

Instruction Memory and Data Memory have a shared view but are easily switched by a toggle button [2].

B. Navigation

EmuGator's navigation is simple as the web application fits entirely on one screen in most desktop cases. When a screen is too small for the text editor, pipeline, register, or memory views, each section has an intuitive scroll bar for navigation.

C. Perception

Interactions are made to be intuitive to users and give confirming feedback. Of the "Assemble" button, "Next Clock" button, and memory view toggle, only the "Assemble" button is available when there is no program assembled, ensuring users do not try to run a program or view data that does not exist.

Furthermore, the buttons give feedback. The "Assemble" button loads the data and instruction memory and starts a program counter outlining within the Monaco text editor. When "Next Clock" is pressed, register values update (if necessary), and the program counter increases (including the text editor outline). When the memory view is toggled between instruction and data memory, the currently selected memory is underlined.

D. Responsiveness

EmuGator is fully responsive. Multiple tasks can be processed simultaneously and buttons immediately perform actions with no form of unnecessary polling. The latency between pressing the next clock cycle button and the emulation finishing and the visualization updating is imperceptible. When running until hitting a breakpoint is added, it will run within a Web Worker allowing the user interface to remain responsive when the emulator is running in the background.

III. Build Quality

A. Robustness

Crashes do not occur during the normal Program Entry, Assemble, Step Through workflow. There is some robustness to invalid user input, errors due to invalid RISC-V code do not cause crashes or unexpected behavior.

B. Consistency

There is no unpredictable or undefined behavior present in the system. For a given user input the internal systems, persistent state, and external interface are all fully deterministic.

The assembler is implemented as a pure function on the program inputted by the user. The emulator is implemented as a pure function over the last emulator state and the current memory maps.

By implementing these as pure functions (with no internal state) it is easy to perform exhaustive automated testing and provide stability guarantees for these components.

Dioxus is structured such that the view is a function of the state, so by ensuring that the state transitions are deterministic, we can be relatively confident the external interface is deterministic as well.

C. Aesthetic Rigor

The design of EmuGator is intending to strike a balance between two properties in opposition: feeling spacious and having all information available at a glance. To enable this balance, we are very intentional about what is presented on the screen and where. In addition, text was laid out to give breathing room between other text allowing even dense information to not feel cramped.

As we continue to add features, we will gradually make presentation of data more dense. However, we will do so gradually so that we can ensure it does not cross the point of feeling dense.

Currently, there are no known interface bugs, with the consistency of Dioxus allowing us to be relatively exhaustive during manual testing.

IV. Vertical Features

A. Persistent State

There are two main chunks of persistent state, the memory maps, and the emulator state. The emulator state contains the internal signals present in the emulator such as the values present in each of the registers. The memory maps hold both the instruction memory and data memory for the emulator between execution steps.

B. External Interface

The memory maps are reactively bound the the memory view panel. This panel displays the current memory map and highlights the location of the program counter in instruction memory.

The emulator state is currently rendered in a textual format, showing the values of each signal present within the emulator.

C. Internal Systems

The assembler processes the inputted program, which is passed to the emulator. The emulator processes the assembled code, preforming the expected operations for each instruction.

V. Project Links

The repository for this project can be found at github.com/EmuGatorUF/EmuGator. The repository "README" section includes step by step instructions to compile and run the project.

The latest public release of EmuGator can be found at emugator.com. No setup or compilation is necessary at this site.

VI. Milestone Effort

During this milestone we set up hosting, created an architecture plan to add the 5-stage pipeline, and added breakpoints to the front end.

For hosting we chose GitHub pages because it offered a simple, free solution for 24/7 availability. This process included three main steps: configuring repository settings for GitHub pages, writing a deployment script to build, test, and deploy through a CI/CD pipeline, and purchasing a domain through Namecheap and configuring a proper Domain Name System (DNS) [3]. We managed to acquire the domain emugator.com.

Through discussions, the team established a overview of how the 5-stage pipeline would be implemented and assigned each team member a stage to work on.

Users can now add breakpoints to their code as red dots next to a line number. However, back end support for this has not yet been implemented. Break points for the back end have been partially implemented on a separate branch but will not be added to the public release until a web worker is added to tackle crashes caused by infinite loops when running continuously.

VII. Known Bugs

- Users cannot continuously run programs, they must step through the program by clock cycles.
- Decreasing the page size vertically causes the editor to overflow vertically. Workaround: refresh the page after decreasing the vertical size of the window
- `.word` allocations over 255 do not currently work.

References

[1] Microsoft, "Monaco Editor." [Online]. Available: https://microsoft.github.io/monaco-editor/

[2] Microsoft, "Toggle controls - Windows apps." [Online]. Available: https://learn.microsoft.com/en-us/windows/apps/design/controls/toggles

[3] Cloudflare, "What is DNS? | DNS Definition | Cloudflare." [Online]. Available: https://www.cloudflare.com/learning/dns/what-is-dns/