The RISC Takers:
Final Report

Sean Keever
swkeever@uw.edu

Yokesh Jayakumar
karthj@uw.edu

John McMahon
mcmahjoh@uw.edu

Abstract

The goal of our project is to create a means of letting a user not only use an OS, but to allow a user to see what is going on inside the OS and CPU. To this end, our project aims to maximize availability to users by providing a browser-based interface that lets the user visit a webpage and visualize an OS from the moment he or she visits the page.

1 Introduction

There are multiple implementations of operating systems made available via a web browser. Instead of re-implementing that functionality, we chose to find a project that has the low-level emulation from C to JavaScript already handled. This way, we can focus on solving a new problem: providing an interface to visualize the OS and CPU.

We looked at multiple different OS implementations and decided on risv-angel, which seemed to offer the functionality we needed without too much underlying complexity. This way, we could focus more attention to creating the visualization layers without having to worry too much about the low-level emulation details.

2 Initial Design

To start, we stripped the riscv-angel project to only show the Terminal. We got rid of all the elements and libraries that were not going to be used in our project. Before we started on our implementation, we prototyped the UI in Figma, shown in Figure 1.

Figure 1: Initial prototype of design

Figma allowed us to come up with a design before we implement anything in code. We aimed for a dashboard-like interface, showing the user information about the CPU and OS in real-time.

3 Beginning Development

Setting up a development environment

One of the challenges for the team was facing ambiguity in this project. It was difficult to interpret exactly what needed to be done.

To facilitate this problem and make development easier, we started approaching the project with an Agile workflow. We are using GitHub's Kanban board functionality to track issues and milestones. We hold weekly stand-ups to find out

- what we did over the past week, and
- what we will do in the next week.

We use the GitHub issues that we created to assign tasks for each member of a group. In doing this, we were able to be more productive by having concrete milestones in place in order to complete the project.

To ensure robustness and quality of our project, we have also developed some tooling for continuous integration. To ensure consistency in the codebase, we use ESLint, a tool that we have configured to apply the rules defined in AirBnB's style guide. We use `lint-staged` to run our linters before any commit. This way, we know if stylistic errors need to be addressed before pushing code to the repository.

We also set up GitHub Actions, which are essentially hooks that execute when code is pushed to the repository. These hooks will run the style checker and our unit tests before code is pushed to remote. The hook will reject any commits if any of these actions fail. The GitHub Actions are certainly overkill for our case, but we wanted to have the infrastructure in place in order to configure our CI easily.

Choosing technologies

We want our interface to be interactive, so we chose React.js as our library of choice, which offers a means of rapidly developing interactive client-side applications.

This doesn't come without its challenges, though. The existing project uses vanilla JavaScript in order to run the emulator. The legacy code accomplishes this by spawning a worker thread that handles all the emulation tasks. The main thread interacts with this worker thread via message passing.

Working with the existing riscv-angel codebase was challenging. We resolved the issues by having rendering each dashboard component separately. This way, the dashboard components can coexist with the existing riscv-angel project code easily.

4 Establishing a Design System

None of the members on our team are trained designers. We had to be resourceful to learn some of the patterns of good UI design. To make this easier, we took a lot of inspiration from professionals by using the Material Design system used and maintained at Google. We imported a Material Design kit into Figma, which includes many Material Design assets such as buttons, cards, etc. Doing so allowed us to wireframe a UI prototype that we feel was closer to our vision for the app's design. Figure 2 shows the first steps of prototyping our UI using Material Design and Figma.

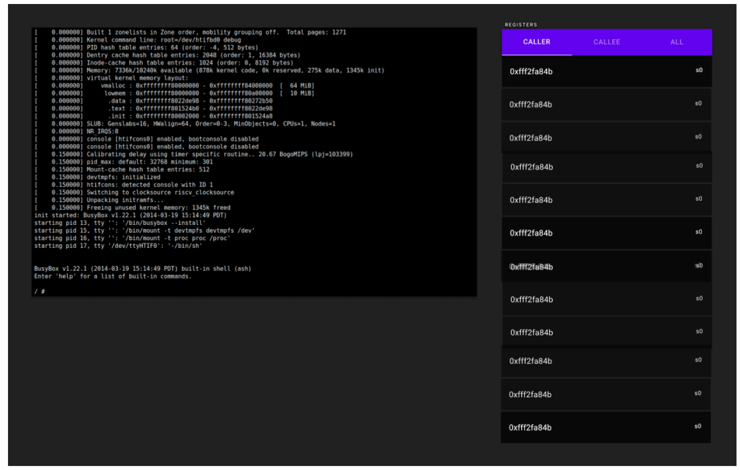

Figure 2: Initial prototype using Material Design

In addition to our team not being expert designers, we are also not experts in CSS, which is vital to making the application look good. We wanted our application to be as lightweight as possible, so we wanted to have our own custom CSS, as opposed to using a CSS framework like Bootstrap. To learn CSS and UI design patterns further, we read the book Refactoring UI, which provided many design tips from a developer's point of view.

We established a design system that is used throughout the app. We defined a system for colors, fonts, spacing/font size, and more. The system restricts properties of the UI to follow a discrete set of values. Although at first glance, it may seem that this restriction would limit our ability to create a design. In fact, the restriction provided by the design system helped to make the application feel polished and consistent.

5 Implementing a Working Prototype

A Modular Design

We describe our app as a modular, dashboard user interface that lets users use a basic OS and see properties about the OS and CPU in real time.

Our modular design means we have separate components for each visualization in the dashboard. For example, there is a separate component for the window showing the registers, a separate component for the view showing the ratio of instructions being executed, and so forth. We split our app into modular components for several reasons:

- easier to maintain and test
- allows features to rapidly developed
- offers more flexibility in UI structure

We will elaborate on each of these points.

First, because our dashboard interface uses modular UI components, whenever we work on a single component, we can be confident that breaking a single component won't break other parts of the app.

The modular design also allows us to split the work and develop features independently. This is great for us during these times of quarantine because of COVID-19. So, for example, while one team member is working on some feature *A*, another team member can be styling feature *B*. And since the components are modular, we can do so without worrying about conflicting with another person's work.

Finally, having the components act as modular entities allows us to change the layout of the UI very easily should we choose too. The components are simply independent building blocks that can be moved around as we please. This flexibility will become very important when it comes time to finalizing our app.

Finishing our Prototype

Information about the CPU state lives in the existing `riscv-angel` code, which is written in vanilla JavaScript. To pass information about the CPU to React, we bind a JavaScript object from the `riscv-angel` code to the globally accessible `window` object.

We then create a custom React hook that safely retrieves this state to be used by React. With the state of the CPU accessible in our React components, we could implement our dashboard.

In our app, we show the contents of the 32 user registers,

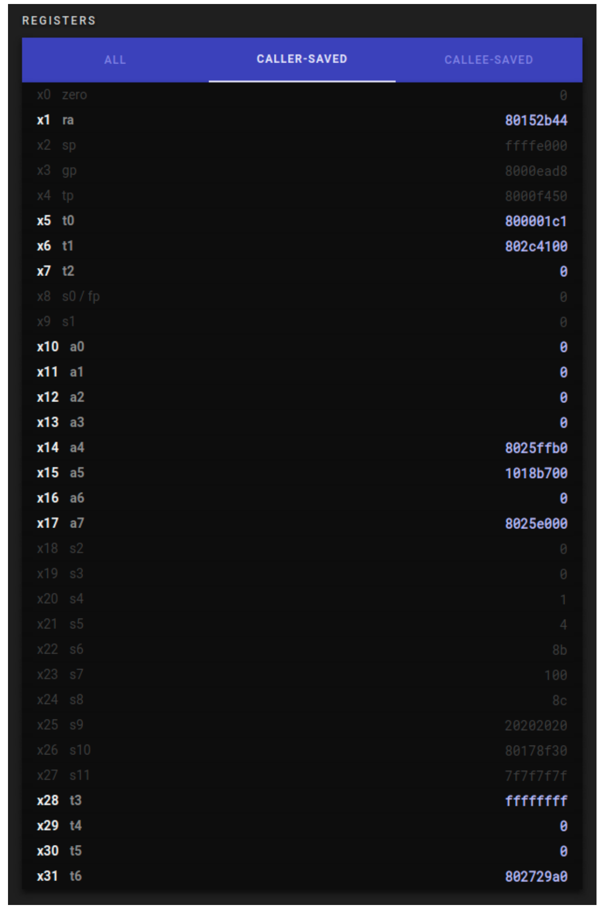

Figure 3: Register panel

the ratio of CPU instructions executed,

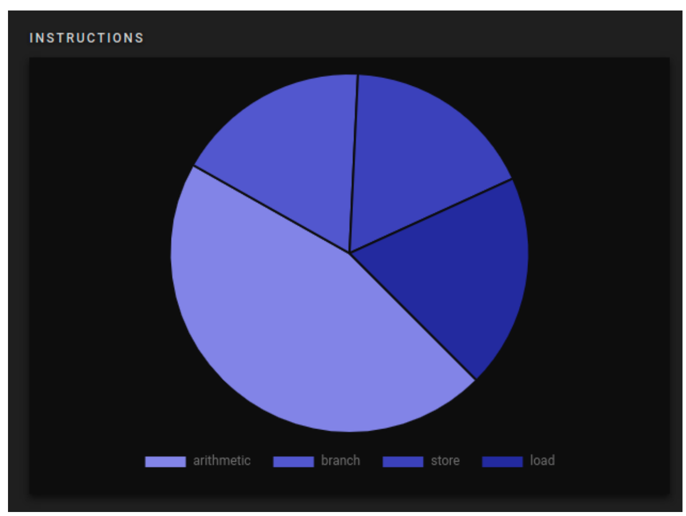

Figure 4: Instruction panel

and a time series graph showing memory utilization

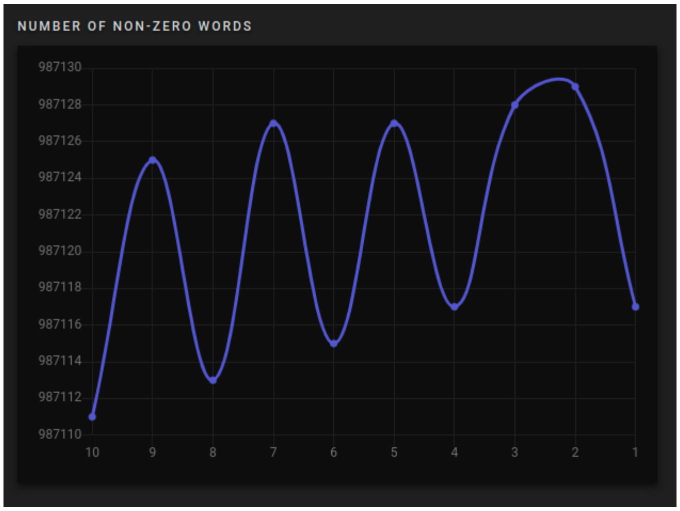

Figure 5: Memory panel

We feel that this offers a useful feature set to get a glance at what is going on under the hood of an operating system utilizing the RISC-V instruction set.

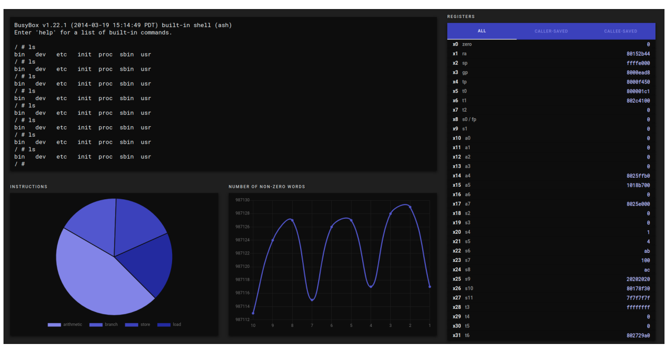

Figure 6: MVP

Deployment

After our prototype is complete, we prepare our application for deployment. To keep things as simple as possible, we host the app on the CSE department's servers at University of Washington. In this way, this project can easily be used as an example in future iterations of the CSE481a capstone course.

Next Steps

This project has a lot of room to grow. There are features still to be desired. Namely,

- A way to see inside the file system
- A way to get information about running processes

We feel our project provides a modular foundation for these features to be developed in the future. In hopes to reinvigorate life back into the riscv-angel project, we submitted a pull request to merge our extensions into the base riscv-angel branch, from which this project is based on.

Conclusion

In this project, we've taken an existing project, riscv-angel, and extended it to display an interactive dashboard that lets users see the internals of an OS and CPU running on a virtual machine. We believe our project could be used by educators to easily show students properties of an OS in real time. We designed and developed this application with modularity in mind. Our React component-based architecture allows features to rapidly developed and added to the UI. We believe this application has the potential to reinvigorate interest in the original riscv-angel project. At the very least, people can use our app to learn and experiment with an operating system running on a RISC-V architecture with very little barrier to entry.

Acknowledgments

Thanks to all of the contributors of riscv-angel, in which this project is based on.

Availability

This project is open-source and is available at https://github.com/swkeever/riscv-angel-extended