Visual RISC V Simulator

Project specification

Noah Hollowell

Department of Computer Science

University of Warwick

1 Project Statement

Within the understanding of how a modern processor works we often look into the underlying instructions and physical operations it performs. However, often being able to visualise those are far more beneficial to our understanding than simply emulating the physical code to produce a result.

RISC V is a new open source instruction set that has numerous emulators available online with a varying degree of visual elements. These all lack the ability to fully visualise the internal operations during each instruction execution. Thus, the aim is to produce a fully fledged RISC V simulator that not only emulates RISC V code but also visually demonstrates the flow of data during execution all the way down to data moving across the processor to their respective components updating in real-time.

2 Objectives

The first part of this project will be to produce a RISC V emulator built in Java. This will require a deep understanding of each RISC V instruction and how each interacts independently within a processor such that they can be combined to produce a logical flow of instructions producing an output.

Objectives include:

- A lexical analysis of inputted RISC V code,
- A semantic analysis of inputted RISC V code,
- Emulation of the base RV32I instruction set [11] as per the RISC V specification,
- Emulation of the "Standard Extension for Integer Multiplication and Division" instruction set,
- Emulation of the "Standard Extension for Single-Precision Floating-Point" instruction set

The second part of the project will be to visualise the emulator into a simulator providing a fully fledged interface to type and load code, as-well as visualise the internal operation of said code as its runs in real-time.

Objectives include:

- A visualisation of the processors register set,
- A visualisation of the memory holding both the instructions and data or just data,
- A comprehensive system allowing for the display of data moving around the processor including:
  - Numerical data moving from memory to other components and vice-versa,
  - Addressing requests,
  - Manipulation of multiple values simultaneously to simulate the effect of processor operations (e.g. addition, subtraction, shifts, etc),
- Control the speed and steps of the simulation,

To achieve the set out objectives we will approach the project with 2 main overarching goals:

1. A minimum viable product that incorporates the base RV32I instruction set and visualisations
2. A fully fledged product including both the Floating point and Multiplication extensions

And finally an aspirational objective to incorporate more RISC-V extensions dependant of time.

3 Methods

In order to successfully develop this project we'll be using a dynamic but structured approach, following along with some of the Agile [1] software methodology and parts of a plan driven approach. This will ensure that all deliverables are produced on time and ensure that a constant flow of progress is achieved with goals in mind and targets to hit.

We'll be supplementing development with a series of notes, thus ensuring decisions are tracked and sufficient documentation can be made later in the year.

3.1 Resources

The project will be developed entirely in Java 19 [7] using the OpenJDK [2] disimplementation. This choice was made due to my long running history of development with Java. Java is also highly maintained and constantly updated and has a world-wide community of people providing support at all levels. Java is also architect independent being compiled into byte code that is then run independently on each target device without having to recompile for each target. This is important or this project as the end program needs to be able to run on a range of devices that prospective users may have.

To produce the visual front end of the application well be using JavaFX [8]. JavaFX is a more modern GUI library lending itself towards efficient and rapid development which is exactly what the project needs. Java Swing is also a viable alternative, however being more familiar with JavaFX and have already obtained a visual pack called MaterialFX [9] to present a more modern feel, we shall be sticking with JavaFX.

To develop the application we'll be making use of Intellij IDEA Ultimate [6]. It's an all in one Java IDE, designed to maximise productivity and prove a wealth of support and debugging tools. Intellij is my main IDE of choice for

Java projects and from previous usage it's prefered over other IDE's such as VSCode for Java due to its highly inter-grated tooling and debugging infrastructure.

To ensure thorough testing of the application, we'll be making use of JUnit [10]. JUnit is a testing framework designed for test-driven development and provides a quick and elegant way to produce unit test for all aspects of our application. JUnit is also thoroughly documented and widely used across both the java community and professional industry.

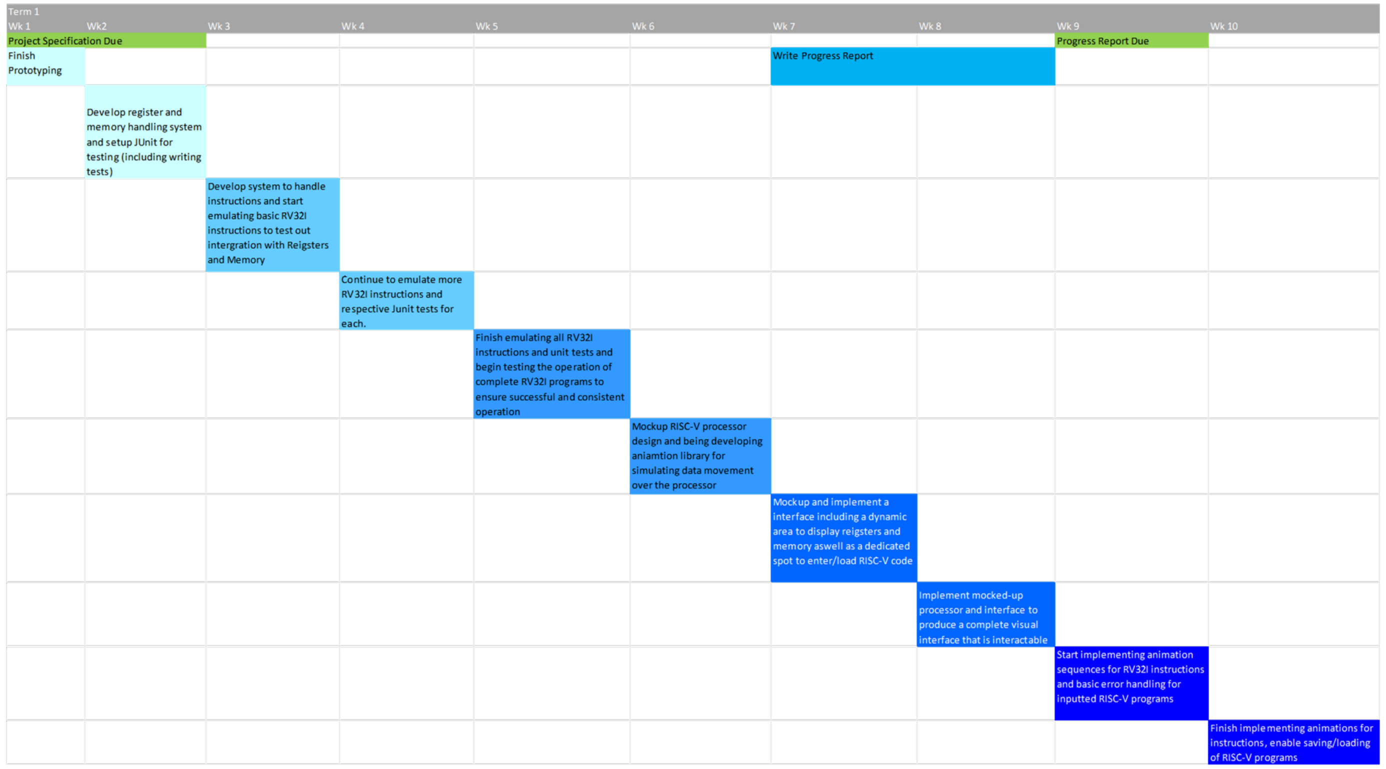
Version control will be done via Git [3] and Github [4] with a additional remote on the DCS systems as an extra backup. We have chosen Git over other Version-Control systems due the the fact that I am more familiar with Git and have extended usage of git in the past.

3.2 Testing

Test-driven development (TDD) will be a very important part of the project. As the nature of the project involves emulating individual instructions we can easily apply TDD using JUnit [10] over all the instructions we aim to implement.

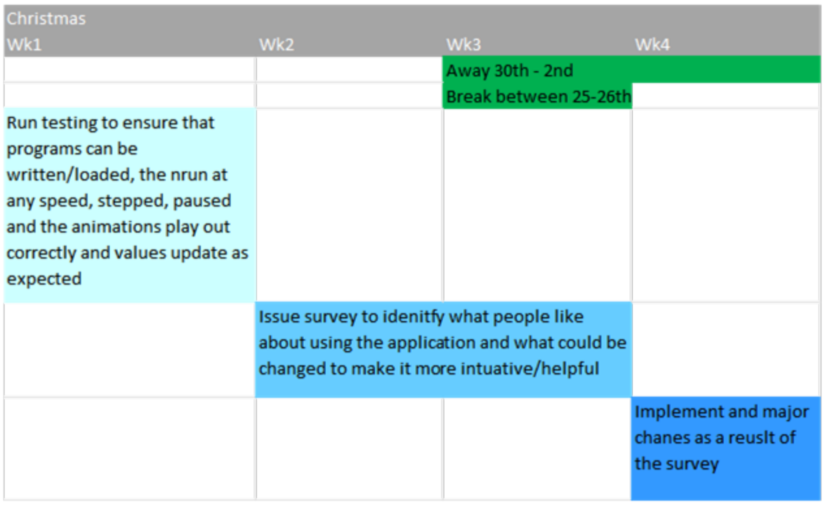
Manual testing will also occur for the more visual aspects, in which well ensure that we can visually observe animations taking place at the correct time and in the correct order, also ensuring animated element start and end at the intended place.

4 Timetable



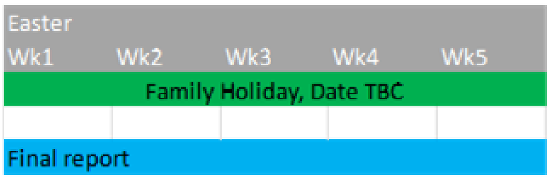
Term 1 Wk 1	Wk2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10
Project Specification Due								Progress Report Due	
Finish Prototyping						Write Progress Report			
	Develop register and memory handling system and setup JUnit for testing (including writing tests)								
		Develop system to handle instructions and start emulating basic RV32I instructions to test out intergration with Reigsters and Memory							
			Continue to emulate more RV32I instructions and respective Junit tests for each.						
				Finish emulating all RV32I instructions and unit tests and begin testing the operation of complete RV32I programs to ensure successful and consistent operation					
					Mockup RISC-V processor design and being developing aniamtion library for simulating data movement over the processor				
						Mockup and implement a interface including a dynamic area to display reigsters and memory aswell as a dedicated spot to enter/load RISC-V code			
							Implement mocked-up processor and interface to produce a complete visual interface that is interactable		
								Start implementing animation sequences for RV32I instructions and basic error handling for inputted RISC-V programs	
									Finish implementing animations for instructions, enable saving/loading of RISC-V programs

Figure 1: Term 1 Timetable



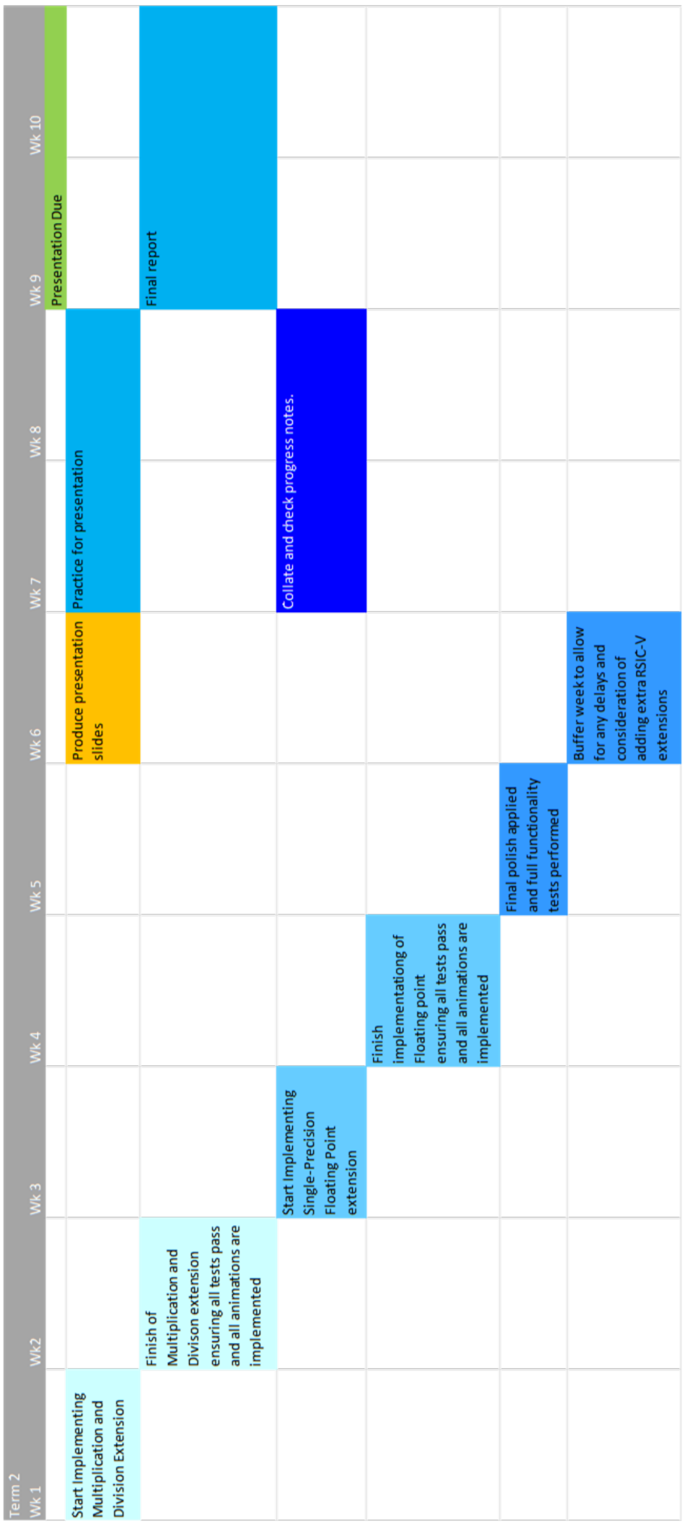
Christmas			
Wk1	Wk2	Wk3	Wk4
		Away 30th - 2nd	
		Break between 25-26th	
Run testing to ensure that programs can be written/loaded, the nrun at any speed, stepped, paused and the animations play out correctly and values update as expected			
	Issue survey to idenitfy what people like about using the application and what could be changed to make it more intuative/helpful		
			Implement and major chanes as a reuslt of the survey

Figure 2: Christmas Timetable



Easter				
Wk1	Wk2	Wk3	Wk4	Wk5
Family Holiday, Date TBC				
Final report				

Figure 3: Easter Timetable



Term 2 Wk 1	Wk2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10
								Presentation Due	
Start Implementing Multiplication and Division Extension					Produce presentation slides	Practice for presentation			
	Finish of Multiplication and Divison extension ensuring all tests pass and all animations are implemented							Final report	
		Start Implementing Single-Precision Floating Point extension				Collate and check progress notes.			
			Finish implementationg of Floating point ensuring all tests pass and all animations are implemented						
				Final polish applied and full functionality tests performed					
					Buffer week to allow for any delays and consideration of adding extra RSIC-V extensions				

Figure 4: Term 2 Timetable

5 Risk Assessment

There is a relatively small amount of risk associated with the project. By having a minimal viable project goal and then a more fully fledged goal we alleviate any risk of ending up with nothing to show and through the spread out nature of implementing sections we can ensure that the entire project will be completed on time with all goals achieved.

A possible risk is loosing all code/documentation due to unforeseeable circumstances. However, by making use of Github as a remote and also a remote on the Department of Computer Science's server (DCS), along with zipped backups online we can ensure that there will always be a updated copy of the project available should one become lost, corrupted or unavailable.

6 Legal, Social, Ethical and Professional Issues

RISC-V is free and open with a permissive licence. Encourages all implementations are compliant to specification [11]

Java and its JDK's are available for free use to all users post JDK 16 (we'll be using OpenJDK 19 as OpenJDK has better rights usage compared to OracleJDK). JavaFX follows the same licence as Java and MaterialFX makes use of the GNU Lesser General Public Licence v3.0 [5] permitting Private and Commercial use.

Within surveying for how users find the program we need to consider who we are surveying and why, ensuring no biases are taken to ensure a comprehensive set of feedback from a range of technical backgrounds, and ensure that the collected information is stored appropriately and disposed of when no longer required.

Professionally we'll be sticking to widely known design principles and a professional manner of commenting, ensuring that all comments are meaningful.

References

[1] Atlassian. *What is Agile? | Atlassian*. Atlassian, 2022. URL: `https://www.atlassian.com/agile` (visited on 07/10/2022).

[2] Oracle Corporation. *OpenJDK*. Openjdk.org, 2022. URL: `https://openjdk.org/` (visited on 07/10/2022).

[3] Git. *Git*. Git-scm.com, 2022. URL: `https://git-scm.com/` (visited on 07/10/2022).

[4] Github. *Build software better, together*. GitHub, 2013. URL: `https://github.com`.

[5] GNU. *GNU Lesser General Public License v3.0 - GNU Project - Free Software Foundation*. Gnu.org, 2022. URL: `https://www.gnu.org/licenses/lgpl-3.0.en.html` (visited on 07/10/2022).

[6] JetBrains. *IntelliJ IDEA*. JetBrains, 2022. URL: `https://www.jetbrains.com/idea/` (visited on 07/10/2022).

[7] Sun Microsystems. *Java*. Java.com, 2022. URL: `https://www.java.com/en/` (visited on 07/10/2022).

[8] Sun Microsystems. *JavaFX*. Openjfx.io, 2022. URL: `https://openjfx.io/` (visited on 07/10/2022).

[9] palexdev. *palexdev/MaterialFX: A library of material components for JavaFX*. GitHub, May 2022. URL: `https://github.com/palexdev/MaterialFX` (visited on 07/10/2022).

[10] JUnit Team. *JUnit 5*. Junit.org, 2019. URL: `https://junit.org/junit5/` (visited on 07/10/2022).

[11] Andrew Waterman and Krste AsanoviÄ. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213*. 2019. URL: `https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf` (visited on 07/10/2022).