Spencer Kaiser - 28750375

# CSE 7381: Computer Architecture Final Project

## Project Goals

The goal of this project was to to create an instruction pipeline simulator that could be used to help demonstrate how data flows between the 5 stages of a simple instruction pipeline. For some students, it can be difficult to visualize this data flow. This visualization becomes increasingly difficult to understand after the concept of hazards is introduced, which requires the use of stalls to prevent data hazards from occurring. While there are some solutions available online, many are overly complex and lack visual components that may further aid students in understanding the core concept.

The end-result of this project will be a simulator with a simple user interface that could be used to help visualize the 5-stage instruction pipeline in an easily understandable way for students of Computer Science and Computer Engineering courses, such as Digital Computer Design and Computer Architecture. The simulator would take advantage of user interface principles, such as the effective use of animation, and a dynamic instruction pipeline to help guide understanding in an interactive environment.

## Approach

Initially, Python appeared to be a practical language for building this simulator. Python as a whole is a great language for use in hardware simulation as it is scalable, easy to understand from a readability perspective, and it is very versatile. While there are many options for Graphical User Interface (GUI) libraries for Python, the Tkinter library is the most widely used and is generally accepted to be the standard for Python. Tkinter allows the programmer to create scalable user interfaces with widgets like buttons, sliders, frames (sub-windows within the greater window), and more.

Getting started with Tkinter was pretty straight forward and it was relatively easy to create a simple user interface with a variety of buttons and text labels. When it became time to incorporate more intricate aspects of the simulator, however, it became apparent that Tkinter did not provide appropriate methods of modifying the UI dynamically without the use of additional third-party libraries. This lack of functionality was a rather large drawback to using Python, and as a result, the choice was made to switch to an alternative language. After scrapping the Python project, there were a few alternative environments in which the simulator could be efficiently built; web-based, Java, and Swift, Apple's new iOS programming language.

Creating a web-based project was alluring as the combination of HTML, CSS, and jQuery allows for every possible type of necessary animation and the ability to dynamically change UI components, however, there is a significant amount of overhead involved with importing libraries, linking files of different types, and potentially dealing with file permissions issues. Despite the established confidence that this type of project was feasible, the overhead involved left quite a bit to be desired.

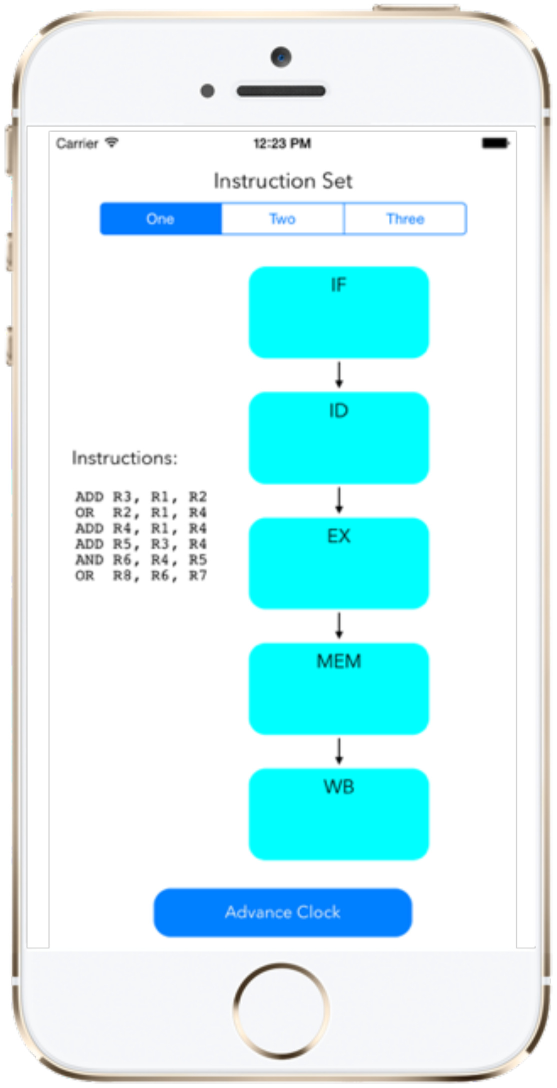
The Java standard library includes a wide variety of UI components, however, not everything needed for this project may be included. In addition, the learning curve for Java, and creating user interfaces with Java, can be quite steep. Furthermore, creating a UI with Java requires a trial and error approach with respect to position, size, and alignment.

The Swift programming language, the newest language created by Apple to create iOS applications, incorporates all the beneficial qualities of a web-based project without the potential overhead. In addition, the Xcode IDE also includes an incredibly powerful UI editor known as 'Storyboard', which can be used to quickly create the a foundation for the user interface as well as incorporate it within the codebase for programatical reference and modification. At first glance, it wasn't apparent that this type of project could be well executed as a mobile application, however, given the versatility and power of Swift, it became clear that it was the most promising language to use for the production of the simulator.

Creating the app in Swift required creating an interactive user interface, which was done within the ‘storyboard’ mentioned above, creating three varied instruction sets which each cause stalls, and creating functionality to visualize the data flow within the pipeline. The user interface was relatively simple to create; it consists of a text field to contain the list of instructions, a segmented control for the choice of instruction set, views to contain each of the 5 stages, which each contain text labels and have an adjustable background color, and finally, a button to allow the user to advance the clock.

## Results

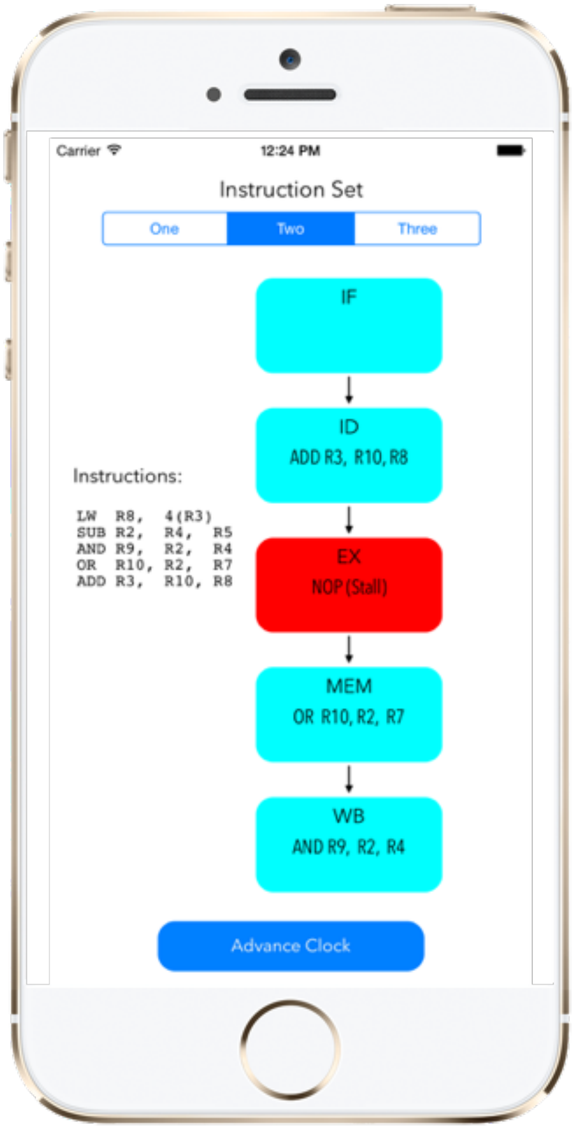
The app consists of one primary view with a number of items that the user can interact with. At the very top of the screen, the user can choose from one of three different instruction sets. Each contains a different selection of operations, which result in at least one stall. When the user chooses a different instruction set, the instructions list on the left hand side of the screen changes accordingly. On the right side of the screen, the 5 stages of the instruction pipeline are shown, which are eventually populated with instructions as the user advances the clock with the button at the bottom of the view.



Instructions:

```
ADD R3, R1, R2
OR  R2, R1, R4
ADD R4, R1, R4
ADD R5, R3, R4
AND R6, R4, R5
OR  R8, R6, R7
```

When the user clicks on the 'Advance Clock' button, instructions move through the instruction pipeline. Each time this happens, the instructions fade in and fade out of their respective stages in order to draw the eye to the movement and better indicate how instructions move through the pipeline.



Instructions:

```
LW  R8,  4(R3)
SUB R2,  R4,  R5
AND R9,  R2,  R4
OR  R10, R2,  R7
ADD R3,  R10, R8
```

When a stall is needed, a NOP is inserted wherever necessary in order to successfully avoid a data hazard. When this takes place, the stage where the NOP is inserted becomes red to indicate that a stall has occurred, and the instruction within that stage is replaced with "NOP (Stall)".

Eventually, all the instructions will finish execution and the pipeline will contain no instructions. When this happens, the simulator resets and prepares to begin executing the selected instruction set. At any time, the user can select a new instruction set. If this occurs while a current instruction set is executing, the pipeline will be flushed before beginning execution of the next set.

In the future, the simulator could be expanded to allow for the inclusion of additional features that fell outside the scope of this project. These features include adding the ability to allow the user to enter a custom instruction set, including settings that would allow the user to enable/disable data forwarding, and finally, including a variety of other types of instruction pipelines. During this project, it became apparent that incorporating these additional components would drastically increase the complexity of the simulator, so while they may significantly contribute to the user experience and the user's ability to understand the instruction pipeline, they were not feasible for this project.

## Conclusions

The end-result of this project provides students of Computer Science and Computer Engineering courses with an easy to use, interactive method of understanding the instruction pipeline, specifically in the context of using stalls to prevent data hazards. The simulator is very simple from a UI perspective, animations help guide the user's eye to changes in the pipeline, and providing 3 instruction sets provides even more context to the user and will hopefully further improve the experience. Furthermore, by allowing the user to individually affect the pipeline on a per-clock-cycle basis, the user is able to take as much time as necessary to fully understand the changes before advancing the clock again. Overall, the simulator is a great starting point to aid the understanding of a basic 5-stage instruction pipeline in a simple, yet engaging environment.