ECE649 Final Project Report

5/11/2024

Dr. Xiaolong Guo

Tannyr Singleterry, Zane Zollers

Table of Contents

Team Member Contributions……………………………………………………………………...3

Introduction………………………………………………………………………………………..4

CPU Hardware…………………………………………………………………………………….5

Describing Hardware Layout………………………………………………………….......5

Logisim Schematics……………………………………………………………………….5

CPU Software……………………………………………………………………………………10

Provided Testing Code…………………………………………………………………...10

Advanced Tasks Test Code………………………………………………………………10

PONG Code……………………………………………………………………………...10

LCD Code………………………………………………………………………………..10

Conclusion……………………………………………………………………………………….11

Team Member Contributions

Both: We both decided on what different extra credit challenges we wanted to do based on how much knowledge we had of design beforehand. We both worked on the ALU module.

Tannyr: Program Memory module, Decode module, some of the control lines, and report.

Zane: Registers module, Data Memory module, most of the control lines, implemented the keyboard input module for PONG, and test code.

Introduction

For this project, we were tasked to build a CPU that supports a subset of RISC-V instructions. For this project we used Logisim-Evolution to handle the design work, and Ripes to write test code to port over to Logisim. The base requirement tasks for this project are as follows: 32-bit instruction set architecture (ISA), at least 8 registers, an Arithmetic Logic Unit (ALU), instruction support for RISC-V instructions (add, sub, addi, lw, sw, beq), 8 \* 8 LED output, and testing.

We were also given the opportunity to do advanced tasks for our CPU. These advanced tasks in most cases for us were just a basic expansion of the basic tasks and we felt they could be implemented without much difficulty. The advanced tasks we decided to implement are: support for 32 registers for addi/add/sub/lw/sw, support for R-Type instructions (and, or, xor), 3 stage pipeline support, extending the LED matrix to 16 \* 16, a pixel game (PONG), and data hazard enhancement support. We felt like between the basic tasks and these advanced tasks, there will be enough of a challenge to be fun for our group but not too complicated that it felt unrealistic for us to complete the project.

CPU Hardware

For this section, we will be presenting detailed diagrams of our Logisim hardware designs and discussing key features and points of interest.

*Describing Hardware Layout*

The program will take in an instruction set and load it into the Program\_Memory module. Which will store all the instructions and load them accordingly. It will take these instructions and put them into the Decode module, which will fetch the instructions and decode them into their bits for the registers. The Registers module will take in addresses from the Decode module and will store and manipulate the instructions provided based on what we need. The ALU module performs any arithmetic provided via the instructions. The Data\_Memory module acts as RAM for our CPU. Within the main module, there is a circuit to take in keyboard inputs, we are using this circuit to make the game of PONG.

A diagram of a circuit

Description automatically generated*Logisim Schematics*

Figure 1: Main Module Diagram

A diagram of a circuit board

Description automatically generatedA diagram of a building

Description automatically generated

Figure 3: Decode Module Diagram

Figure 2: Program Memory Module Diagram

A diagram of a computer

Description automatically generatedA computer screen shot of a diagram

Description automatically generated

Figure 4: Registers Module Diagram. This is only the required 8 of the 32 implemented.

Figure 5: ALU Module Diagram.

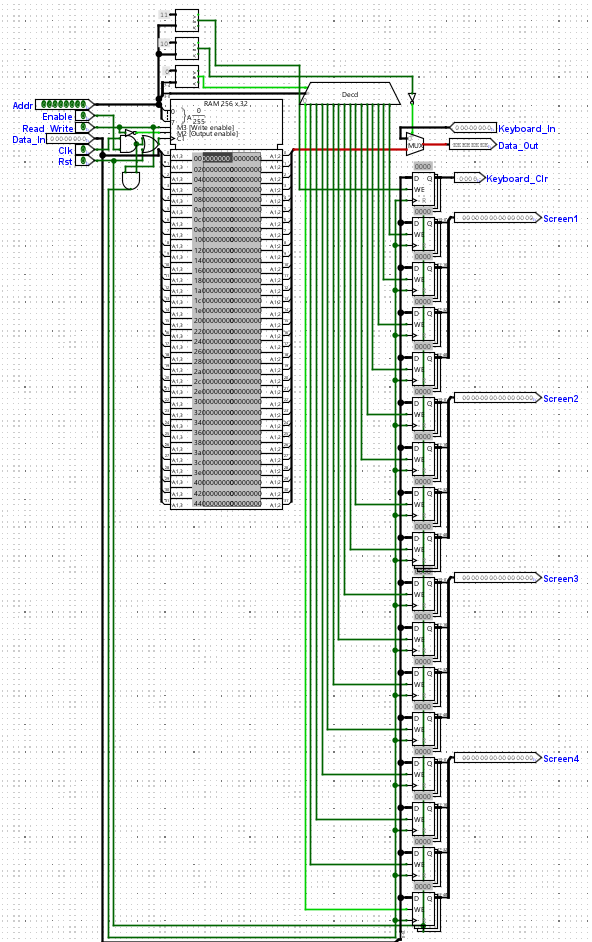


Figure 6: Data Memory Module Diagram

A computer screen shot of a computer

Description automatically generated

Figure 7: Keyboard Input Circuit in Main Module

CPU Software

For this section, we will outline the functions and purposes of our ASM programs. We will have four testing codes: the provided testing code, our own testing code for the advanced codes, code for PONG, and code to display the LED.

*Provided Testing Code*

This was testing code provided by Dr. Guo for this project. This is to test basic tasks and instructions for our CPU. This is code to test instructions like add, sub, and lw. Everything worked without any problems.

*Advanced Tasks Test Code*

This was testing code written by Zane to test all the advanced task instructions we decided to implement.

*PONG Code*

This is code for us to run PONG on our LED screen. The code makes the pixel ball move across the LED screen and bounce on the paddles. The code will also make the pixel paddles, one on the left and one on the right side of the screen. Using the keyboard module in the main circuit, we have it coded so W and S moves the left paddle up and down respectively, and O and L to move the right paddle up and down.

*LED Code*

This was code to show that the LED works and can show an image. We decided to go with a face, not the exact one shown in the example final project, but our own since we have a bigger LED than the base circuit. An image of the LED face can be found below.

Conclusion

To summarize, this project went smoothly overall. There was enough of a challenge that we felt like we both learned, but it was not so challenging that it became overwhelming. Zane handled most of the testing code since that was his biggest area of expertise. While Tannyr wrote the report since he has more experience writing reports for projects and classes. We both split up the actual design work in Logisim. We felt overall this project was fun and we achieved everything we wanted to and then some.

If we had to start anew, we would not do anything differently. We decided what we wanted to do as a group, assessed both of our strengths and weaknesses, and got to work. Recommendations for people looking to enhance this project are to keep everything in subcircuits. It was much easier to work with the Logisim subcircuit module than a massive main circuit. It was also good for multiple group members because we could split up smaller circuits and divide them out accordingly.