Computer Architecture
CE/CS - 321/330

Final Project Report

RISC-V Processor

Nehal Naeem Haji - nh07884
Muhammad Shawaiz Khan - mk07899
Aina Shakeel - as08430

April 22, 2024

Contents

Chapter 1

Introduction

1.1 Objective

The objective of this project was to build a 5-stage pipelined processor capable of executing any one array sorting algorithm. For our project, we made use of the bubble sort algorithm. In most cases, the instructions we have already implemented will allow us to run a sorting algorithm program with minor additions. These included: as implementing bgt or blt instructions to detect when you need to swap two values. This will require some modifications to the circuit.

1.1.1 Sorting Algorithm

We will be using bubble sort to test our processor functionality. Complete Project (RISC V Processor on Github)

Figure 1.1: Registers and Memory

Chapter 2

Tasks

2.1 Task1

2.1.1 RISC V Implementation (single cycle)

We implemented the converted algorithm on the RISC V single-cycle processor developed in our lab. To make the processor compatible with the algorithm, we made some adjustments to the ALU and instruction memory. Specifically, we added a Branch Unit module to support bgt and blt commands used in the algorithm. Additionally, we introduced an addermuxselect input to control the last mux in the processor. These modifications enabled us to run the sorting algorithm effectively on the RISC V processor.

2.1.2 Changes

We made changes in ALU64 bit and added the funct 3's of blt and bge so that their functionality can be added. Moreover, we added the function for slli. Further, we altered the data memory, and instruction memory. Instruction memory was initialised with dummy values initially. We initialised all to 0 so as to make it function properly. The data memory was initialised with array values for sorting. An entire branch unit module was added to manage branch and output a signal addermuxselect which was then ANDED with branch in the toplevelmodule to give the branch decission.

2.1.3 Code and Waveform

The link for the code can be found here.

Figure 2.1: Single Cycle Simulation

2.1.4 Synopsis

To make our sorting algorithm function properly, we introduced changes to the ALU and the final MUX module (by modifying the select line). Additionally, we added a new module called the Branch_module.

2.2 Task2

2.2.1 Testing 5-Stage/Pipelined RISC V Processor for Sorting Algorithm

Here we upgraded our previously developed processor for Pipeline Execution: Integration of 5 Stages and 4 New Modules.

2.2.2 Changes

To implement a pipelined design, the following modules were added to enable the staging to work effectively:

- IF_ID
- ID_EX
- EX_MEM
- MEM_WB

Each stage passes the data and instructions to the next stage through pipeline registers, which help to ensure the proper timing of the data flow and avoid data hazards. These stages and pipeline registers form the backbone of the pipelined processor's operation.

2.2.3 Testing Forwarding Unit

We upgraded our previously developed processor by implementing a Forwarding Unit, which involved adding the Forward Unit module. The forwarding path was then integrated with the pipeline register modules to ensure efficient data flow between stages of the pipeline. These modifications helped to resolve data hazards that may have arisen during the pipelined execution of instructions.

2.2.4 Code and Waveform

The link for the code can be found here.

Figure 2.2: Pipeline with Forwarding

2.3 Task 3

2.3.1 Handling Data Hazards

To address hazards in our circuitry, we need to develop modules for detecting hazards and stalling the pipeline. The detection modules will identify where forwarding is required. Once detected, we can use the forward module to actually perform the forwarding. Additionally, by rearranging non-dependent code statements with stalls, we can optimize our algorithm to work in fewer cycles.

2.3.2 Changes

In computer architecture, hazards refer to conflicts that arise when attempting to execute instructions in a pipelined processor. These conflicts can occur due to dependencies between instructions, leading to stalls or incorrect results.

To address this issue, we added a Hazard Detection Unit module to our processor. This module detects hazards and controls the pipeline by sending signals to flush or stall the pipeline. By adding this module, we improved the processor's efficiency and prevented errors caused by hazards.

The Hazard Detection Unit uses the input flush and stalls to detect hazards in the pipeline. If a hazard is detected, the module signals the pipeline to either flush or stall the pipeline to prevent incorrect results. With this addition, our processor is better equipped to handle hazards and operate more efficiently.

2.3.3 Code and Waveform

The link to the code can be found here.

Figure 2.3: Pipelined RISC-V Processor

2.3.4 Synopsis

Although our 5-stage pipelined processor was functioning well, we encountered stalls in our code due to the requirement of the next value of a register in the subsequent line of code. This necessitates a sequential code and environment, causing the processor to pause for several cycles before resuming the sequence.

To address this issue, we incorporated flush and stall signals into our processor. These signals allow us to mitigate stalls and enable the processor to operate at a faster pace. With this improvement, our processor can continue functioning without delays caused by sequential code.

Chapter 3

Conclusion and Challenges

Building the processor required significant brainpower, and we faced challenges with EDA crashing at critical moments, which taught us a great deal. According to theory, pipelined processors are faster than single-cycle processors because they don't wait for the entire instruction to execute. This makes them highly effective for parallel computing or asynchronous programming.

With better hazard handling, our processor can work even faster and more effectively. Although we faced stalls in our sorting algorithm due to time and complexity limitations, better hazard and stall units can mitigate these issues.

Implementing a real processor through gate-level logic has given us a deeper appreciation for Digital Logic Design.

Chapter 4

Comparison

Pipelined processors are typically designed to be faster than non-pipelined processors, with a potential speedup of up to four times. However, in our case, the pipelined processor is not functioning optimally, with a latency of 3000ns, which is longer than the 1500ns latency of the non-pipelined processor. This suggests that there may be issues with pipeline hazards, control, or other design problems that are affecting the efficiency of your processor. To improve the performance of our pipelined processor, we need to analyze and optimize its design.

Chapter 5

Task Division

- Sadiqah Mushtaq: Bubble Sort Code and Single Cycle Processor
- Lyeba Abid: Pipelined Processor and Debugging
- Ali Muhammad: Pipelined Processor and Debugging
- All Three: Debugging and Integration

Chapter 6

References

[1] HU-LMS. *Canvas*. [Online] Available: https://hulms.instructure.com/courses/2589/discussion_topics/30774
[2] Book. *Course Book*. Computer Organization and Design: The Hardware/-Software Interface RISC-V Edition by David A. Patterson, John L. Hennessy

Appendix A

Appendix

GitHub links for codes of different processors:

- Single Cycle Processor (Task 1): https://github.com/AliMuhammadAsad/Computer-Architecture-Spring-23/tree/main/Single%20Cycle%20RISC%20V%20-%20Project%20Task1
- Pipelined Processor (Task 2): https://github.com/AliMuhammadAsad/Computer-Architecture-Spring-23/tree/main/RISC_V%20Pipelined%20Processor
- Pipelined Processor with Hazard Detection (Task 3): https://github.com/AliMuhammadAsad/Computer-Architecture-Spring-23/tree/main/RISC_V_Pipelined_Complete