EECS151 Final Project Report

Team: newbs
Donaldo Wilson (3035696702)
Xiance Xu (3040942440)

May 9, 2025

Contents

1 Project Functional Description and Design Requirements

Our final design ended up using a total of 5 stages. Gaining insight from the 61C cpu, we split our stages into Instruction Fetch, Instruction Decode, Execute, Memory and Write-back stages. Our memory uses both synchronous reads and writes. To take advantage of this register-like behavior, we place our instruction memory between the Decode and Execute stage. We also place data memory and I/O in between our Execute and Memory stages. Originally, our three stage cpu design could not handle any increase in clock frequency. This is the main reason why we decided to use no forwarding to decrease our critical paths for a 5 stage version. For the sake of simplicity, we decided to delete all forwarding paths to maximize our frequency. After these additions, we found that our current critical path is in our Execute stage, between our immediate generated register and memory.

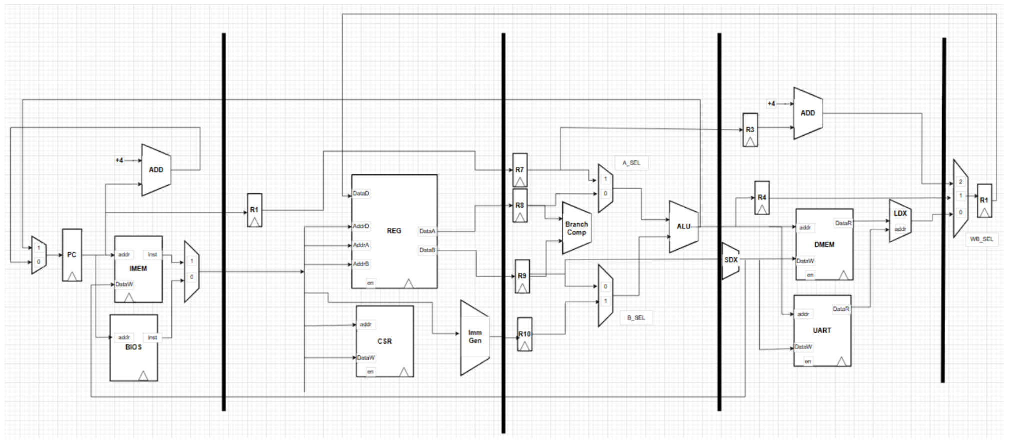
Figure 1: Five-stage CPU pipeline diagram

2 High-level organization

We split our cpu into five stages. The first stage is for incrementing the next pc, and loading it into the address port for instruction memory. The second stage takes in the instruction, and parses its values out to registers, the csr, and the immediate generator. In this stage, we also decode the instruction to determine all of the control signals for the later stages. We use one large case statement that checks the opcode and then calculates the control signals for each function. These control signals then get pipelined throughout each stage to choose outputs for mux's, and determine reads and writes. In the third stage, we execute the instruction. This includes doing combinational logic in the alu. This stage also compares branching values, and adds a no-op control signal to the pc if a branch is taken. In the fourth stage, we write/read from data memory. We use a separate module to format our data depending on if we load/save a byte, a half-word, or a whole word. Finally, in our write-back stage, we send out our data to be written in the register memory.

3 Datapath and Control

Our datapath routes from instruction memory, to registers, then does its computational logic. Finally, it sends its computations to data memory and/or back to register memory. To create all of our control logic for each mux, we decipher our instruction in the second stage(instruction decode). Then, depending on the opcode of the instruction, it will pipeline the controls down the stages. By having a 5 stage pipeline, we could afford having a stage dedicated to creating the control logic for the rest of the cpu. We decided to delete all forms of forwarding to minimize our critical paths. Instead, we use nops, and stalls for all conflicts. Our decode stage compares current registers with ones being sent on the pipeline. Then, we decide to either stall, or proceed with the instruction. For branching, we use the initial assumption of no jumping. We then use a branch comparator to check this and perform a nop in the third stage to clear the pipeline if needed.

4 Verification

To test our cpu, we used every single provided test. Thankfully, every bug we encountered was detected with these tests and allowed us to eventually perform instructions on the FPGA. Over the course of this project, we encountered numerous bugs. These bugs included lots of bad logic in our system. One of the biggest bugs that we had in the beginning was getting our pc register logic correct. We kept failing tests due to skipping instructions right after reset. To remedy this, we looked at both GTKwave, and DVE waveforms to analyze our traces. Then were eventually able to find a solution of clocking the reset signal to persist until a positive clock edge. Other bugs that we encountered were creating latches in our logic. We found these bugs after running synthesis. To fix these, we realized all of our case logic was incomplete, and we needed to simplify our pc counter logic. When rewriting our cpu to a five stage pipeline, we also used waveforms to both test and visualize our logic for jumps and branches. Another lengthy bug that we had was failing our cpu_tests. This, we found out to be actually an error in our testbench, and we need to extend the timeout of each test inorder to have it properly work.

```
SV Assert: "assert property (@(posedge clk) (reg_ra1 == 5'd0) i-> (reg_rd1 == 32'd0));"
```

This assertion states that when we try to read from the 0th register, our output should always be zero. We also mimic this assertion for the second read port of the register memory.

5 Status and Results

Status	Fully Working
CPI (after Checkpoint 2/3)	25.42
FPGA usage	1293 LUTs, 418 SLICE registers, 34 BRAMs, 0 DSP Blocks
Critical Path Length	0.106ns
Best CPI	1.24
Clock Frequency	95 MHz
Figure of Merit	58.57
Best time/program	`33757*1.24*1.05*1e-8`

Table 1: Project Performance Summary

5.1 Critical Paths

During the optimization process, we have had to try and overcome many different and recurring critical paths. These paths range from our original forwarding control signals. For instance, our first critical path was from reading the instruction in instruction memory, through the alu, and back into the first stage of our cpu in the pc register. Another critical path that we had to optimize was routing from instruction memory to our branch comparator, and then back into our program counter select wire. The next critical path that we had to optimize was moving through our immediate generator, through our b mux, and through the alu to data memory. And the last and current critical path that we have encountered is our execution stage, taking in our pipelined register values, and pushing them into our data memory.

5.2 Optimizations

For our cpu, we have attempted numerous optimization techniques in order to increase our figure of merit. Our methodology was mainly based on maximizing our clock frequency to run faster. This did hurt our CPI performance, and we had to find a tradeoff that increased the FoM in our favor. Our first original three stage design ran at 50 MHz and had a figure of merit at around 25. To tackle this, our first optimization was adding two stages to our cpu. We followed the design from 61C, and decided to try and not add forwarding paths. The reason for this is that our forwarding logic was our original critical path. Our 5 stage design was able to run at 90 MHz. To further optimize our cpu, we then moved around our branch comparator to the execute stage. This minimized our critical path, but was not able to run at any higher frequency. Our next step was moving our immediate generator to the instruction decode stage. This shortened our critical path and allowed us to run at 95 MHz. After this, we tried to condense our control logic from a separate module, to their own assigned statements. Unfortunately, this sometimes increased our critical path and stopped our processor from working at 95MHz. Another optimization method we used was to minimize the bits for each wire. We changed our logic to use specific bits from our instruction instead of giving a full opcode or funct3. This helped with our cost, but did not allow us to push our FoM by much. Another technique we tried was adjusting our implementation script. We tried adding mux optimizations and rerouting logic. Unfortunately, this also increased our critical path such that we had negative slack.

6 Conclusion

Overall, this project was very fun,and full of challenges! Especially at the end during our optimization phase. From it, we learned how to create neat code to make our debugging lives easier. We also learned how to make our code very customizable, such that we can add and remove logic when needed.

I really appreciate the nights in Cory, Debugging days seemed particularly hard to bear,but now looking back into at the path we've taken, I'm so proud of my teammate :Donaldo Wilson, he did a great job in every parts of this project, being a visiting student, I'm so fortunately to meet such a good team member, the days when I'm hopeless and despair, he just like the hero in Marvel,we help others to finish this unforgettable journey!!! I'll remember these moments forever ! I sincerely hope that we have a chance in future to work together. Lastly, best wishes for our bright future.