Lab 3 Report

Adrian Saldana & Noor Fakih

board EE180-23z

**Assignment/Problem Description:**

Lab 3 required us to build a working 5 stage pipelined MIPS processor that is capable of executing 44 different kinds of MIPS instructions using the hardware description language Verilog. The goal of the assignment was to build a deeper understanding of processors, pipelining, MIPS, forwarding, stalling and general hazard control.

**Discussion**:

*Solution/Program Description*

There are three major components to actually completing the assignment:

1. Implement the following MIPS instructions (all of which has been implemented and tested)

1. add

2. addiu

3. addu

4. addi

5. beq

6. bgez

7. bgtz

8. blez

9. bltz

10. bne

11. slti

12. slt

13. sltu

14. sltiu

15. j

16. jal

17. jalr

18. jr

19. lui

20. lw

21. ll

22. lb

23. lbu

24. and

25. andi

26. ori

27. nor

28. xori

29. xor

30. or

31. movn

32. movz

33. mul

34. sll

35. sra

36. srav

37. srl

38. sllv

39. srlv

40. sc

41. sb

42. sw

43. sub

44. subu

1. Add hazard detection, forwarding and stalling to the processor in order to avoid issues like using stale data due to a Read After Write (RAW) hazard.
2. Demonstrate the completed processor by running a Sobel filter program by using our MIPS processor and assigned lab board

*Any Implementation Issues*

Due to our unfamiliarity with pipelines and proper hazard control, it was initially difficult to understand how to even begin and where to look but otherwise it turned out fine.

*Known Bugs and/or Errors*

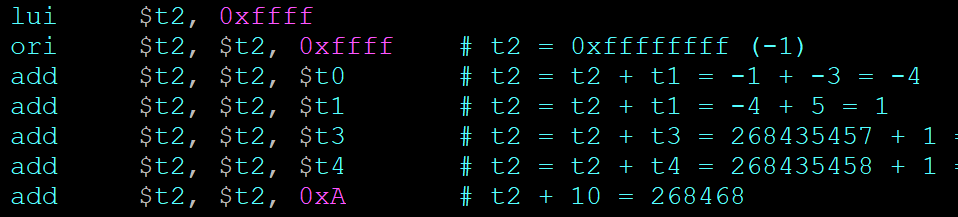
Since we were given starter code that created the five-stage pipelined processor with the pipeline registers already inserted, we were able to start with bug free code. After extensive testing, we don’t believe our design has any bugs and/or errors.

**Test Description and Results**

With our testing, we looked to cover as many possible variations and cases as possible to root out any hiding issues such as incorrect MIPS instruction implementation, RAW hazards or incorrect stalling/forwarding.

*Example test case*

Repeatedly using the same register to read and write

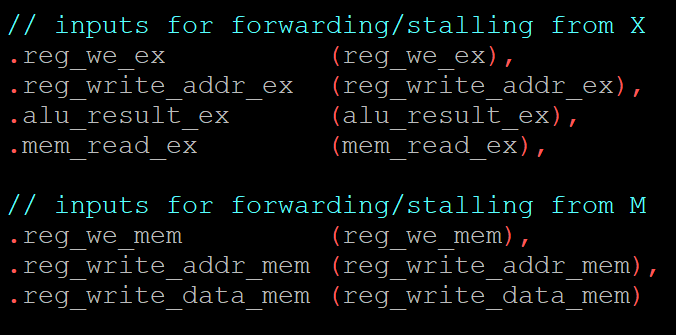


In this test case, we ensure that no stale data is being used (no RAW hazard) by repeatedly reading and writing to $t2 in each instruction. If this test had failed, it would’ve indicated either an issue with add’s implementation or that the processor is not properly forwarding. Additional RAW hazards were checked for in

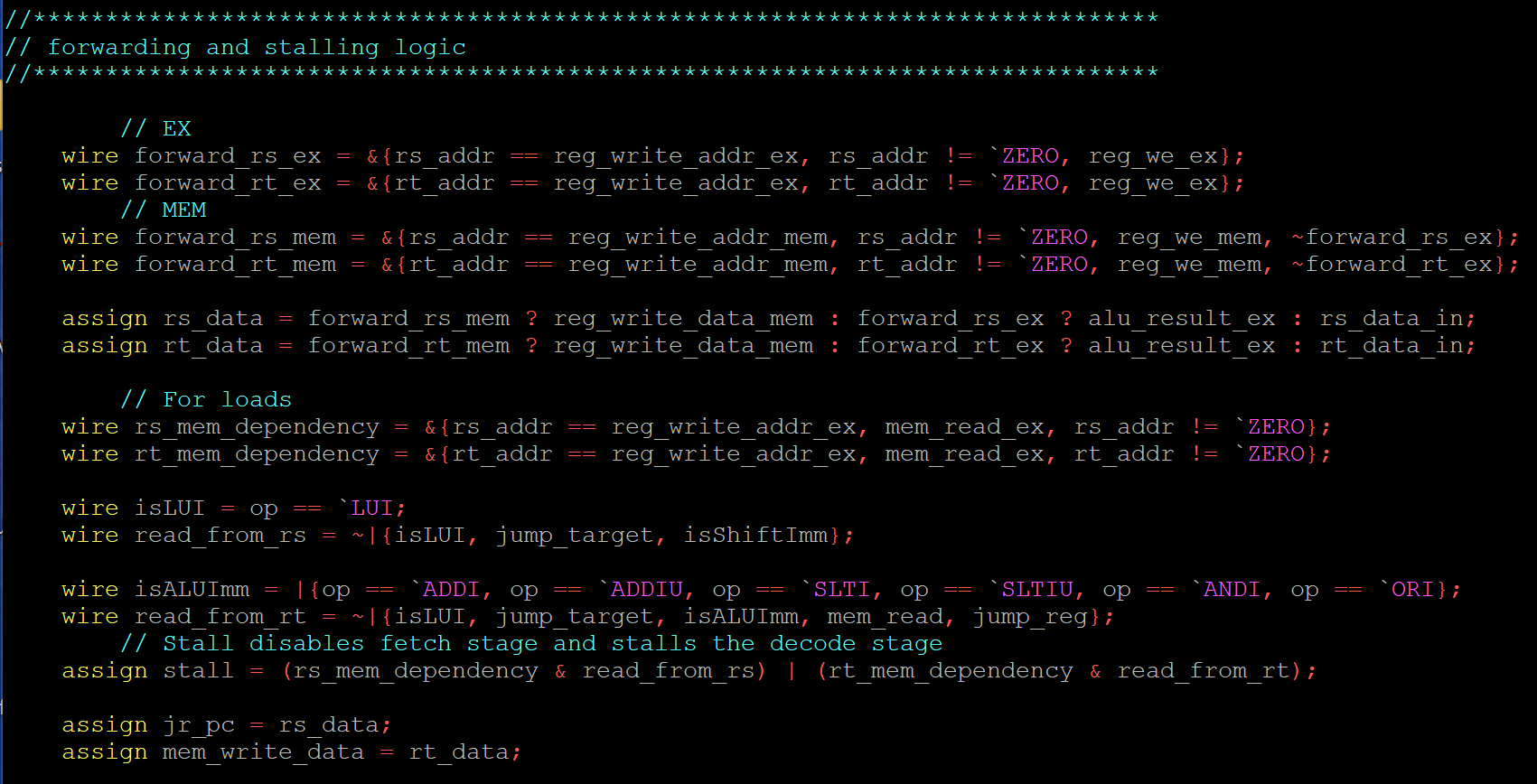
*Hazard Detection, Forwarding & Stall Control*

In order to handle hazard detection, forwarding and stalling, we had to make changes to the control and datapath. The only files we had to alter to gain this functionality was decode.v (Instruction Decode - stage 2) and the portion of mips\_cpu.v (Memory - stage 4) where the dcode mudle is instantiated as all forwarding and stalling occurs from the ID stage.

The first change was ensuring that all the correct inputs from mips\_cpu.v for forwarding and stalling from the Execute stage and Memory stage were being passed to decode.v where the forward/stall actually happens and that all the flip-flops were set up correctly.



In decode.v, some of the above inputs from mips\_cpu were never used or incorrectly utilized.



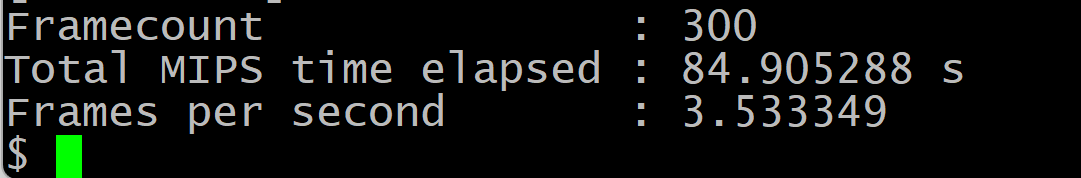
Changes in decode.v:

* Added forward\_rs\_ex and forward\_rt\_ex to create functionality for forwarding/stalling for the values in the execute stage
* Adjusted forward\_rs\_mem and forward\_rt\_mem to acknowledge that it should not forward from memory if the execute stage still needs to forward its data.
* Adjusted rs\_data and rt\_data so $rs and $rt will hold the correct data depending on what function is being executed (ex. if it was an ALU function, it will assign the registers to alu\_result instead of the original data value input).
* Corrected the stall variable to acknowledge the $rt register.

*Synthesis*

After validating our processor design we moved to synthesizing. As you can see, our implementation correctly applies a sobel filter to the baxter.avi video file. We were able to achieve 3.5 FPS, which is on par with lab 2 starter code.





**Lessons Learned/Epilogue:**

For this lab, we were able to finish the five-staged pipelined MIPS processor skeleton code we were given and have it properly synthesize and run the sobel\_calc program. We gained a solid understanding of processors, pipelining, forwarding, stalling, and hazard detection. We also became intimately familiar with the MIPS instructions through implementing and testing them, so we definitely achieved the learning objectives of the lab.

This lab was difficult, especially when initially starting due to misunderstandings like: we were writing Verilog without using a software program like Vivado, we didn't know how to set up waveforms so they’re helpful, didn’t catch fact that we could just start trying to implement the MIPS functions and that it wouldn't break the whole program because there wasn’t hazard control yet, etc.

*Future Improvements*

The program that displays the waveforms is really subpar compared to Vivado. The font is so small and the image quality is awful so it makes it really hard to read. If that could be somehow changed or updated, it would be greatly appreciated for future students taking the course. It would have been helpful if during the lab 3 review session, there was less emphasis that we are coding using Verilog and show demonstrations such as how to access the waveforms and how to find the signals, how to synthesize, etc. Also, there were questions posted on Ed that were either completely unanswered or answered after days. For example, there was one question someone else had posted that we would have benefited from: [post #169](https://edstem.org/us/courses/16226/discussion/1146401). It went unanswered for days until the original poster reposted it again: [post #185.](https://edstem.org/us/courses/16226/discussion/1155993) That post went unanswered for 3 days until a fellow student answered it. It has been 7 days since the original post and there is no answer endorsed by the teaching staff or their own response. Taking into consideration that this is a long and difficult project which we were given about 2 weeks to do, a better response time would be very much appreciated.