EE 180
Winter 2022
Lab 3

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	16.	jal	31.	movn
2.	addiu	17.	jalr	32.	movz
3.	addu	18.	jr	33.	mul
4.	addi	19.	lui	34.	sll
5.	beq	20.	lw	35.	sra
6.	bgez	21.	ll	36.	srav
7.	bgtz	22.	lb	37.	srl
8.	blez	23.	lbu	38.	sllv
9.	bltz	24.	and	39.	srlv
10.	bne	25.	andi	40.	sc
11.	slti	26.	ori	41.	sb
12.	slt	27.	nor	42.	sw
13.	sltu	28.	xori	43.	sub
14.	sltiu	29.	xor	44.	subu
15.	j	30.	or		

2. 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.

3. 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

```
lui     $t2, 0xffff
ori     $t2, $t2, 0xffff    # t2 = 0xffffffff (-1)
add     $t2, $t2, $t0       # t2 = t2 + t1 = -1 + -3 = -4
add     $t2, $t2, $t1       # t2 = t2 + t1 = -4 + 5 = 1
add     $t2, $t2, $t3       # t2 = t2 + t3 = 268435457 + 1
add     $t2, $t2, $t4       # t2 = t2 + t4 = 268435458 + 1
add     $t2, $t2, 0xA       # t2 + 10 = 268468
```

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

EE 180

Winter 2022

Lab 3

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.

```
// inputs for forwarding/stalling from X
.reg_we_ex           (reg_we_ex),
.reg_write_addr_ex   (reg_write_addr_ex),
.alu_result_ex       (alu_result_ex),
.mem_read_ex         (mem_read_ex),

// inputs for forwarding/stalling from M
.reg_we_mem          (reg_we_mem),
.reg_write_addr_mem  (reg_write_addr_mem),
.reg_write_data_mem  (reg_write_data_mem)
```

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

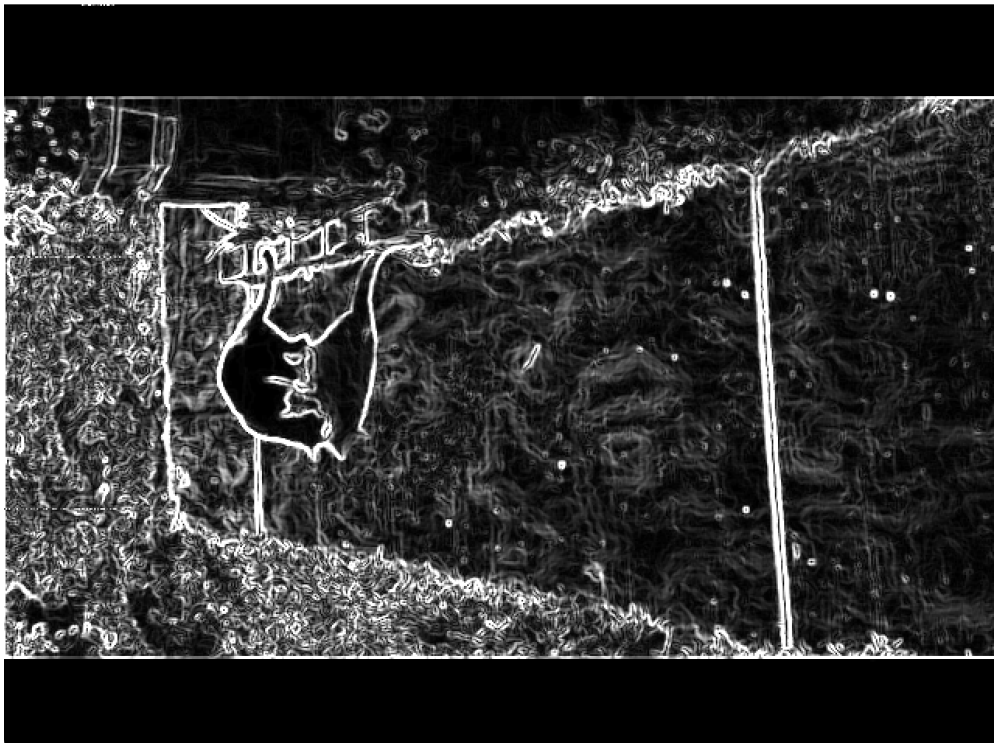
```
//*****************************************************************************
// forwarding and stalling logic
//*****************************************************************************

        // EX
    wire forward_rs_ex = &{rs_addr == reg_write_addr_ex, rs_addr != `ZERO, reg_we_ex};
    wire forward_rt_ex = &{rt_addr == reg_write_addr_ex, rt_addr != `ZERO, reg_we_ex};
        // MEM
    wire forward_rs_mem = &{rs_addr == reg_write_addr_mem, rs_addr != `ZERO, reg_we_mem, ~forward_rs_ex};
    wire forward_rt_mem = &{rt_addr == reg_write_addr_mem, rt_addr != `ZERO, reg_we_mem, ~forward_rt_ex};

    assign rs_data = forward_rs_mem ? reg_write_data_mem : forward_rs_ex ? alu_result_ex : rs_data_in;
    assign rt_data = forward_rt_mem ? reg_write_data_mem : forward_rt_ex ? alu_result_ex : rt_data_in;

        // For loads
    wire rs_mem_dependency = &{rs_addr == reg_write_addr_ex, mem_read_ex, rs_addr != `ZERO};
    wire rt_mem_dependency = &{rt_addr == reg_write_addr_ex, mem_read_ex, rt_addr != `ZERO};

    wire isLUI = op == `LUI;
    wire read_from_rs = ~|{isLUI, jump_target, isShiftImm};

    wire isALUImm = |{op == `ADDI, op == `ADDIU, op == `SLTI, op == `SLTIU, op == `ANDI, op == `ORI};
    wire read_from_rt = ~|{isLUI, jump_target, isALUImm, mem_read, jump_reg};
        // Stall disables fetch stage and stalls the decode stage
    assign stall = (rs_mem_dependency & read_from_rs) | (rt_mem_dependency & read_from_rt);

    assign jr_pc = rs_data;
    assign mem_write_data = rt_data;
```

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.



```
Framecount                : 300
Total MIPS time elapsed : 84.905288 s
Frames per second         : 3.533349
$
```

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.

EE 180
Winter 2022
Lab 3
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. It went unanswered for days until the original poster reposted it again: post #185. 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.