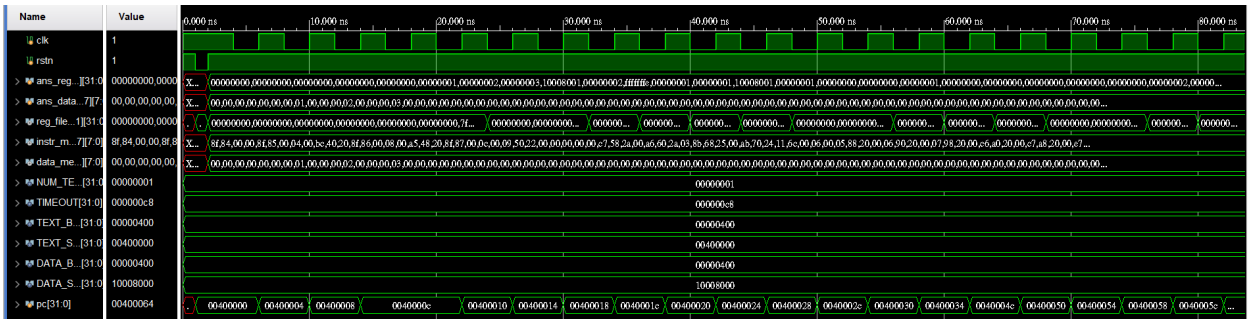
CO_lab4_111550076

1. Experimental Result

1. Show the waveform screen shot of the test we provided.




```
#### Test Result ####
Passed  1 : 0
Failed  0 :
#### all passed!
```

2. What other testcase you've tested? Why you choose them?

I tested several features including beq forwarding (one stall and two stalls), lw-sw forwarding, addi, and some stall operations, and ensured that the operation immediately following beq is executed.

```
main:   lw      $a1, 4($gp)
        lw      $a0, 0($gp)
        lw      $a2, 8($gp)
        add     $t1, $a1, $a1
        add     $t0, $a1, $gp
```

```
        lw      $a3, 12($gp)
        sub     $t2, $0, $t1
        addi    $t7, $a1, 200 #addi
        nop
        slt     $t3, $a2, $a3
        slt     $t4, $a1, $a2
        lw      $t7, 4($gp)
        addi    $t1, $t1, 10  #addi
        addi    $t2, $t2, 20  #addi
        and     $t6, $a1, $t3
        or      $t5, $gp, $t3
        lw      $t4, 4($gp)
        sw      $t4, 0($gp) #lw sw forward
        add     $t7, $t7, $t7
        add     $t4, $t4, $t4   #beq forward
        beq     $t7, $t4, end   # taken
        add     $s1, $0, $a1    #being executed
        add     $s2, $0, $a2
        add     $s3, $0, $a3
        add     $s4, $a2, $a2
        add     $s5, $a2, $a3
        add     $s6, $a3, $a3
end:    add     $s7, $s4, $a2
```

2. Answer the following Questions

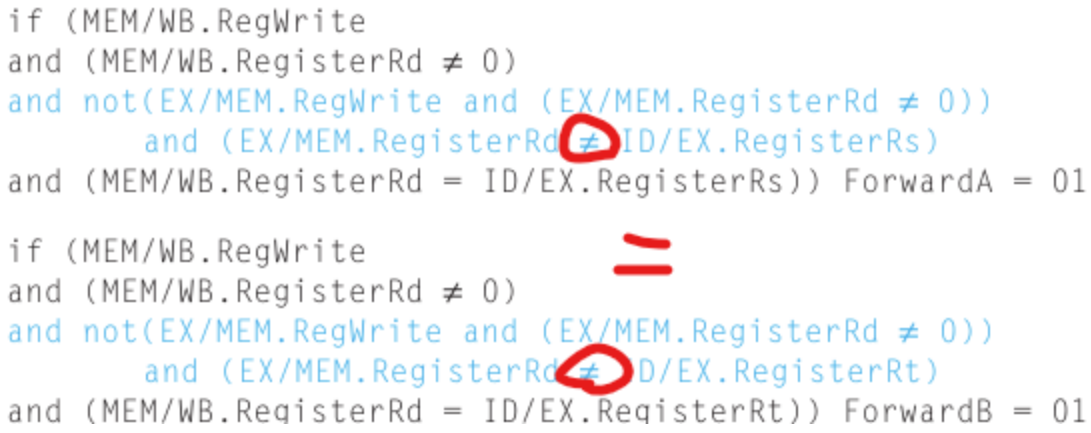
1. (%) List out the equation to detect EX & MEM hazard in forwarding unit. Which part of the equation in textbook p.369 is wrong?

When I implemented lw-sw forwarding, I found that the lw-sw operation would incorrectly trigger `forward_B` to `2'b10`. To address this, I manually added the

`ID_EX_mem_write` signal to ensure that if the next operation is `sw`, forwarding is not triggered.

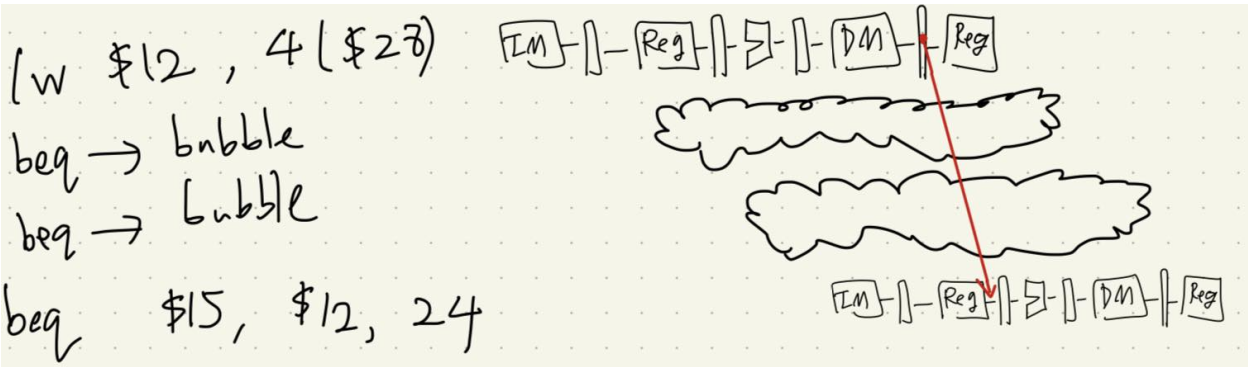
```
//EX forward
if(EX_MEM_reg_write && (EX_MEM_rd != 0) && (EX_MEM_rd == ID_EX_rs)) begin
    forward_A = 2'b10;
  end
  //lw forward
  else if(MEM_WB_reg_write && (MEM_WB_rd != 0) && (MEM_WB_rd == ID_EX_rs)) begin
    forward_A = 2'b01;
  end
  else begin
    forward_A = 2'b00;
  end
  // Forward B
  //EX forward
  if (~ID_EX_mem_write && EX_MEM_reg_write && (EX_MEM_rd != 0) && (EX_MEM_rd == ID_EX_rt)) begin
    forward_B = 2'b10;
  end
  //MEM forward
  else if (MEM_WB_reg_write && (MEM_WB_rd != 0) && (MEM_WB_rd == ID_EX_rt)) begin
    forward_B = 2'b01;
  end

  else begin
  forward_B = 2'b00;
  end
```

The part textbook wrong is the following equation, the red curcle part should be =, not ≠.



2. In forwarding for beq, is forwarding from MEM/WB to ID needed? Why?

Yes, If the `rs` or `rt` of the `beq` instruction is equal to the `rs` of the previous `lw` instruction, and the comparison for the `beq` instruction is moved to the ID stage, it still needs to wait for the `lw` instruction to write back to the register before the comparison can be made. This would cause more stalls. Implementing MEM/WB -> ID forwarding can reduce one stall.



3. Briefly explain how you insert stalls when `beq` reads registers right after `lw` writes it.

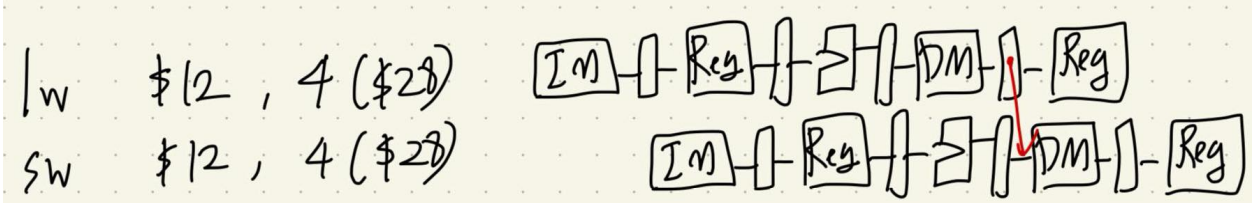
I found that the memory read control signal can only be triggered by the `lw` operation, so I use it to determine whether the stage is operating an `lw` instruction. Therefore, I designed a `branch_hazard_2` signal in `hazard_detection.v` to determine that if the EX stage or MEM stage is operating an `sw` and the write register of `sw` is equal to the `rs` or `rt` of `beq`, a stall will occur. Consequently, if `sw` is in the EX stage, it will stall, and in the next clock cycle, when `sw` is in the MEM stage, it will also stall. This results in two stalls.

```
wire branch_hazard_2 =  take_branch &&
    (ID_EX_mem_read && (IF_ID_rs == ID_EX_rt || IF_ID_rt == ID_EX_rt)
    || (EX_MEM_mem_read && (IF_ID_rs == EX_MEM_rd || IF_ID_rt == EX_MEM_rd)));
wire hazard = lw_hazard || branch_hazard_1 || branch_hazard_2 ;
assign stall = hazard;
assign pc_write = ~hazard;
assign IF_ID_write = ~hazard;
```

4. `sw` right after `lw` is quite common since copy and paste a data from one address to another is used frequently. In textbook, a stall is followed by the lw in

this case. Is it possible to remove this stall? How?

Yes, I added a `sw_forward` module in the MEM stage and found that `mem_write` is only triggered by the `sw` instruction, and `mem_to_reg` is only triggered by the `lw` instruction. I use this to determine if an `sw` follows directly after an `lw`. If the `rt` of `sw` is equal to the `rd` of `lw`, forwarding is needed. Thus, I use a signal to make `data_mem_write_data` become the WB stage result data. However, I discovered that the `lw-sw` operation can cause issues in `hazard_detection.v` and `forwarding.v`. To address this, I added signals to ensure that other forwarding and hazards only occur when the instruction is not `sw`.




```
// in sw_forward module
if(EX_MEM_mem_write && MEM_WB_mem_to_reg && EX_MEM_rt != 0 && EX_MEM_rt== MEM_WB_rd)begin
        sw_forward <= 1;
    end
    else begin
        sw_forward <= 0;
    end
//in pipelined.v
assign wb_result_data = MEM_WB_mem_to_reg ? MEM_WB_read_data : MEM_WB_ALU_result;
assign data_mem_write_data = sw_forward ? wb_result_data : EX_MEM_read_data2;
```