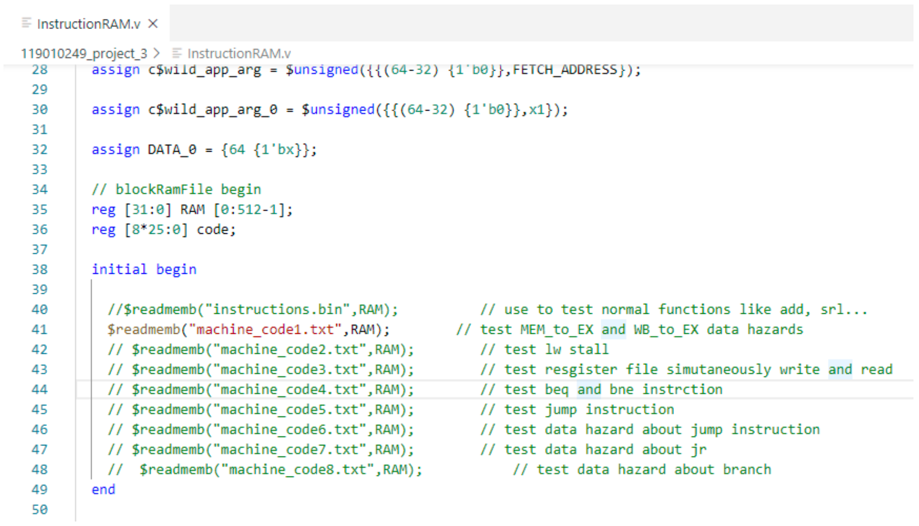
Project 3 Report

In this project, a pipelined 5-stage CPU solving hazards is designed and implemented. Before I implemented the CPU by Verilog, I first came up with a basic idea of the whole CPU. This CPU includes basically 8 parts: (1). IF stage (2). ID stage (3) EX stage (4) MEM stage (5) Wb stage (6) 4 * pipeline registers (7) Forwarding Unit (8) Hazard Detection Unit. These 8 basic parts cooperate with each other and form a CPU solving all the hazards in 8 test files.

There is a big picture thought of how instructions are executed in this CPU:

[0] How to use test_cpu.v:

Before the execution, the users should modify the InstrutionRAM file to input the test file they like (as the picture showing below). Then input 'make' in the terminal. Further, input './CPU' in terminal to start the execution of instructions.



```
assign c$wild_app_arg = $unsigned({{(64-32) {1'b0}},FETCH_ADDRESS});

assign c$wild_app_arg_0 = $unsigned({{(64-32) {1'b0}},x1});

assign DATA_0 = {64 {1'bx}};

// blockRamFile begin
reg [31:0] RAM [0:512-1];
reg [8*25:0] code;

initial begin

  //$readmemb("instructions.bin",RAM);          // use to test normal functions like add, srl...
  $readmemb("machine_code1.txt",RAM);       // test MEM_to_EX and WB_to_EX data hazards
  // $readmemb("machine_code2.txt",RAM);        // test lw stall
  // $readmemb("machine_code3.txt",RAM);        // test resgister file simutaneously write and read
  // $readmemb("machine_code4.txt",RAM);        // test beq and bne instrction
  // $readmemb("machine_code5.txt",RAM);        // test jump instruction
  // $readmemb("machine_code6.txt",RAM);        // test data hazard about jump instruction
  // $readmemb("machine_code7.txt",RAM);        // test data hazard about jr
  //  $readmemb("machine_code8.txt",RAM);           // test data hazard about branch
end

```

[1] IF stage

In IF stage, if there is a branch signal or jump signal, the fetch address will be assigned the branch address. Otherwise, it will be assigned the value of PC. Then PC will add itself 1, meanwhile, the instruction random access memory will output the instruction to the pipeline registers IF_ID according to the fetching address.

[2] ID stage

In this stage, if there is no hazard signals it will receive the instruction output from IF_ID pipeline registers. Then it will parse the instruction to get necessary address or value, like OP, rs address and rt address. The OP and funct will be sent to the control unit, and the control unit will output the corresponding control signals to the pipeline registers ID_EX.

Meanwhile, the register address will be recognized by the registers file in ID stage and get the corresponding value in register. The values together with the write back address will be output to the next pipeline register ID_EX.

To reduce the stall cycles when facing j and jal instructions, j and jal are recognized in this stage, and the jump signal and address will output to the IF stage.

[3] EX stage

In this stage, if there are no hazard signals, it will receive the values in registers and the control signals from ID_EX pipeline registers. The ALU will conduct the corresponding operation according to the control signals and output the ALU result to next pipeline registers EX_MEM. This stage will also handle the beq, bne and jr instructions.

[4] MEM stage

In this stage, the main memory will receive the corresponding control signals and fetching address or edit serial from the pipeline registers EX_MEM. If the enable signal is 1, it can read data from memory. If the write signal is 1. The value of rt will be stored into the memory. After the memory operations, the ALU result and the value loaded from main memory (if it is lw instruction) will be output to next pipeline registers MEM_WB.

[5] WB stage

This stage will decide what values (from memory or ALU result) shall be wrote back the register files according to the control signal. And then write it back to the register file according to the write back address.

[6] Pipeline registers

I use four pipeline registers to isolate each stage. During each positive edge of CLOCK, the pipeline registers will output the values and signals.

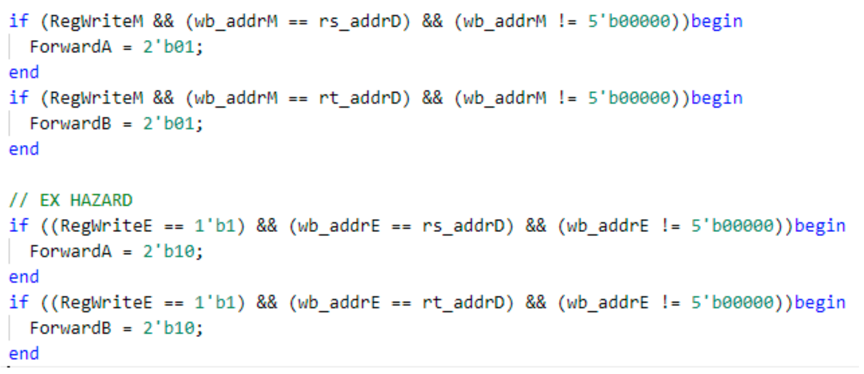
[7] Solution of Hazards

1. structure hazard

I isolation the read and write operation of the registers file. In the stage of WB, I will write the value to the register file when positive edge of CLOCK. In the stage of ID, I will read the data from registers file when the negative edge of CLOCK.

2. Data hazard

I implement a Forwarding unit to detect and solve the data hazard (MEM hazard and EX hazard).



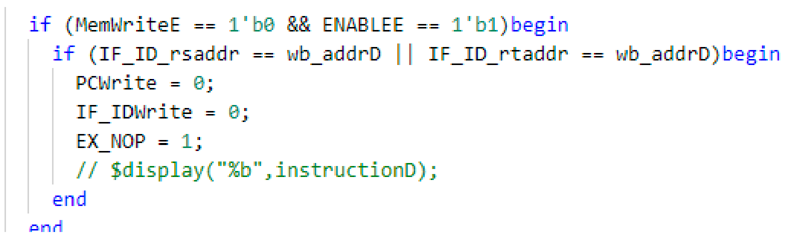
```
if (RegWriteM && (wb_addrM == rs_addrD) && (wb_addrM != 5'b00000))begin
  ForwardA = 2'b01;
end
if (RegWriteM && (wb_addrM == rt_addrD) && (wb_addrM != 5'b00000))begin
  ForwardB = 2'b01;
end

// EX HAZARD
if ((RegWriteE == 1'b1) && (wb_addrE == rs_addrD) && (wb_addrE != 5'b00000))begin
  ForwardA = 2'b10;
end
if ((RegWriteE == 1'b1) && (wb_addrE == rt_addrD) && (wb_addrE != 5'b00000))begin
  ForwardB = 2'b10;
end
```

Forward A and Forward B will be sent to EX stage. The last 2 ALU result from the pipeline registers EX_MEM, MEM_WB, will be sent to the EX stage EX stage to modify the operands.

3. LW hazard

This hazard will be recognized and handled by the Hazard Detection Unit. After detecting this hazard, the unit will send signals to IF_ID, ID_EX to insert a NOP to stop these two stages for a cycle. After one cycle, the hazard can be completely solved by forwarding.



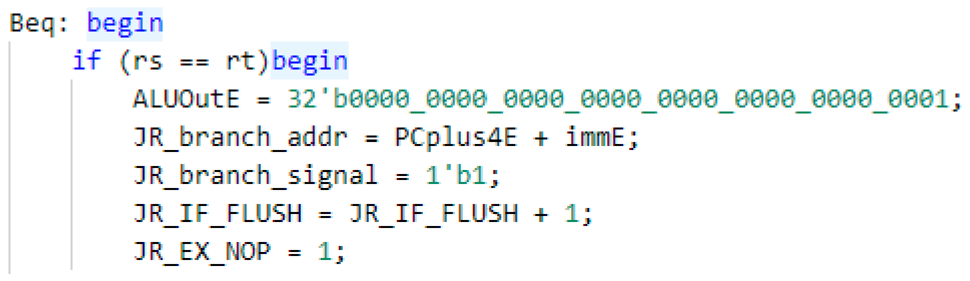
```
if (MemWriteE == 1'b0 && ENABLEE == 1'b1)begin
  if (IF_ID_rsaddr == wb_addrD || IF_ID_rtaddr == wb_addrD)begin
    PCWrite = 0;
    IF_IDWrite = 0;
    EX_NOP = 1;
    // $display("%b",instructionD);
  end
end
```

4. J and Jal hazard

In order to reduce the CPI, J and Jal are handled in ID stage. Then it will send a flush signal to IF_ID pipeline registers to flush the instruction next to the J and Jal instructions.

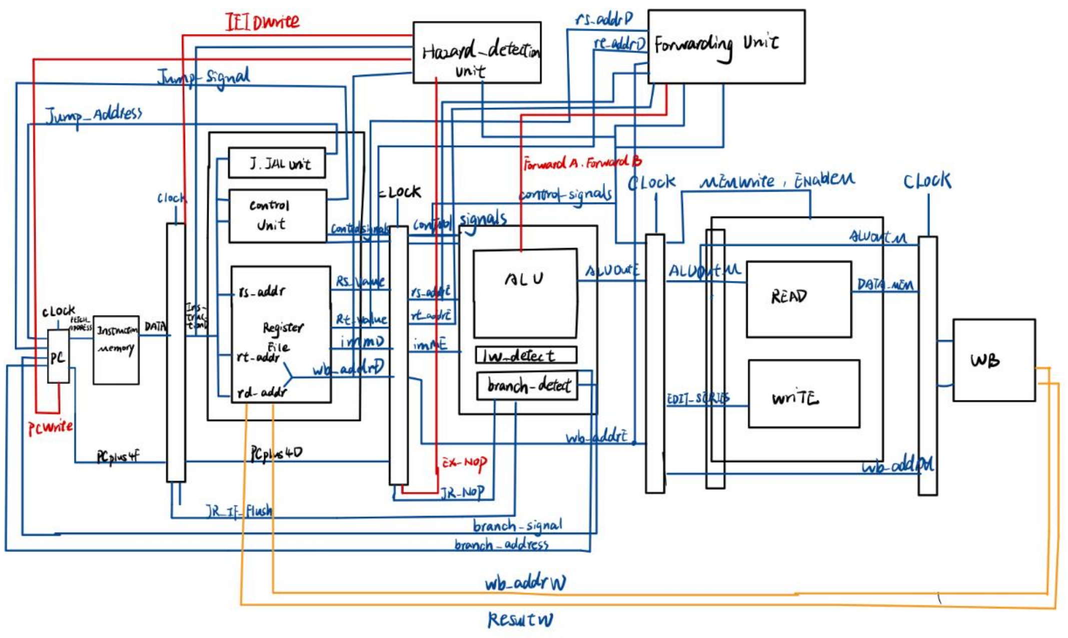
5. Beq, Bne, Jr hazard

This hazard is solved by stalling the next two stages. Once stall, ID_IF and ID_EX pipeline registers will output invalid signals.



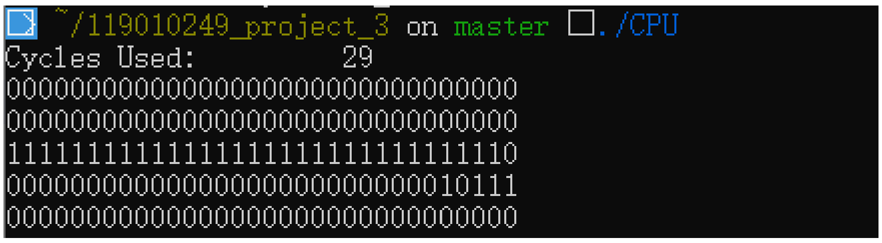
```
Beq: begin
    if (rs == rt)begin
        ALUOutE = 32'b0000_0000_0000_0000_0000_0000_0000_0001;
        JR_branch_addr = PCplus4E + immE;
        JR_branch_signal = 1'b1;
        JR_IF_FLUSH = JR_IF_FLUSH + 1;
        JR_EX_NOP = 1;
```

[Data Flow Chart]



[Sample Output]

Except the values in main memory, it will also display the total cycles used:



```
 ~/119010249_project_3 on master  ./CPU
Cycles Used:          29
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111111111111111110
00000000000000000000000000010111
00000000000000000000000000000000
```

(output sample of machine_code_8.txt)