Final Project Report

Pipelining the RISC-V Processor

By Umesh Tanniru ID: 57164808

and Mohammed Haque ID: 62655407

Implementation of the Pipeline

There are several additions and adjustments needed to be implemented to create a RISC-V Pipeline Processor. The first was the addition of four new flops that acted as large pipes. These flop registers reacted only on the rising edge of the clock cycle. We also implemented the Forwarding and Hazard Detection Systems. Lastly, we had to adapt the processor to add NOPs in cases of loading and branching.

The Pipelines (IFIDFlop.sv, IDEXFlop.sv, EXMEMFlop.sv, MEMWBFlop.sv):

- We implemented large flops that take in the data from the previous section and allows it to continue after the rising edge of the clock cycle. This way we created the structure and implementation of the pipes.
- We created four new files for this implementation: IFIDFlop.sv, IDEXFlop.sv, EXMEMFlop.sv, and MEMWBFlop.sv

Forwarding

- The Forwarding units is placed in the EX stages, where it takes the inputs are Reg1AddrEX, Reg2AddrEX, WriteAddrMEM, WriteAddrWB. And the outputs are ForwardA[1:0] and ForwardB[1:0]. This forwarding unit removes the problem of delayed updates to the registers and make sure that the instructions get the right updated data inputs.
- We implemented this by creating the Forewarding.sv file where it calculates what ForwardA and ForwardB should be.

Hazard Detection:

- Hazard Detection happens in the ID stage where it predicts a potential load and used of the write-address right after.
- The inputs are: MemReadEX, WriteAddrEX, Reg1AddrID, Reg2AddrID. And the only output is Hazard. Hazard becomes 0, when the data hazard is detected, and Hazard is 1 when no data hazard is detected. This way Hazard can be directly connected to the enable of the PC and IFID flops to stall the processor for one clock cycle.
- Also, all the control inputs going into the IDEX flop, goes through the Hazard Detection unit. Therefore, the Hazard Detection can directly modify the values of the controls to 0 when a hazard is detected.
- We implemented this unit by creating the HazardDetection.sv file where it conducts the operations explained above.

NOP Adaptation for Branch Instructions:

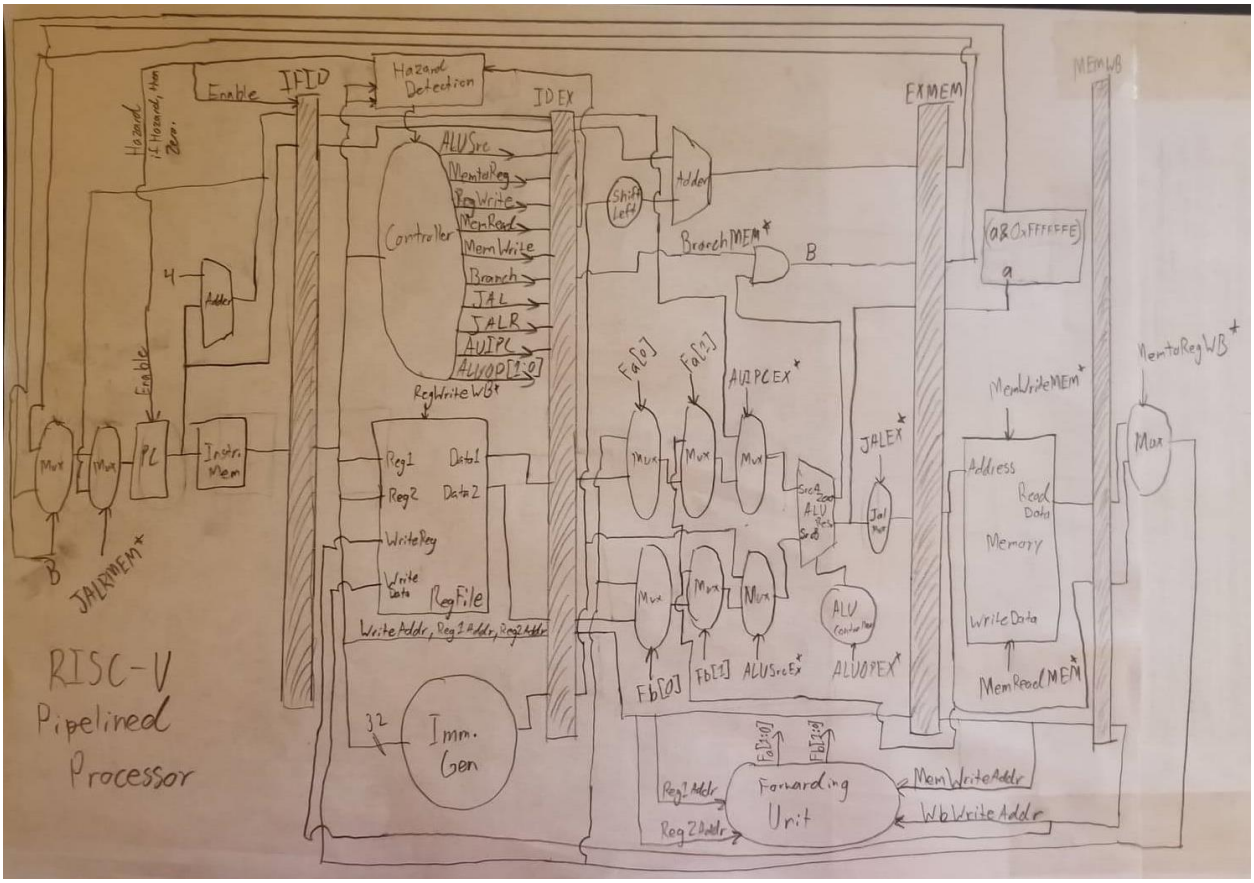
- During Load instructions, the hazard detection unit sends one NOP to counteract the data hazard issue.
- During the Branch instructions, two NOPs are sent if the Branch condition is true:

- If the Branch is true (B = Zero & BranchMEM), the B value will directly go to the reset value of the IFID and IDEX flops and erase the instructions happening before. The PC register will naturally get updated to the newest instruction. **There will be two NOPs in the outputs! (Ex. Branch, NOP, NOP, Instr)**
- If the there is no Branch, the B value will be 0 and the Pipeline will resume normal execution.

All this combined gives a functioning RISC-V Pipeline Processor.

RISC-V Pipelined Processor Datapath

This is the schematic for the RISC-V Pipelined Processor that was implemented. The addition of four Pipelines and four muxes is visibly shown. For clarity purposes, all outputs of the Controller are connected to their associated names in asterisks. For example, ALUSrc is connected to *ALUSrc. Also, the stage of when the control value was used will be written after the name. For example, the MemtoReg control value is used at the end before the data is sent into the RegFile, so the name on the paper will be MemtoRegWB* because the control value is used in the WriteBack stage.



Example Test Code

We used the instructionmemeory.sv provided by the TA's in the Lab3 folder. There are a total 57 instructions that were used that tested all types of instructions: Branch, JAL, JALR, AUIPC, LUI, R-Type, and R-TypeI. Here is a list of those instructions and their correct outputs:

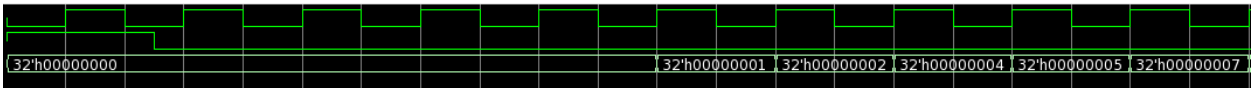
```
32'h00007033;//      and  r0,r0,r0           ALUResult = h0 = r0
32'h00100093;//      addi r1,r0, 1           ALUResult = h1 = r1
32'h00200113;//      addi r2,r0, 2           ALUResult = h2 = r2
32'h00308193;//      addi r3,r1, 3           ALUResult = h4 = r3
32'h00408213;//      addi r4,r1, 4           ALUResult = h5 = r4
32'h00510293;//      addi r5,r2, 5           ALUResult = h7 = r5
32'h00610313;//      addi r6,r2, 6           ALUResult = h8 = r6
32'h00718393;//      addi r7,r3, 7           ALUResult = hB = r7
32'h00208433;//      add  r8,r1,r2           ALUResult = h3 = r8
32'h404404b3;//      sub  r9,r8,r4           ALUResult = hfffffffe= r9
32'h00317533;//      and r10 = r2 & r3       ALUResult = h0 = r10
32'h0041e5b3;//      or  r11 = r3 | r4       ALUResult = h5 = r11

//testing branches
32'h02b20263;//      beq r4,r11,36           ALUResult = 00000001
32'h00108413;//      addi r8,r1,1            ALUResult = h2 = r8
32'h00419a63;//      bne r3,r4,20            ALUResult = 00000001
32'h00308413;//      addi  r8,r1,3           ALUResult = h4 = r8
32'h0014c263;//      blt r9,r1,4             ALUResult = 00000001
32'h00408413;//      addi  r8,r1,4           ALUResult = h5 = r8
32'h00b3da63;//      bgt r7,r11,20           ALUResult = 00000001
32'h00208413;//      addi  r8,r1,2           ALUResult = h3 = r8
32'hfe5166e3;//      btlu r2, r5, -24        ALUResult = 00000001
32'h00008413;//      add  r8,r1,0            ALUResult = 1 = r8
32'hfc74fee3;//      bgeu  r9,r7,-36         ALUResult = 00000001
32'h0083e6b3;//      or  r13 = r7 | r8       ALUResult = hf = r13

//jal
32'h018005ef;//      jal x11, 24(Decimal)    ALUResult = h64 (Go to 30)

//return
32'h02a02823;//      sw  48(r0)<- r10        ALUResult = h30
32'h16802023;//      sw  352(r0)<- r8        ALUResult =h160
32'h03002603;//      lw r12 <- 48(r0)        ALUResult = h30
32'h00311733;//     sll r14, r2, r3          ALUResult = h20 = r14

//branch
32'h00c50a63;//     beq x12, x10, 20  ALUResult = 00000000  (Go to 34)

32'h0072c7b3;//     xor r15, r5, r7         ALUResult = hc = r15
32'h00235833;//     srl r16, r6, r2         ALUResult = h2 = r16
32'h4034d8b3;//     sra r17, r9, r3         ALUResult = hffffffff = r17

//JALR
32'h000586e7;//     jalr x13, 0(x11)        ALUResult = h88
```

```
//branch target
32'h01614513;//     xori r10, r2, 16h       ALUResult = h14 =     r10
32'h02e2e593;//     ori  r11, r5, 2eh       ALUResult = h2f =     r11
32'h06f37613;//     andi r12, r6, 6fh       ALUResult = h8 =      r12
32'h00349693;//     slli r13, r9, 3h        ALUResult = hfffffff0 = r13
32'h00335713;//     srli r14, r6, 3h        ALUResult = h1 =        r14
32'h4026d793;//     srai r15, r13, 2h       ALUResult = hfffffffc = r15

32'h00a8a833;//     slt   r16, r17,r10      ALUResult = h1 = r16
32'h00a8b833;//     sltu  r16, r9,r10       ALUResult = h0 = r16
32'h0028a813;//     slti  r16, r9, 2        ALUResult = h1 = r16
32'h0028b813;//     sltiu  r16, r9, 2       ALUResult = h0 = r16

32'hccccc837;//     lui r16,hccccc          ALUResult = hccccc000
32'hccccc817;//     auipc r16, hccccc       ALUResult = hccccc0b6

32'h00902a23;//     sw 20(r0)<- r9          ALUResult = h14
32'h01402103;//     lw r2<-20(r0)           ALUResult = fffffffe = r2
32'h01400183;//     lb r3<-20(r0)           ALUResult = fffffffe = r3
32'h01401203;//     lh r4<-20(r0)           ALUResult = fffffffe = r4
32'h01404283;//     lbu r5<-20(r0)          ALUResult = 000000fe = r5
32'h01405303;//     lhu r6<-20(r0)          ALUResult = 0000fffe = r6


32'h00459693;//     slli r13, r11, 4h       ALUResult = h2f0 = r13
32'h02d00423;//     sb r13->40(r0)          ALUResult = h28
32'h02802703;//     lw 40(r0) -> r14        ALUResult = 000000f0 = r14

32'h02d01423; //    sh r13 ->40(r0)         ALUResult = h28
32'h02802703;//     lw 40(r0) -> r13        ALUResult = 000002f0 = r13
```

Simulated Waveforms for All Test Instructions

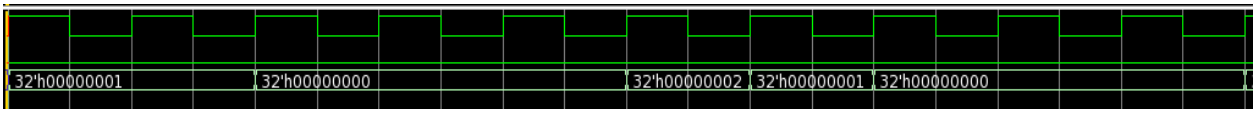
Below are the outputs of the simulated instructions. There is a four clock cycle delay before the first instruction results show. The first instruction's results are 0, so the first picture shows 5 clock cycles of WB_Data being 0.

Instructions: 1-6



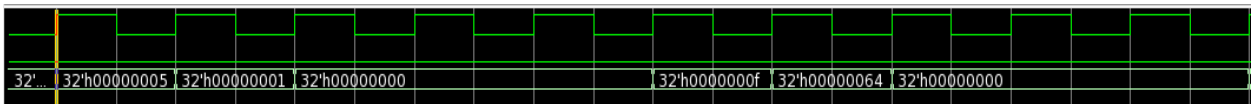
Instructions: 7-13; 13th instruction has 2 NOPs Afterwards

Instructions: 22-23,14-15; 23 and 15 have NOPs Afterwards



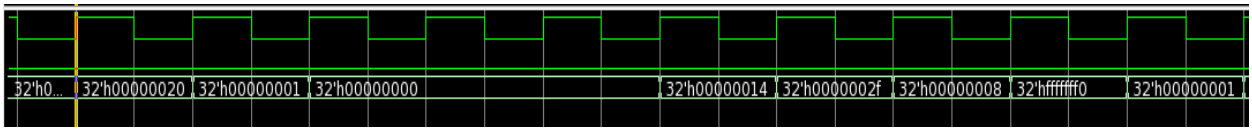
Instructions: 20-21, 16-19; 21 and 19 have 2 NOPs Afterwards

Instructions: 17-18, 24-25; 18 and 25 have 2 NOPs Afterwards



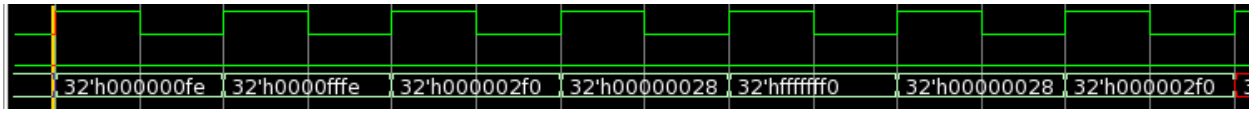
Instructions: 31–34, 26-28; 34th instruction has 2 NOPs Afterwards

Instructions: 29-30, 35-39; 30th instruction has 2 NOPs Afterwards



Instructions: 40-50

Instructions: 51-57



Synthesis Results

As a result of the synthesis, the critical clock cycle and the critical path length has drastically dropped overall. Yet, the Total area has risen a bit. Originally, the critical clock period was 1.50 ns, now it has dropped to 0.9 ns. This proves that pipelining does actually speed up the processor if implemented properly.

Critical Path Length: 0.42 ns

Critical Path Slack: 0.00 ns

Critical Clk Period: 0.90 ns

Total Area: 77411.602501

Total Power: 2.63e+04

Leakage Power: 1.56e+10