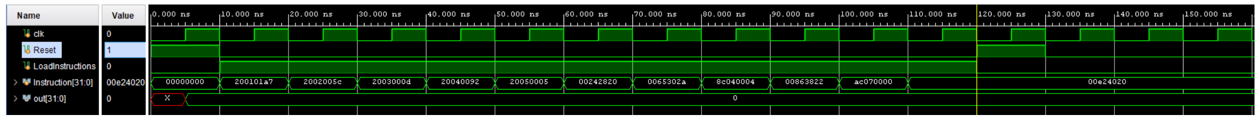
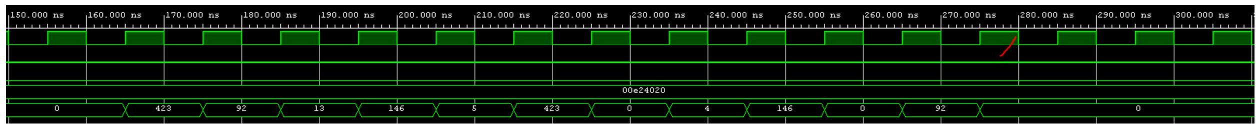
Ande Chen, Sunwoo Park
EC413
November 10, 2021

Lab 7 Report

Task 1: Pre-Lab Waveforms

The outputs of instructions whose operand registers are destination registers of instructions 1-3 positions before are incorrect. This is because it takes 3 clock cycles from when an ALU computes an output to when that output is updated in the register file for use by the next instruction. Please see prelab7waveform1.jpg and prelab7waveform2.jpg for a more detailed view.





Task 2: 1 Ahead Forwarding

To implement 1 Ahead Forwarding, we created new hardware in the ‘fowarding_ahead’ module. It takes the outputs of the Rs and Rt registers from the ID/EX stage and the output of the Rd register from the EX/MEM stage as inputs and has two 2-bit outputs ForwardA and ForwardB. By default, the two outputs are set to 0. If the EX/MEM Rd register is the same as the ID/EX Rs register, then ForwardA is set to 2. If the EX/MEM Rd register is the same as the ID/EX Rt register, then ForwardB is set to 2.

In the CPU, we instantiate the ‘fowarding_ahead’ module and connect the appropriate input and output wires. Then, we instantiate two 3-to-1 muxes with 32-bit inputs and outputs, which select the correct inputs to the ALU based on the ForwardA and ForwardB select signals from the forwarding module. ‘ALUIn1_mux’ takes IDEX_A, the output of the ID/EX RegSrcA register, and MemAluOut, the output of the EX/MEM ALUReg register, as inputs. If ForwardA is 0, the mux will output the value of IDEX_A, and if ForwardA is 2, the mux will output the value of MemAluOut. Similarly, ‘ALUIn2_mux’ takes IDEX_B, the output of the ID/EX RegSrcB register, and MemAluOut, the output of the EX/MEM ALUReg register, as inputs. If ForwardB is 0, the mux will output the value of IDEX_B, and if ForwardB is 2, the mux will output the value of MemAluOut. The outputs of the muxes are passed to the ALU through two new 32-bit wires ALUInput1 and ALUInput2.

Task 3: 2 Ahead Forwarding

For 2 Ahead Forwarding, we added MEMWB_Rd, the output of the MEM/WB DataMem register as another input to the ‘fowarding_ahead’ module. In addition, we added two more comparison statements. If EX/MEM_Rd is not the same as ID/EX_Rs and MEM/WB_Rd is the same as IDEX_Rs, then ForwardA is set to 1.If EX/MEM_Rd is not the same as ID/EX_Rt and MEM/WB_Rd is the same as IDEX_Rt, then ForwardB is set to 1. In the CPU, we added the additional input wires to the ‘forwarding_ahead,’ ‘ALUIn1_mux,’ and ‘ALUIn2_mux’ modules.

Task 4: Arbitration Logic

The logic to decide between whether to use 1 Ahead or 2 Ahead forwarding is handled by performing the AND of EX/MEM_Rd not equal to ID/EX_Rs and MEM/WB_Rd equal to IDEX_Rs. Checking for the opposite of the 1 Ahead condition ensures that the 2 Ahead forward if-statement will not be considered for a 1 Ahead situation. Furthermore, the conditions are four separate if-statements, rather than an if-else statement, so these conditions are evaluated independently of each other.

Task 5: $0 Write

We prevented forwarding for instructions attempting to write to register 0 by checking that EX/MEM Rd is not 0 in the two 1 Ahead if-statements and checking that MEM/WB Rd is not 0 in the two 2 Ahead if-statements. These conditions are ANDed with the existing conditions, so the mux selects will stay at 0 for instructions that attempt to write to register 0. To test this, we added the instructions “add R0, R1, R2” and “add R3, R0, R2” to the testbench. We verified that the forwarding logic worked as intended, because the contents of R0 stayed at 0 throughout the duration of the simulation, and the contents of R3 became 92, the contents of R2.

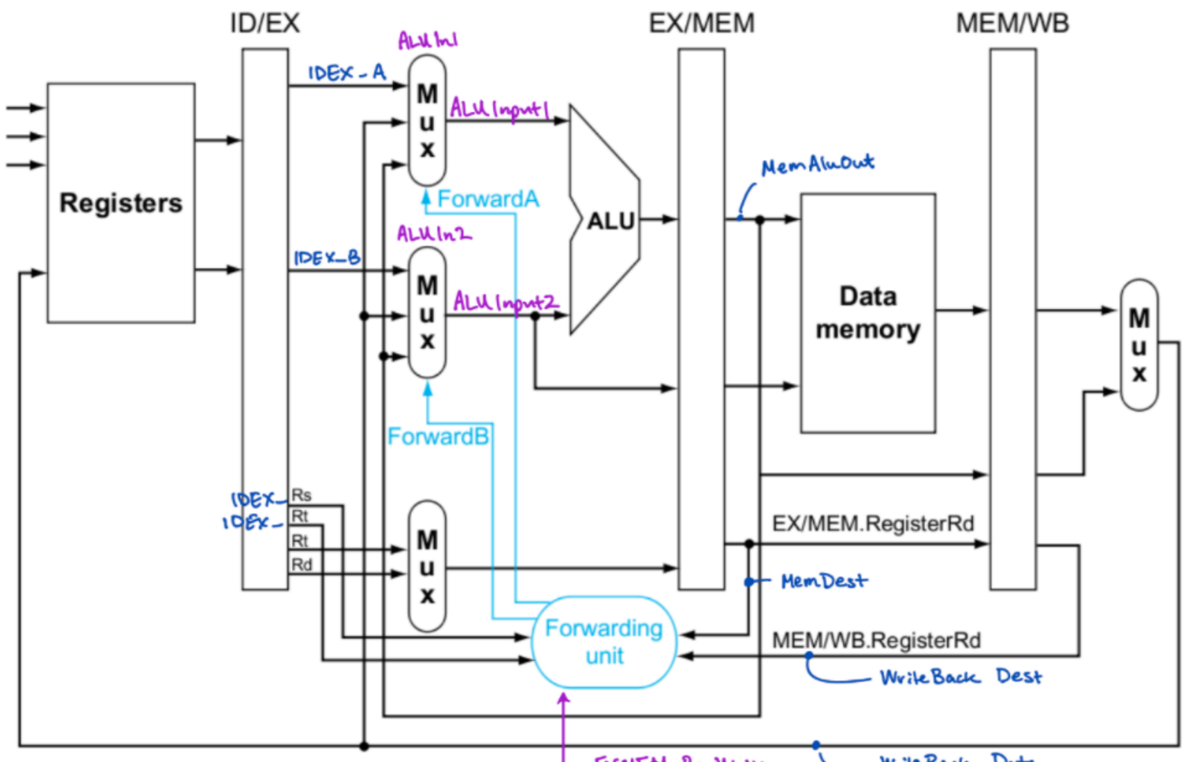
Task 6: No Write

To prevent forwarding for instructions that do not write to a register, we added the 1-bit EXMEM_RegWrite control signal as an input to the forwarding module. Then, we modified the if-statements in the ‘fowarding_ahead’ module, so that each if-statement ANDs EXMEM_RegWrite with the existing conditions. This way, when write is disabled, updated values are not forwarded to the ALU. We verified that the forwarding logic worked as intended, because the contents of R5 do not change for the “SW R5, 0(R0)” instruction.

Task 7: Check Register Bypass

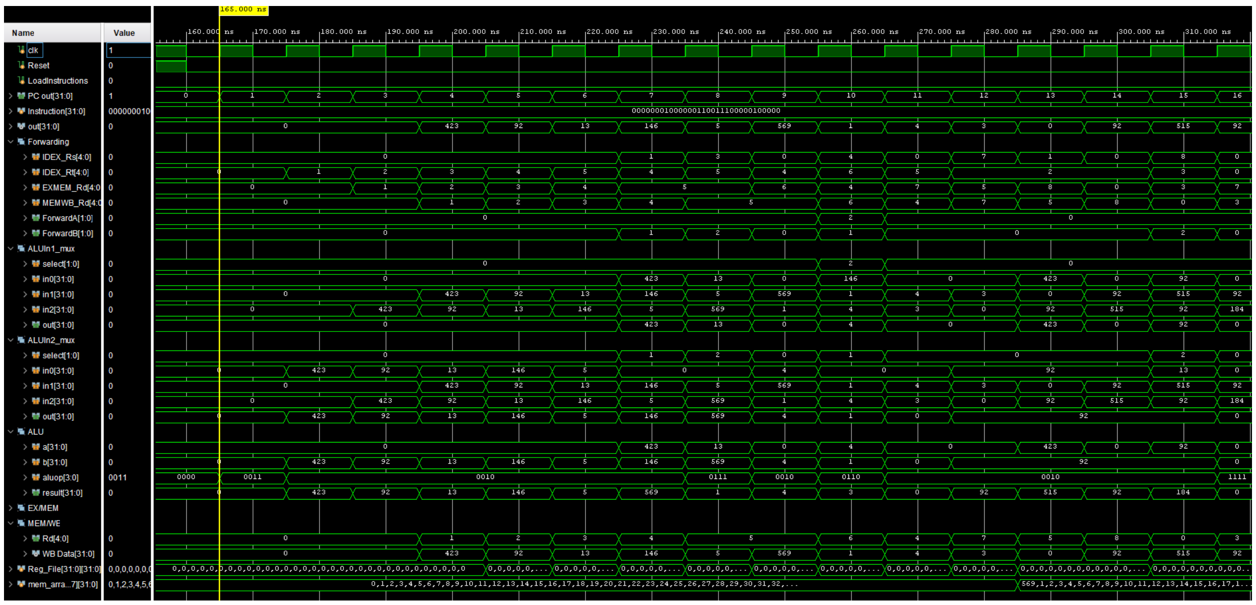
To check that register bypass works correctly, we added an “add R7, R8, R3” test cases to our testbench. An “add R8, R7, R2” instruction is three instructions before in the testbench. If the register bypass works correctly, the contents of R8 in “add R7, R8, R3” will use the value of R8 (92) computed in “add R8, R7, R2.” Therefore, “add R7, R8, R3” should be interpreted as R7 = 92 + 92, which is confirmed by the waveform, since the contents of R7 becomes 184.

Diagram of Modified Hardware



b. With forwarding

Final Waveform



Please see the lab7waveform.jpg or the included .wcfg file for more detail.