Instruction Memory

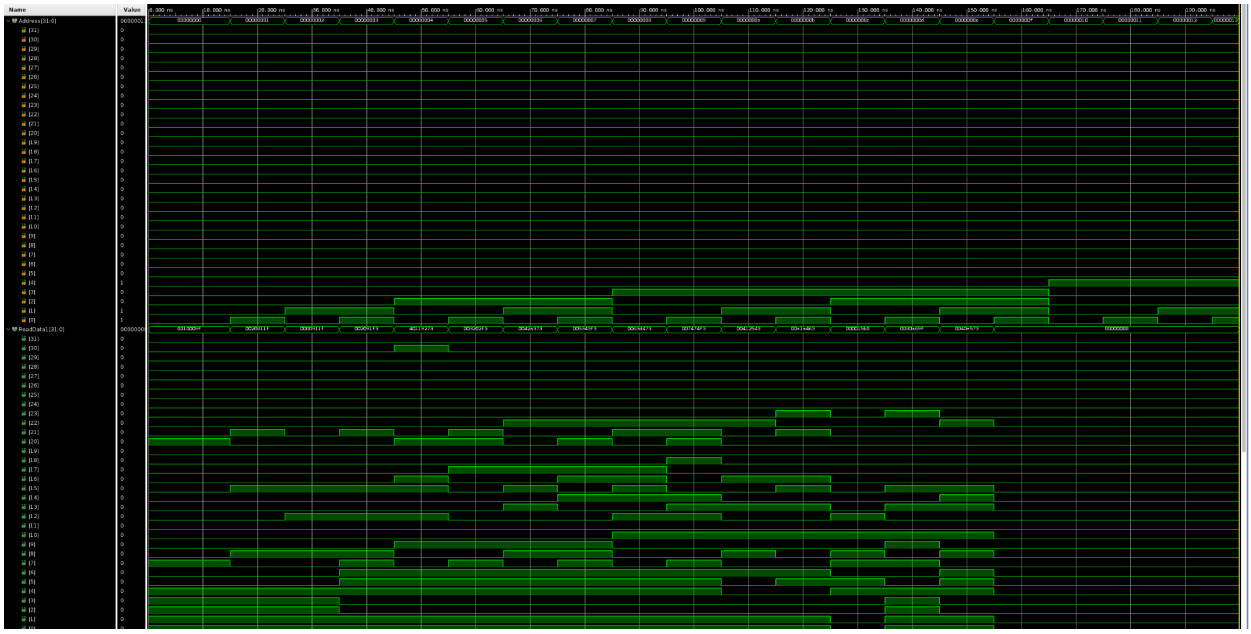

Figure 1: Instruction Memory Waveform.

In Figure 1, we can see the instruction memory waveform. The instruction memory includes all of the test scenarios necessary to execute the single-cycle data path. We can see that each of the instructions of the datapath is output through readData. The address is derived from the pc and we can see that it is counting by 1 which means it receives the signal properly from the PC. The output readData needs to be read by the control logic subsequently.

Register File

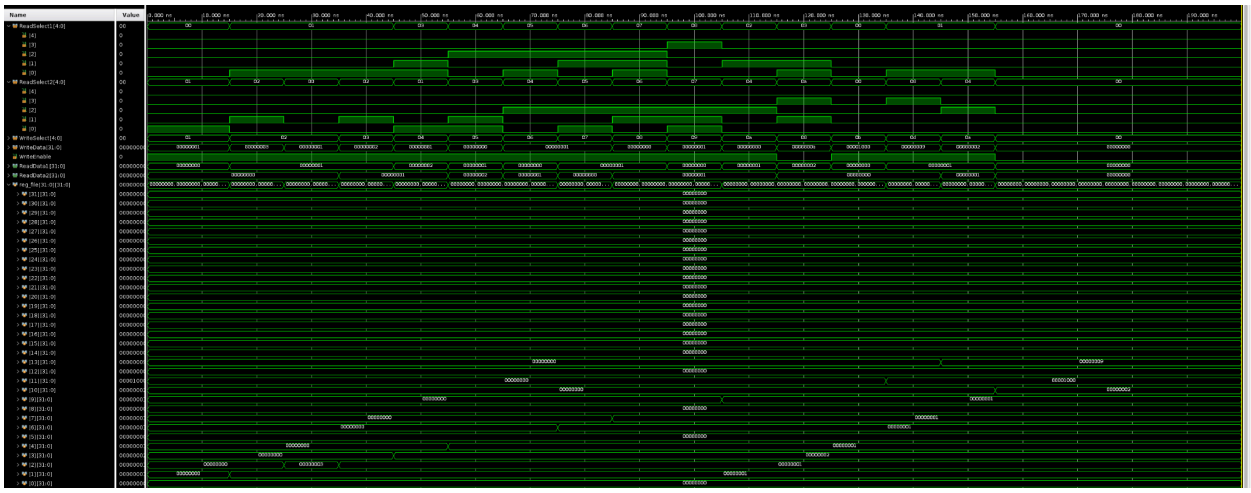

Figure 2: Register File Waveform.

The register file waveform needs to correctly store the results from each instruction in memory. To verify this functionality, we can test a range of inputs and observe whether the registers accurately store each value after execution. By reading the specific registers that require modification from the instruction memory, we can track which registers are populated with values, confirming the accuracy of our code. The output is directed to the ALU, with a MUX integrated into the register output, which also feeds into the ALU.

ALU

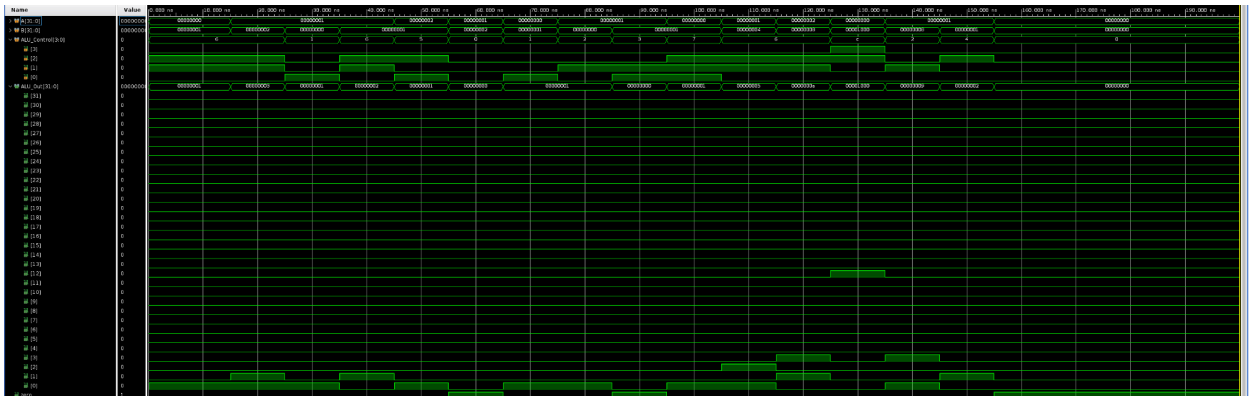

Figure 3: ALU Waveform.

The ALU opcodes, provided in the lab handout, specify how different instructions should be processed. To ensure correct functionality, we need to verify that the ALU outputs can translate accurately to addresses used by the data memory. Integrating the ALU with the datapath can allow us to properly see the ALU's arithmetic capabilities.

Control Logic

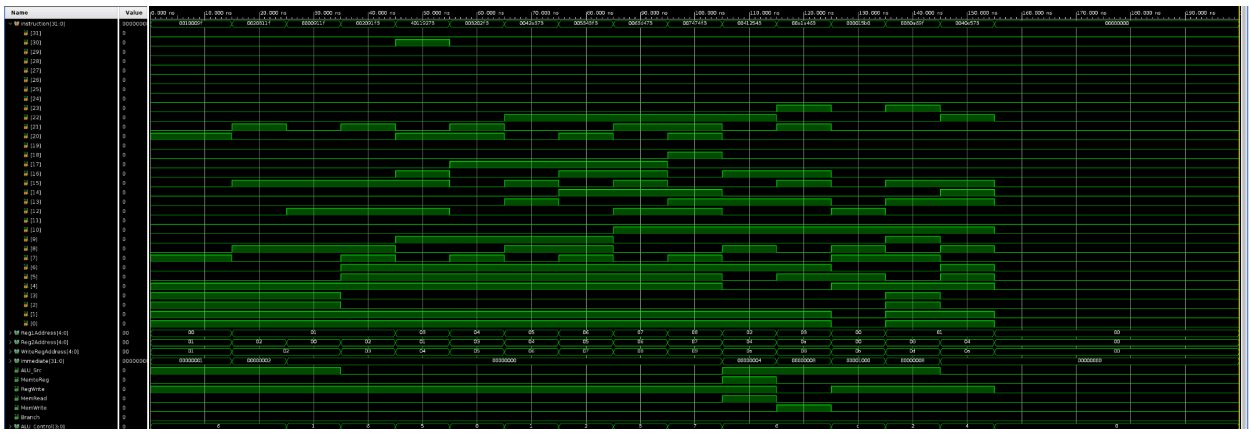

Figure 4: Control Logic Waveform

In Figure 2, we can see our control logic waveform. Our control logic combines both the ALU control and control unit to simplify code. The input we can see properly reflects the output of the

instruction memory from the previous figure. Each 32-bit instruction specifies which operation to perform, and the ALU control outputs for each instruction reflect the necessary operations.
For each instruction, an associated ALU opcode determines if additional steps are required for execution. For I-type instructions, an immediate value is included. We aim for each instruction to output its corresponding ALU control signal, opcode, and immediate. We can then test this by using instruction memory to verify if it accurately reads these outputs as inputs to the control logic.

Immediate Generator

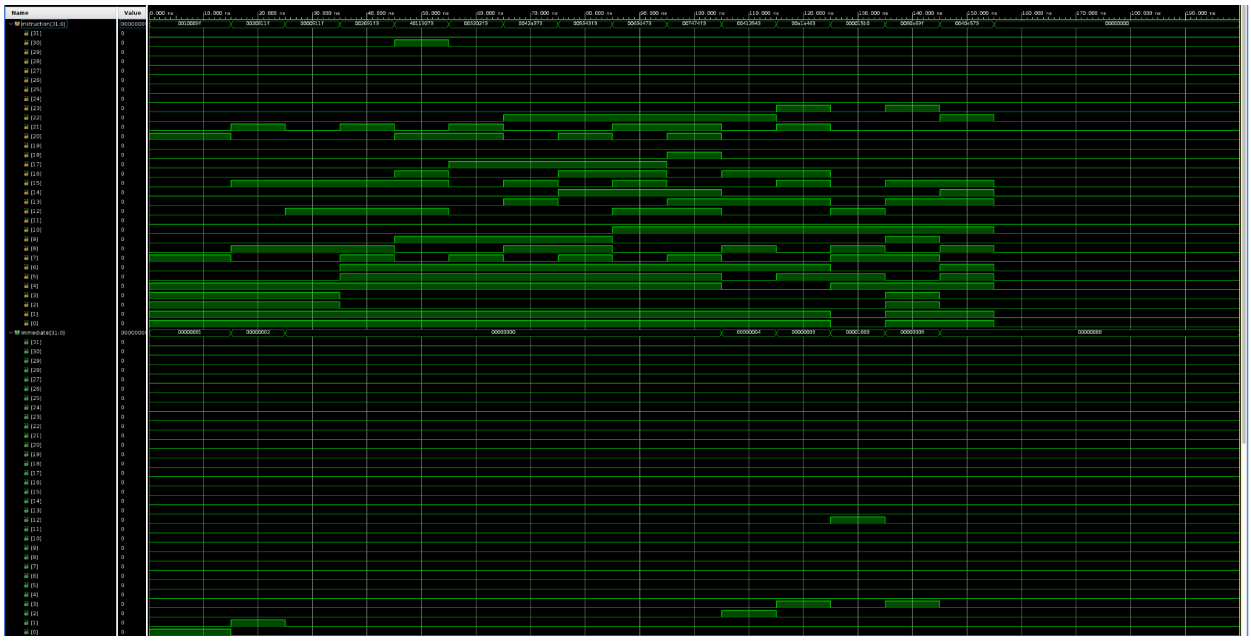

Figure 5: Immediate Generator Waveform

The immediate generator waveform also receives input from instruction memory. It interprets this input and outputs an immediate value whenever needed, which we can observe in the immediate output signal. This immediate value is then passed to the branch component.

Data Memory

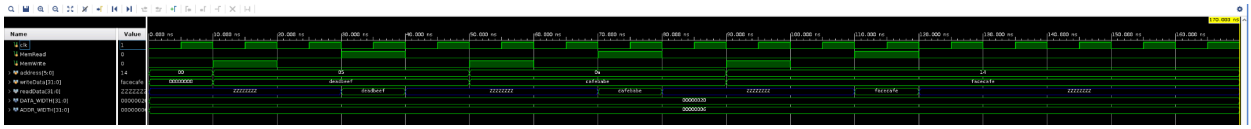

Figure 6: Data Memory Waveform

Figure 3 shows the waveform for data memory. The mux is implemented into the data memory to reduce the amount of modules needed in the datapath. Data memory operates only when a memory read is triggered. We can confirm its correct behavior: whenever MEMread is set to 1, readData runs; otherwise, it remains inactive. In the datapath, later on, we can test the functionality of the MEMread and see if it behaves as intended with the register file.

Branch

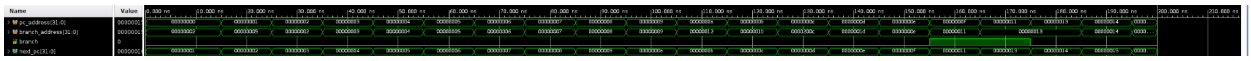

Figure 7: Branch Control Waveform

The branch we can test its viability by feeding it a beq or blt instruction if the address changes with the branch command we can confirm its effectiveness. The branch also receives zero flags to indicate if it's needed or not.

PC

Figure 8: Program Counter Waveform

The PC is simple, all it needs to do is increment by 1 and we know it properly functions. The adder for incrementing is implemented into this module for simplification. The PC also needs to be synced properly to the clock which we can see it working perfectly in the waveform. In the datapath, later on, we can further test the PC by seeing if it properly increments whenever the instruction is a branch. The address needs to then be fed to the instruction memory.

Single Cycle Data Path

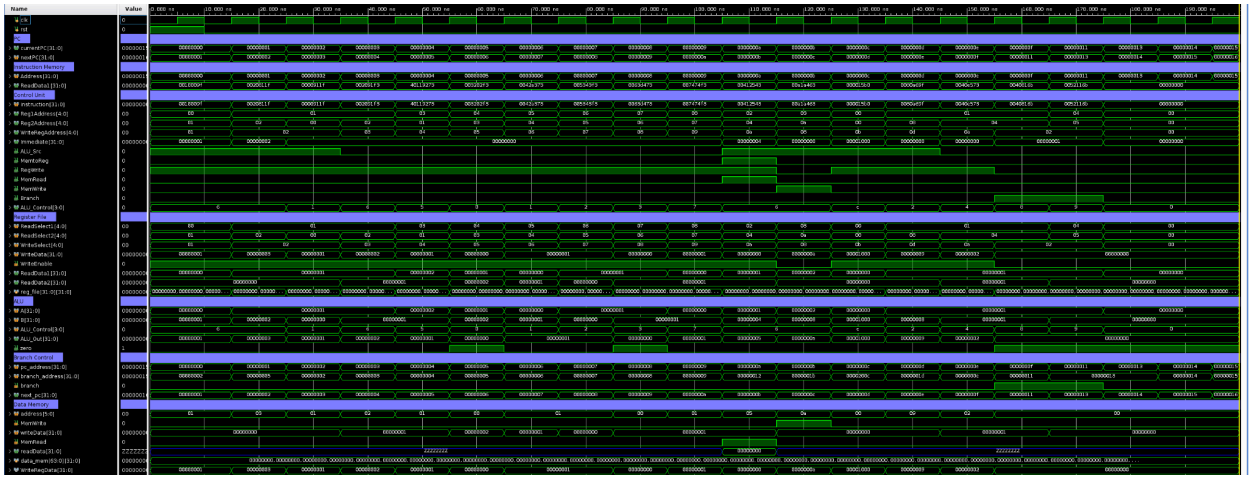

Figure 9: Single Cycle Datapath Waveform 1

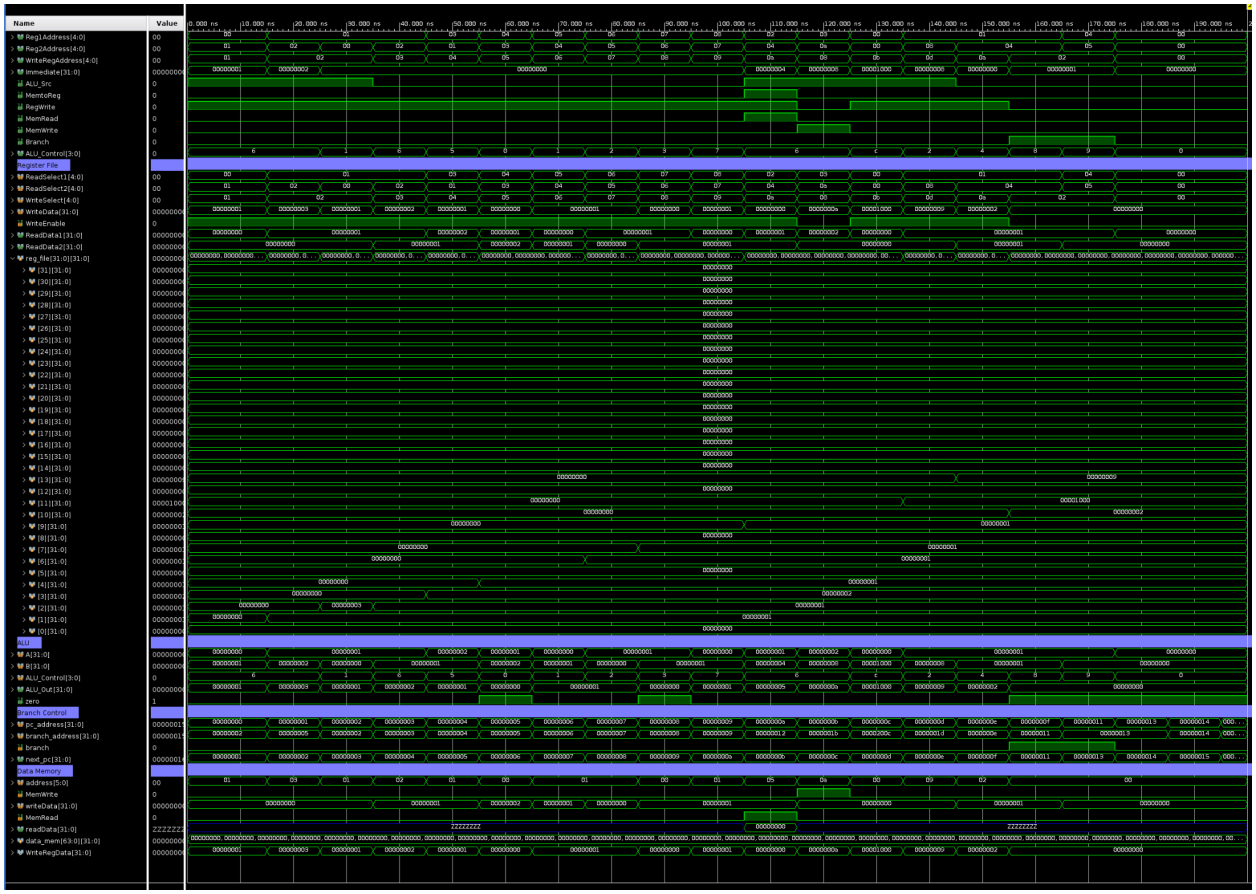

Figure 10: Single Cycle Datapath Waveform 2

The figure above shows the final single-cycle datapath waveform, demonstrating all instructions outlined in the lab handout. The top module's only inputs are clk and rst, where the datapath completes a single cycle with each clock tick, and the positive clock edge triggers the execution of a new instruction. At the start, we initialize the module with rst set to 1.

In this program, all registers are initialized to 0. We began with the instructions addi x1, x0, 1 and addi x2, x1, 2. After the first cycle, x1 is set to 1, and after the second cycle, x2 becomes 3. Next, we execute ori x2, x1, 0, which results in x2 holding the value 1. Then, the instruction add x3, x1, x2 is executed, making x3 equal to 2. Following that, sub x4, x3, x1 sets x4 to 1 in the register, and and x5, x4, x3 results in x5 storing the value 0. The subsequent or x6, x5, x4 instruction sets x6 to 1, and xor x7, x6, x5 makes x7 hold the value 1. We then ran slt x9, x8, x7, resulting in x9 being set to 1 since x8 is 0 and x7 is 1.

The following instructions involve data memory, starting with lw x10, 4(x2). At 105 ns, we see an ALU output address of 5, which is correct as x2 holds 1, causing data memory to load a 0 into x10 (since data memory at that address is 0). The next instruction, sw x10, 8(x3), stores 0 at address 10 in data memory. Afterward, lui x11, 0x(1) loads a 1 followed by 12 zeros, which we see in the register file as 00001000 in hexadecimal. Then, xori x13, x1, 8 stores 9 in x13. The

instruction sll x10, x1, x4 results in x10 holding 2 in hexadecimal as it performs a left logical shift on binary 1 by one position.

The final two instructions use branch control. First, beq x1, x4, 1 increments the program counter (PC) by 2, as x1 and x4 are equal, causing the PC to jump from 15 to 17. The last instruction, blt x5, x4, 1, at PC address 17, is executed, and since x5 is less than x4, the PC increments again to reflect this branch.

With these tests complete, we are confident in the accuracy of our final single-cycle datapath.