446 EXP3 – Report

1.2.1: Datapath Design

Question 1:

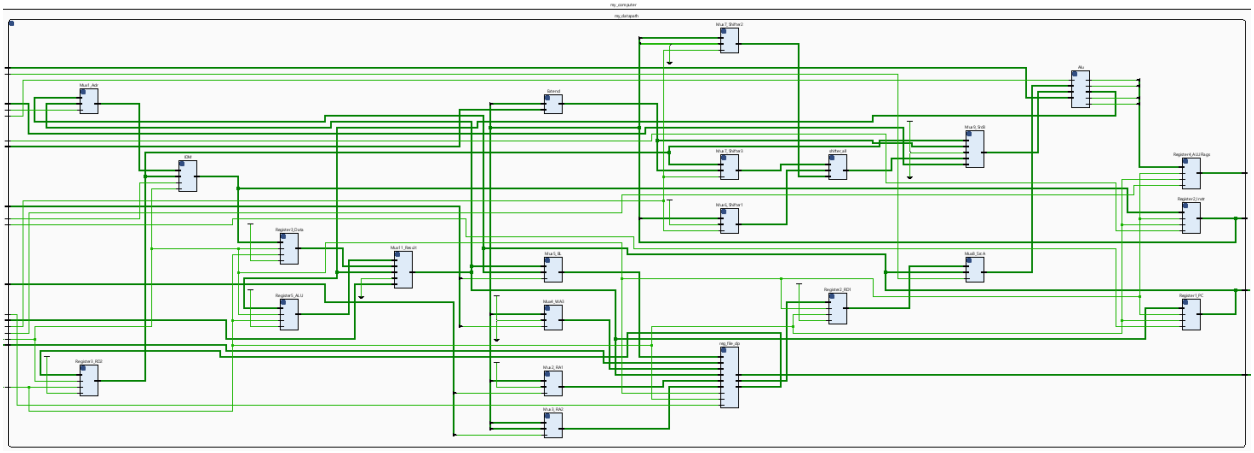

Figure 1: Datapath's RTL Schematic

Question 2:

1. **Register-Shifted Immediate Operations**
   The design introduces a dedicated shifter stage in the datapath, selected by the Shift_ctrl signal. When Shift_ctrl is low, the second ALU operand (in WriteData) may be shifted by the bits in Instr[11:7] if needed; when Shift_ctrl is high, the operand is taken from the extended immediate (ExtImm) and rotated according to {Instr[11:8], 1'b0}. Three multiplexers (Mux6_Shifter1, Mux7_Shifter2, and Mux7_Shifter3) choose which signals drive the shifter's control, shift amount, and input. The shifter module then outputs the appropriately shifted or rotated result to be used by the ALU in the execute stage.

2. **MOV Operations**
   MOV instructions are supported by allowing the ALU to pass the operand through without performing any arithmetic logic. In the ALU, when ALUControl is set to the MOV code (4'b1101), the ALU effectively copies the second operand to its output. This means that the datapath can load a register directly from either another register (if Shift_ctrl is low and WriteData is chosen) or an immediate value (if Shift_ctrl is high and ExtImm is chosen), giving the standard "move" behavior.

3. **BL (Branch with Link)**
   The BL mechanism uses two bits of control (BL_ctrl) to redirect where the write-back address and data come from. The first bit of BL_ctrl selects the link destination register (e.g., R14 for ARM-style link). The second bit of BL_ctrl chooses whether to write back the current PC or the typical ALU result. By combining these bits, the design can store the link value (PC+4 or PC+8, depending on the pipeline offset) into R14, allowing subroutine calls to return properly.

4. **BX (Branch and Exchange)**
   BX is detected in the execute stage when the instruction's function bits (Instr[27:4]) match a specific pattern (24'b000100101111111111110001). If so, the datapath sets the ALU to the MOV operation with ALUSrcB = 2'b00, which selects Rm as the operand. The PCWrite signal is then asserted so that the PC is updated with the contents of Rm. This causes an immediate branch to the address in Rm, implementing the BX instruction without extra hardware beyond a simple control check and the existing ALU pass-through mode.

These extensions allow the datapath to handle shifts or rotations of immediate values, use MOV for loading register contents directly, and perform BL/BX instructions using the same hardware resources (ALU, shifter, register file) with small additions to the control logic and multiplexing.

1.2.2: Controller Design

Question 1:

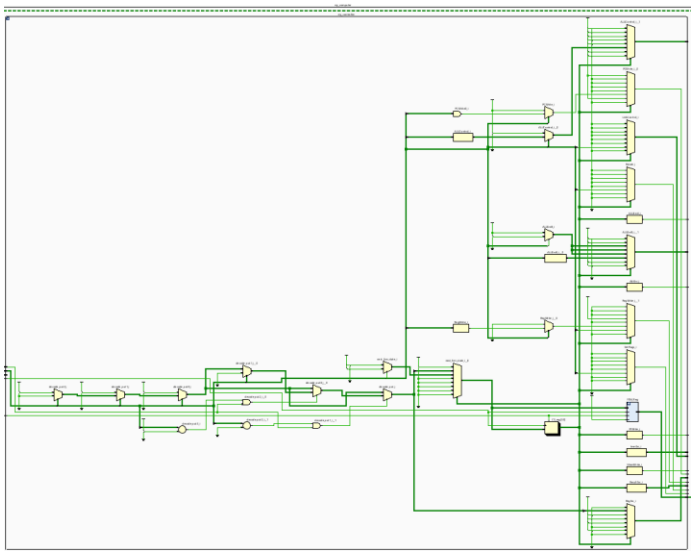

Figure 2: Controller's General RTL Schematic

Question 2:

1. **Register-Shifted Immediate Operations**
   The Controller monitors the instruction's "I" bit (Instr[25]) and uses it to determine whether to enable the shifting of an immediate value. When this bit is set, the Controller asserts Shift_ctrl high in the execute path, causing the datapath's multiplexers to select the extended and rotated immediate rather than a register value. This logic is mostly in the Execute states (e.g., ExecuteR) where Shift_ctrl is driven by Instr[25].

2. **MOV Operations**
   MOV instructions are identified by decoding the instruction's opcode and function fields inside the Controller (for instance, when Instr[24:21] == 4'b1101 for a standard MOV). The Controller then sets ALUControl to the MOV code, which in turn instructs the ALU (in the datapath) to pass the operand directly. The Controller also sets or clears RegWrite depending on whether the MOV needs to write a register (typical move) or write the program counter (MOV to PC).

3. **BL (Branch with Link)**
   When a BL instruction is detected (by checking bits in the instruction such as Instr[24] for link), the Controller asserts PCWrite to branch and also sets RegWrite so that the link register (R14) can be updated with the appropriate return address. The Controller drives the two-bit signal BL_ctrl to redirect where the link is written (e.g., to R14) and what data is written (e.g., PC+4 or PC+8). In the "Branch" or "BL" state, the Controller also selects the next FSM state or returns to Fetch.

4. **BX (Branch and Exchange)**
   BX is detected inside the Controller by checking whether certain instruction fields match the BX opcode pattern (e.g., Instr[27:4] == 24'b000100101111111111110001). Once recognized, the Controller sets ALUControl to a MOV operation and uses ALUSrcB = 2'b00 so that Rm is the operand. It then asserts PCWrite so that the datapath updates the PC with the contents of Rm (branch). This sequence is implemented in the ExecuteR or Executel path (whichever state is chosen for a data-processing instruction) by overriding normal ALU usage with the BX logic.

By integrating these checks and generating the correct control signals (Shift_ctrl, BL_ctrl, etc.), the Controller ensures that the datapath properly handles shifted immediates, MOV instructions, and branching with link or exchange.

1.2.4: Testbench

It is important to note that I have edited *Helper_Student.py* to change the cycle amount for each operation. In my design, DP operations take 3 cycles, Memory load operations take 4 cycles. Memory store and branch are the same with the lecture notes.

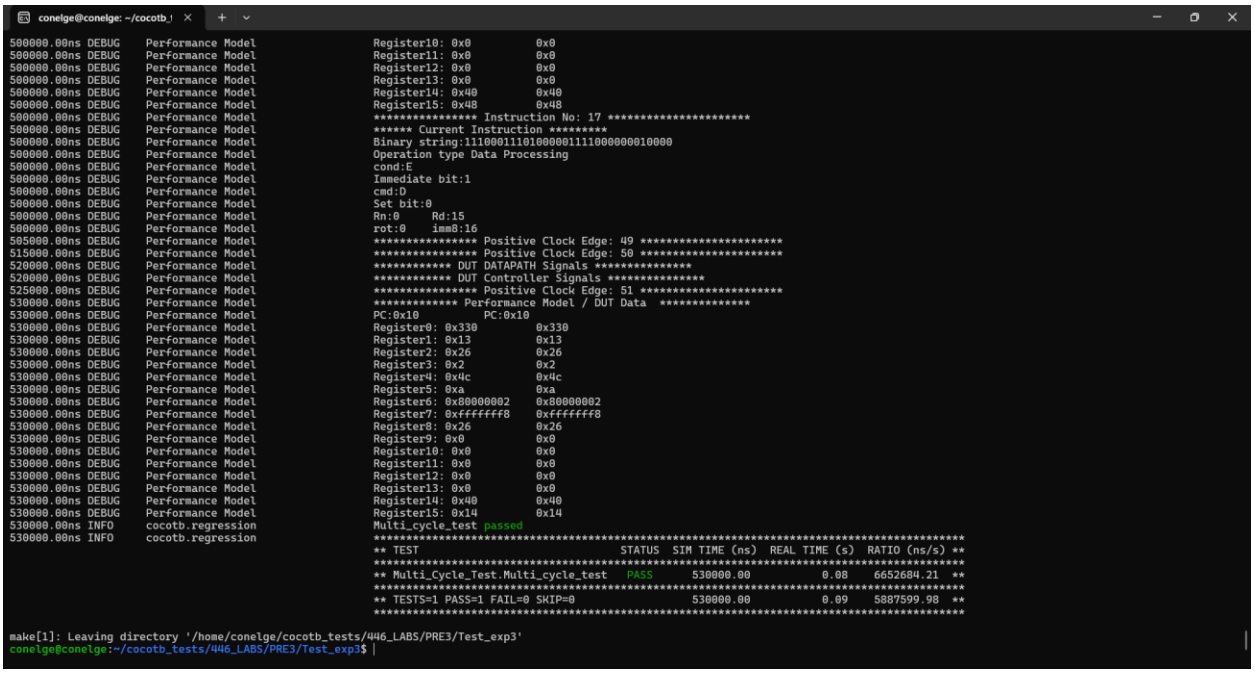

Figure 3: Test Bench Results