EE446 – Experiment4
Çetincan Önelge - 2375426

446 EXP4 – Report

1.2.1: Datapath Design

Question 1:

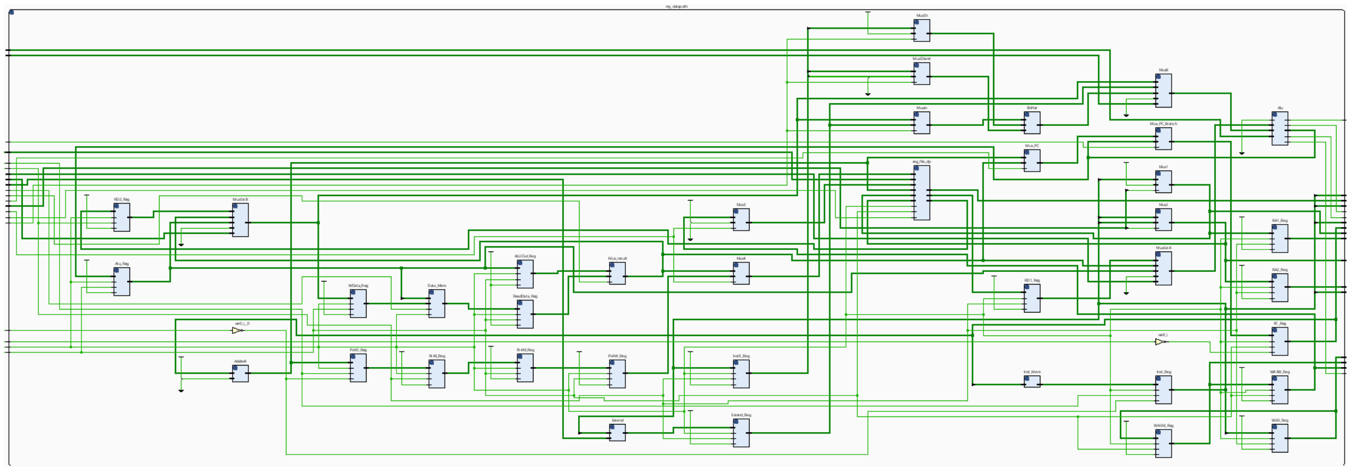


Figure 1: Datapath's RTL Schematic

Question 2:

1- Register-shifted-immediate operations

- I instantiated the same combinational **shifter** module used in data-processing instructions and also wired up a pair of 2-to-1 muxes ahead of it. One mux chooses between the raw register operand (Rm) and the zero-extended immediate field, while the other mux selects either the 5-bit shift amount from bits [11:7] or the rotate-imm value (bits [11:8] plus a zero).
- A third mux then picks the final shift control (LSL/LSR/ASR/ROR) either from the instruction's shift-type bits or, in the case of rotate-immediate, forces it to "ROR."
- In the ALU stage, my 4-to-1 ALU-B mux uses this shifter output as one of its inputs, thus seamlessly handling any register-shifted-register or register-rotate-immediate patterns (e.g. ADD R1,R2,R3,LSL#4, AND R4,R5,#0xFF00 ror #8, etc.).

2- **MOV (register and rotate-immediate)**

- I treat both variants as ALU "pass-through" operations. In the controller design drives ALUControl = MOV (4'b1101) and assert RegWrite.
- For *register* moves (MOV Rd,Rm), the design clears the shifter_control signal so that the ALU directly sees Rm.

- For *rotate-immediate* moves (MOV Rd,#imm8), the design asserts shifter_control so that the shifter consumes the 8-bit immediate (rotated by the rot-imm field) and presents it to the ALU, then write that result into Rd.

3- BL (branch-and-link)

- In the Decode stage I added a third bit in the design's RegSrc[2] control. When Op=10 (branch family) and the link-bit is set, the design:
    1. Drive BranchD = 1 so that the EX stage computes the new PC.
    2. Drive RegWriteD = 1 and set RegSrc[2]=1, which in Write-Back flips design's RF write-address mux to target R14 instead of the normal Rd.
    3. Route the value PC+4 into the RF write-data path so that R14←PC+4 at WB.
- Meanwhile the normal PC update path uses the same EX-stage branch adder and BranchTakenE signal.

4- BX (branch-and-exchange)

- The design detects the unique encoding {Funct=01001, Rd=1111} in the controller (when Op=00 but the full [Funct,Rd] pattern matches).
- On a BX, the design sets BranchD=1, leave PCSrcD=0, and drive ALUControl=MOV with ALUSrc=0 so that the ALU simply passes through Rm. The EX stage then asserts BranchTakenE into the PC mux, causing PC←Rm.
- No register write occurs (we clear RegWriteD), so BX works purely by forwarding the register value into the PC without touching the RF.

1.2.2: Controller Design

Question 1:

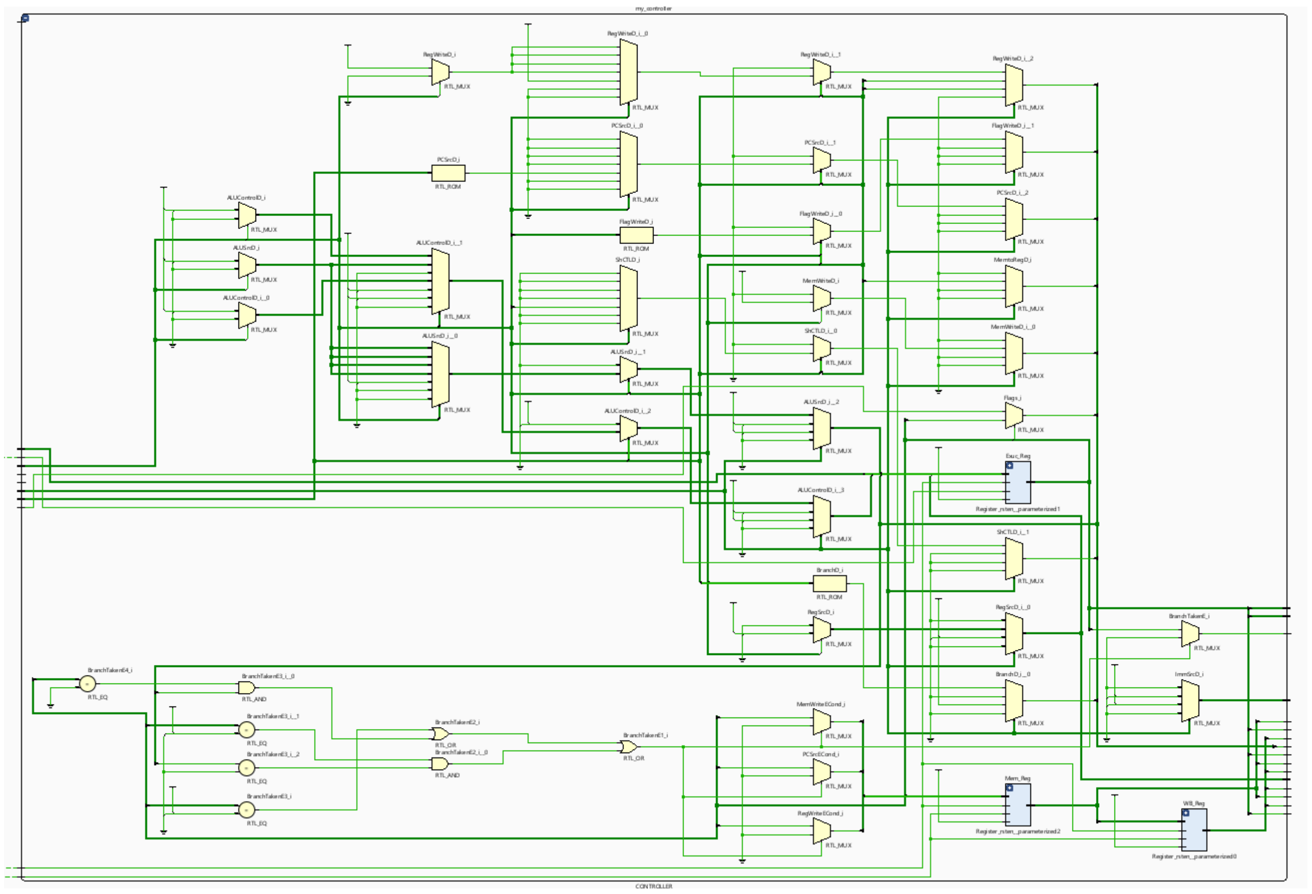


Figure 2: Controller's General RTL Schematic

Question 2:

1. **Register-shifted-immediate**
    - I introduced a new shifter_control bit in the ID stage control bundle. In the design's combinational decode logic, whenever it sees a data-processing instruction whose shift/immediate bit is set *and* the opcode is not MOV/CMP, the design asserts shifter_control=1. That bit is carried forward to EX, gating the shifter input and shift-amount muxes.
2. **MOV (both types)**
    - In the case(Op=00) branch of design's main decode always-block, I added a case arm for MOV (opcode 1101).
    - If Rd==15, I treat it as an immediate branch (same as BX of immediate), so design asserts PCSrcD=1 and drive shifter_control based on the I-bit (whether to use Rm or rot-imm).
    - Otherwise, I set ALUControlD= MOV, RegWriteD=1, MemtoRegD=0, and choose ALUSrcD=1 so that the second operand comes from the extender →

shifter chain for rotate-immediate MOV, or from the register path for plain MOV.

3. **BL**
   - Under Op=10 (branch family) with Funct[4]=1, design detects BL. Design asserts
     - BranchD=1 so that EX emits a branch target,
     - RegWriteD=1 so that WB will write into R14,
     - RegSrcD=01 (to route Rd→R14), and
     - ImmSrcD=10 (to extend the 24-bit branch offset).
4. **BX**
   - Finally, within Op=00 data processing, design sniffs the full 10-bit pattern {Funct[5:0],Rd} for the BX encoding.
   - For that pattern, design clears all other writes, set BranchD=1, leave PCSrcD=0 (EX's own branch logic handles it), and ALUControlD=MOV with ALUSrcD=0 to forward Rm to the PC.

All these new control bits (shifter_control, RegSrc[2], BranchD, etc.) flow through our three pipeline control registers (ID→EX, EX→MEM, MEM→WB) so that the correct signals reach each stage at the right time.

1.2.3: Hazard Unit

Question 1:

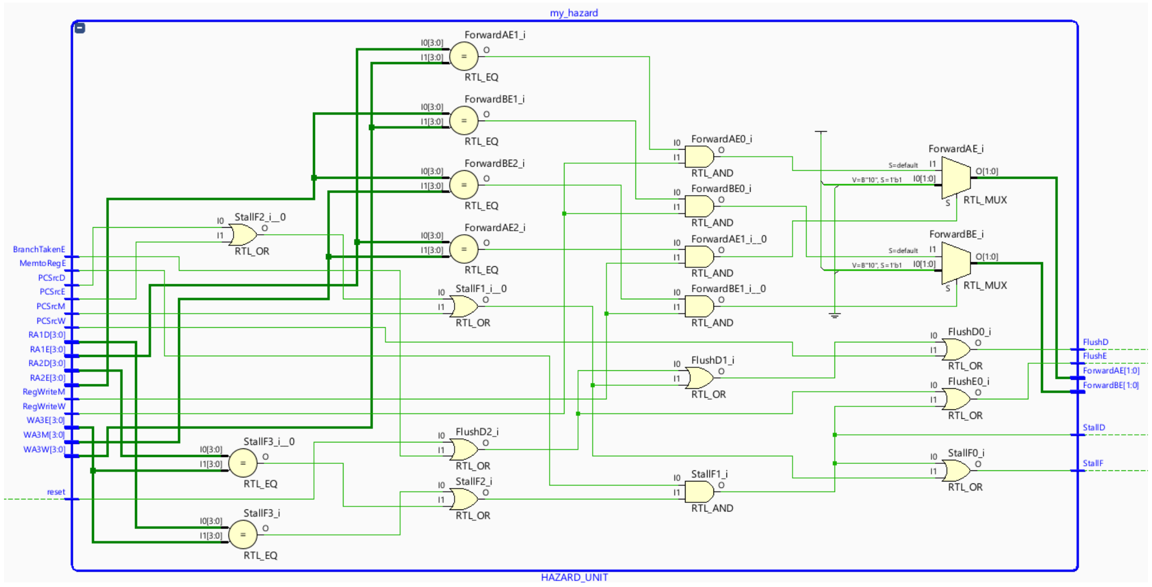


Figure 3: Hazard Unit's General RTL Schematic

Question 2:

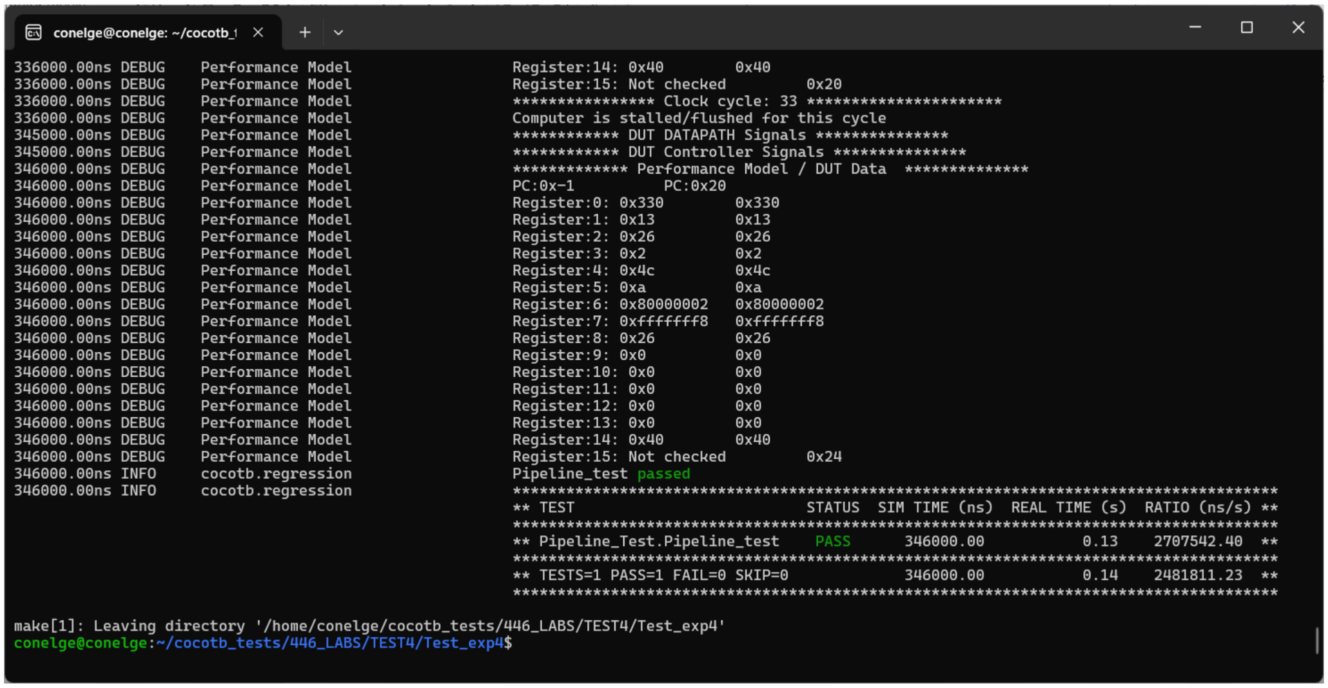
Hazard-detector already handled raw register-number matches and stalling; because the new operations (shifts, MOV, BL, BX) still use ordinary register operands and destinations, **no special extra logic** was required beyond:

1. **Data-hazard forwarding**
    - Design compares EX-stage source addresses (RA1E, RA2E) against both MEM/WB destinations (WA3M, WA3W), and forward as before. This automatically covers if the ALU output was the result of a MOV or any shift—forwarding doesn't care which ALU opcode produced it.
2. **Load-use stall**
    - The classic test (MemtoRegE && ((RA1D==WA3E)||(RA2D==WA3E))) still applies equally when the pending EX-stage write is a rotate-immediate MOV or part of a BL link; if the very next instruction needs that value, computer stalls a cycle.
3. **Control-hazard flushes**
    - BL and BX both drive BranchTakenE=1 in EX, so design automatically flushes IF/ID and ID/EX via (BranchTakenE || ...) conditions.
    - I also watch PCSrcD and PCSrcM (for early branch resolution in ID/MEM if applicable) and PCSrcW to catch any link writes that might reroute the PC.

In short, by reusing the same matching, stall, and flush conditions keyed only on register numbers and branch signals, design seamlessly supports the **shifted-immediate**, **MOV**, **BL**, and **BX** variants without adding any ad-hoc cases in the hazard logic itself.

1.2.4: Testbench

Testbench does not have any modifications, it is used as it is.



```
336000.00ns DEBUG    Performance Model                   Register:14: 0x40        0x40
336000.00ns DEBUG    Performance Model                   Register:15: Not checked         0x20
336000.00ns DEBUG    Performance Model                   **************** Clock cycle: 33 **********************
336000.00ns DEBUG    Performance Model                   Computer is stalled/flushed for this cycle
345000.00ns DEBUG    Performance Model                   ************ DUT DATAPATH Signals ***************
345000.00ns DEBUG    Performance Model                   ************ DUT Controller Signals ***************
346000.00ns DEBUG    Performance Model                   ************* Performance Model / DUT Data  **************
346000.00ns DEBUG    Performance Model                   PC:0x-1          PC:0x20
346000.00ns DEBUG    Performance Model                   Register:0: 0x330        0x330
346000.00ns DEBUG    Performance Model                   Register:1: 0x13         0x13
346000.00ns DEBUG    Performance Model                   Register:2: 0x26         0x26
346000.00ns DEBUG    Performance Model                   Register:3: 0x2          0x2
346000.00ns DEBUG    Performance Model                   Register:4: 0x4c         0x4c
346000.00ns DEBUG    Performance Model                   Register:5: 0xa          0xa
346000.00ns DEBUG    Performance Model                   Register:6: 0x80000002   0x80000002
346000.00ns DEBUG    Performance Model                   Register:7: 0xfffffff8   0xfffffff8
346000.00ns DEBUG    Performance Model                   Register:8: 0x26         0x26
346000.00ns DEBUG    Performance Model                   Register:9: 0x0          0x0
346000.00ns DEBUG    Performance Model                   Register:10: 0x0         0x0
346000.00ns DEBUG    Performance Model                   Register:11: 0x0         0x0
346000.00ns DEBUG    Performance Model                   Register:12: 0x0         0x0
346000.00ns DEBUG    Performance Model                   Register:13: 0x0         0x0
346000.00ns DEBUG    Performance Model                   Register:14: 0x40        0x40
346000.00ns DEBUG    Performance Model                   Register:15: Not checked         0x24
346000.00ns INFO     cocotb.regression                   Pipeline_test passed
346000.00ns INFO     cocotb.regression                   ***************************************************************************************
                                                         ** TEST                              STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                         ***************************************************************************************
                                                         ** Pipeline_Test.Pipeline_test        PASS       346000.00           0.13    2707542.40  **
                                                         ***************************************************************************************
                                                         ** TESTS=1 PASS=1 FAIL=0 SKIP=0                  346000.00           0.14    2481811.23  **
                                                         ***************************************************************************************

make[1]: Leaving directory '/home/conelge/cocotb_tests/446_LABS/TEST4/Test_exp4'
conelge@conelge:~/cocotb_tests/446_LABS/TEST4/Test_exp4$
```

Figure 4: Test Bench Results