Middle East Technical University
Electrical & Electronics Engineering Department

EE446 Computer Architecture II - Experiment 2

Preliminary Work Report

Necati Teoman Bahar 2515583

March 23, 2025

Introduction

In this report, the development process of a Single-Cycle ARM Processor that is capable of executing a limited amount of ARM instructions will be inspected and documented. The processor's building blocks, newly defined instruction-based paths and control signals will be talked about. The design process will be done using Verilog HDL in Vivado Design Suite environment. The process is mainly divided into 2 major design blocks, **datapath design and controller design**. In the upcoming sections, each block will be explained in further detail. Finally the testbench results will be shared and analyzed.

Datapath Design

The design process stared with the integration of smaller modules given to us by the teaching assistants. As the starting point, the proposed datapath in our EE446 lectures, given in Figure **??**, is selected. The datapath is implemented as is at the beginning, after the implementation, new paths are created by adding MUXs and appropriate modules when they are needed for an instruction execution.

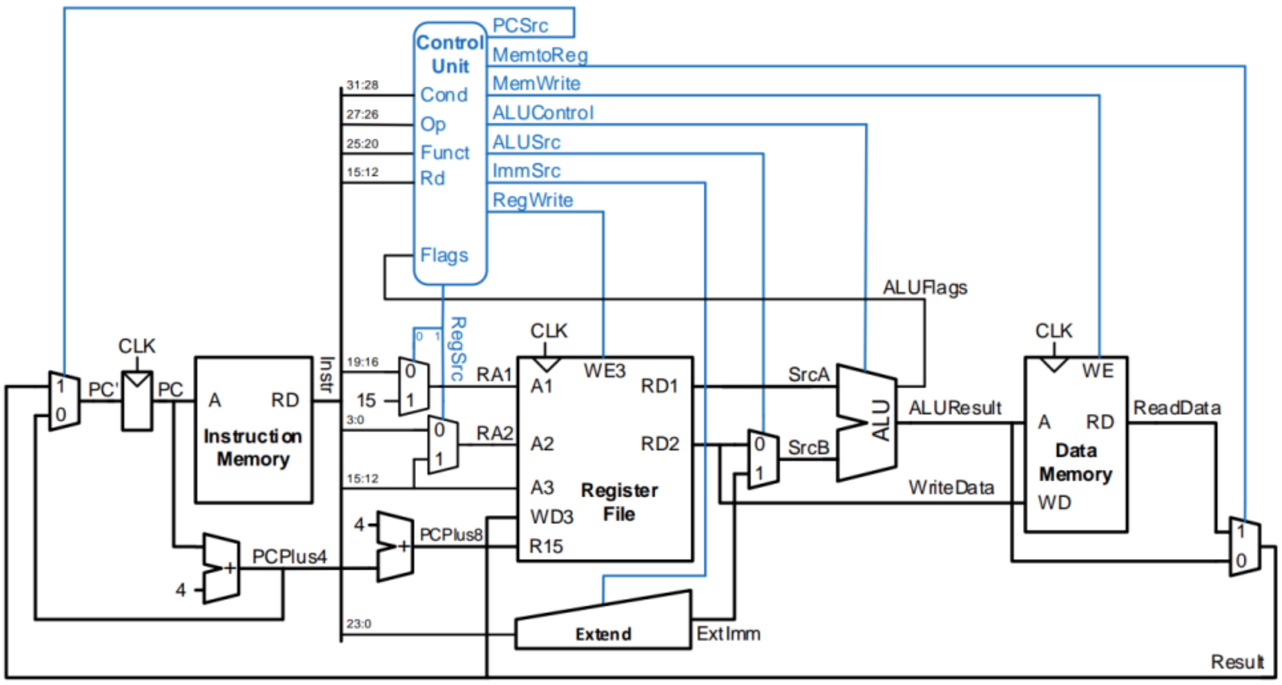


Figure 1: Proposed Datapath

The finalized datapath RTL view can be seen in Figure 2. The design is done using gate-level approach which only utilizes wires and module instantiations.

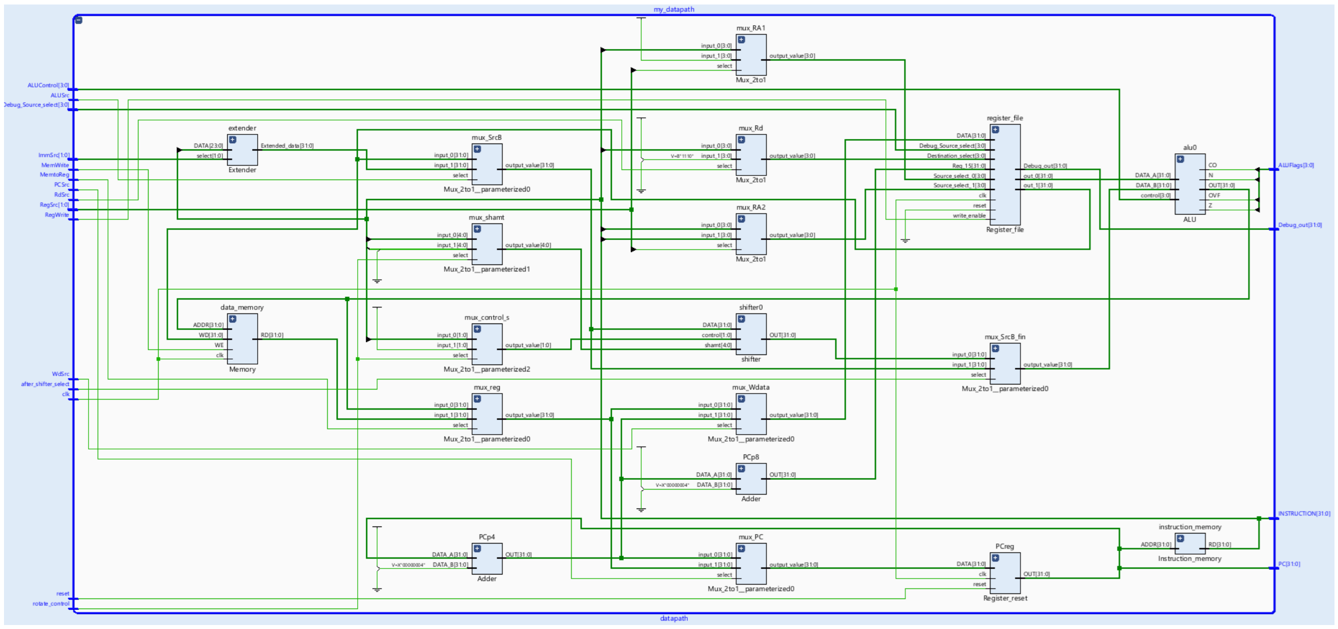


Figure 2: RTL view of the Finalized Datapath

MOV Instruction and Shifter

For the instructions with shifting, a shifter is added at the second readout of the register file. However, this addition was insufficient for **MOV** instruction with **imm8**. As a solution, shifter is moved to between the MUX for the SrcB selection and the input of the ALU. The shifter's shamt and control inputs are received from their own MUXs. These MUXs select inputs from the instruction directly or the concated inputs for the **MOV** instruction. Also the output of the shifter is also MUXed with its input. This MUX allows the selection of shifted or the not-shifted value to enter the ALU.

Branch Instructions

The simple branch instruction is already within the capability of the initial datapath. But the **BL** instruction requires a constant binary in the **"Write Destination"**. To achieve this functionality, a MUX is added. This MUX gives constant *"4'b1110"* or the R_d part of the instruction. Also the **"Write Data"** input is MUXed with the ALUResult and the "PCPlus4" to achieve proper BL instruction.

For the **BX** instruction, the final datapath is sufficient. The proper control signal generation achieves the function.

Controller Design

Similar to the datapath, the proposed controller architecture in our EE446 Lectures are selected as a base to the design. Without using a decoder, cases like **Data-Processing, Branch and Memory** instructions are determined with the AND gates. The created signals are then used with condition check signals to create final control signals that enters the datapath.

The control signals for the shifter is created by finding the needs for each instruction then using AND gates for the creation of a single signal. These signals are then re-used for the ALUControl assignments. The 4-bit ALUControl signal is mainly the **cmd** part of the instruction and it is activated with an **ALUop** signal. In cases like MOV and CMP instructions, the ALUControl is selected as Move or SUB function with the use of if-else statements in the Verilog HDL. For the special Branch instructions, proper signals are created in the same fashion as MemWrite or RegWrite signal generation.

In addition, the design is able to handle all conditions. The RTL view of the finalized controller design can be seen in Figure 3.

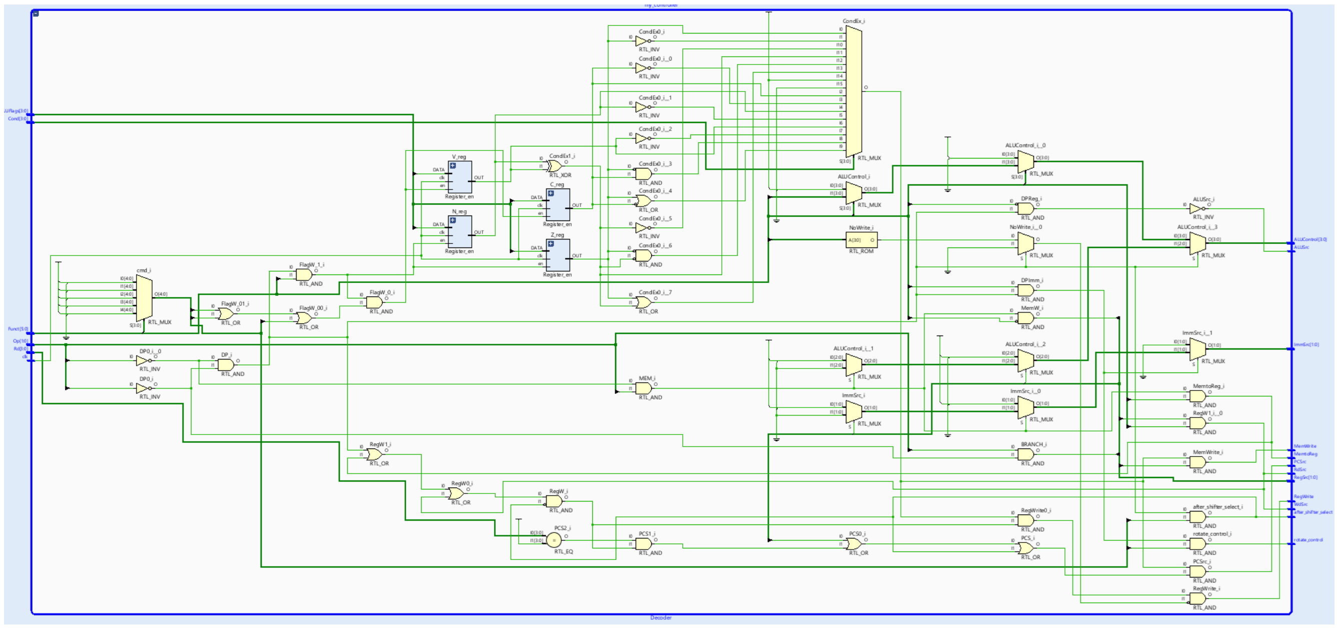
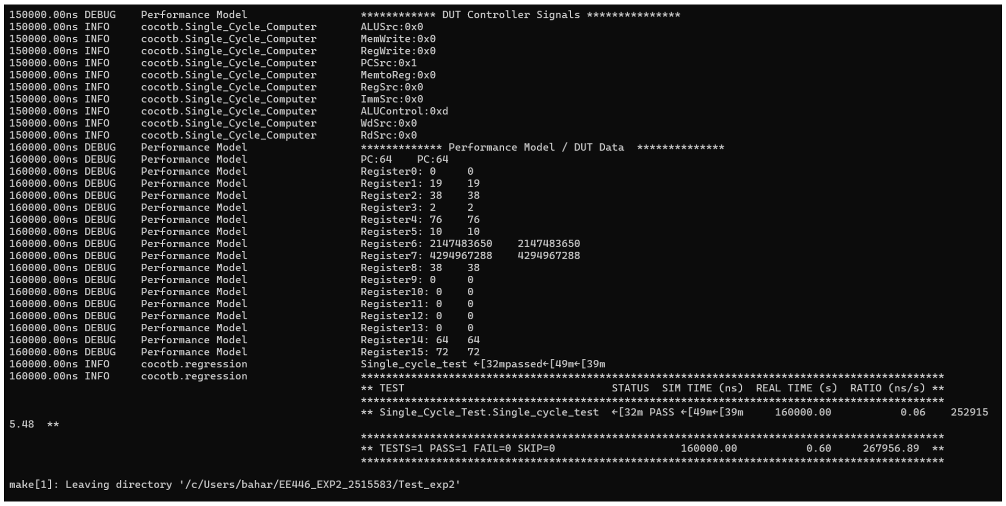


Figure 3: RTL view of the Controller

Testbench Results

For this assignments, we were given a testbench with debugging helpers. The debugging helpers are used during the finalization of the design and the resulting design is able to pass the testbench. The result of the testbench with a **Pass** flag can be seen in Figure 4 below.



```
150000.00ns DEBUG    Performance Model                  ************ DUT Controller Signals ***************
150000.00ns INFO     cocotb.Single_Cycle_Computer       ALUSrc:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       MemWrite:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       RegWrite:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       PCSrc:0x1
150000.00ns INFO     cocotb.Single_Cycle_Computer       MemtoReg:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       RegSrc:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       ImmSrc:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       ALUControl:0xd
150000.00ns INFO     cocotb.Single_Cycle_Computer       WdSrc:0x0
150000.00ns INFO     cocotb.Single_Cycle_Computer       RdSrc:0x0
160000.00ns DEBUG    Performance Model                  ************* Performance Model / DUT Data  **************
160000.00ns DEBUG    Performance Model                  PC:64    PC:64
160000.00ns DEBUG    Performance Model                  Register0: 0     0
160000.00ns DEBUG    Performance Model                  Register1: 19    19
160000.00ns DEBUG    Performance Model                  Register2: 38    38
160000.00ns DEBUG    Performance Model                  Register3: 2     2
160000.00ns DEBUG    Performance Model                  Register4: 76    76
160000.00ns DEBUG    Performance Model                  Register5: 10    10
160000.00ns DEBUG    Performance Model                  Register6: 2147483650    2147483650
160000.00ns DEBUG    Performance Model                  Register7: 4294967288    4294967288
160000.00ns DEBUG    Performance Model                  Register8: 38    38
160000.00ns DEBUG    Performance Model                  Register9: 0     0
160000.00ns DEBUG    Performance Model                  Register10: 0    0
160000.00ns DEBUG    Performance Model                  Register11: 0    0
160000.00ns DEBUG    Performance Model                  Register12: 0    0
160000.00ns DEBUG    Performance Model                  Register13: 0    0
160000.00ns DEBUG    Performance Model                  Register14: 64   64
160000.00ns DEBUG    Performance Model                  Register15: 72   72
160000.00ns INFO     cocotb.regression                  Single_cycle_test ←[32mpassed←[49m←[39m
160000.00ns INFO     cocotb.regression                  *********************************************************************************************
                                                        ** TEST                                STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                        *********************************************************************************************
                                                        ** Single_Cycle_Test.Single_cycle_test  ←[32m PASS ←[49m←[39m     160000.00           0.06    252915
5.48  **
                                                        *********************************************************************************************
                                                        ** TESTS=1 PASS=1 FAIL=0 SKIP=0                      160000.00           0.60     267956.89  **
                                                        *********************************************************************************************

make[1]: Leaving directory '/c/Users/bahar/EE446_EXP2_2515583/Test_exp2'
```

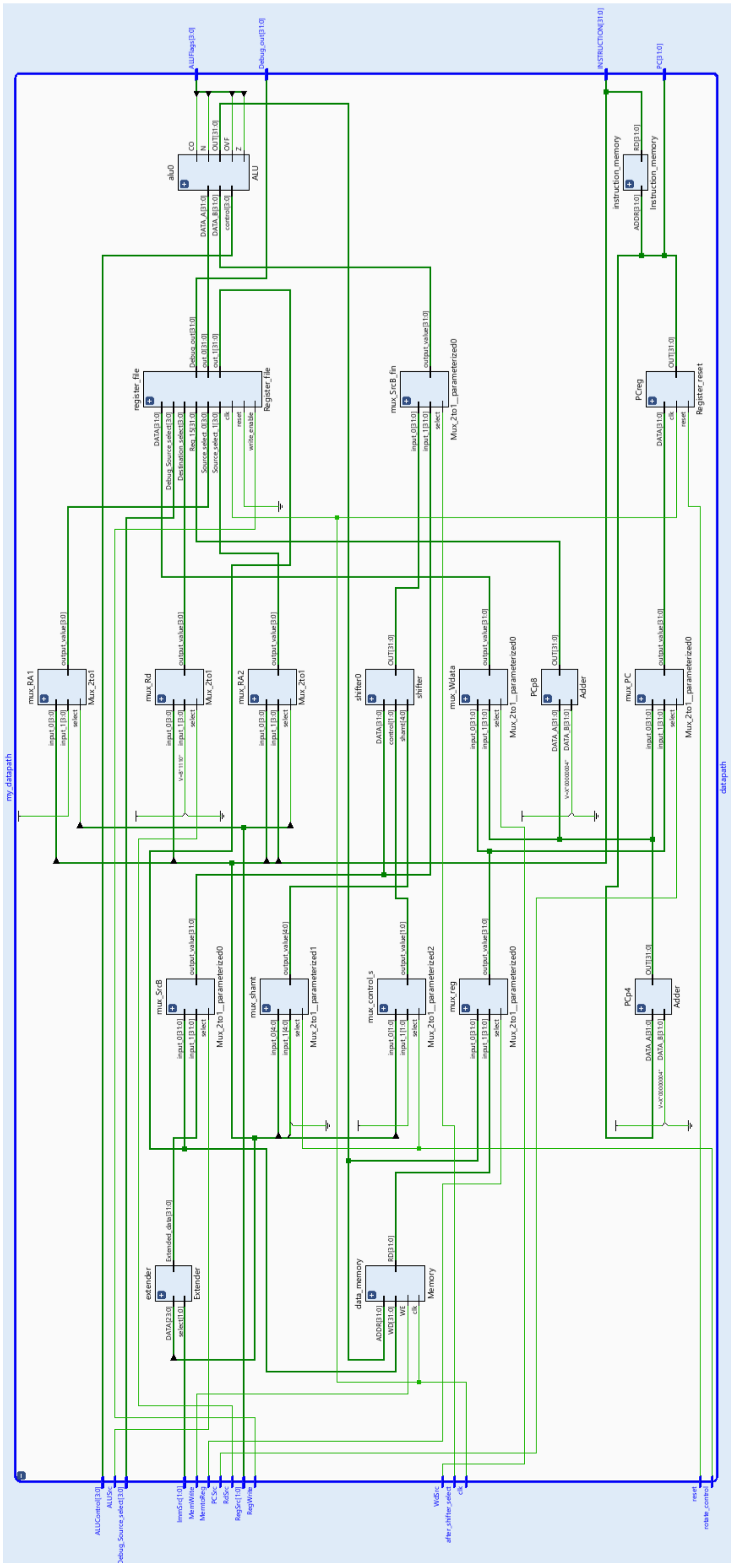
Figure 4: Testbench Results

Conclusion

In this design, the design process of Single-Cycle Processor is examined. A design approach similar to the **Basic Computer Design** we have tackled in the previous semester was sufficient in this assignment as well. The creation of new paths for specific instructions was an insightful tasks for upcoming architecture designs. This tasks allowed me to understand design approaches more which will be beneficial in the upcoming assignments of this course.

Appendix

Appendix I. Better Version of the RTL View of Datapath



Appendix II. Better Version of the RTL View of Controller