The American University in Cairo

School of Sciences and Engineering

Computer Science and Engineering

CSCE 3301 – Computer Architecture Spring 2022

Project 1: RISC-V Processor

Milestone 3&4

Ahmed Dardir: 900182176

Akram Aziz: 900184016

Tamer Osman: 900192103

Project Description

In this final submission of the RISC-V processor project, the single cycle implementation we had at the beginning was transformed into a pipelined version. Subsequently, it was transformed into a single-memory pipelined version accommodating 6 multiplication instructions as well as the basic 40 from the RV32I set. The major change that took place was changing the datapath for the processor from being a single cycle processor to a pipelined datapath. Also replacing the instructions memory and the data memory with one single-ported memory placed in the IF stage. The usage of the clock was changed to the equivalent of stalling a cycle after each instruction to avoid the structural hazards introduced by the single memory approach. In addition, all other possible hazards were taken into consideration and handled. Thorough testing was done to ensure the accuracy of the program.

Technical Summary

The following are the changes implemented modules to change the processor implementation to a pipelined single-memory one.

RISC_V: This is the top module of the processor that organizes all the other wirings, inputs and outputs. Registers were added to store the data between each stage of the pipeline and all necessary signals were propagated in these registers. These registers are IF_ID, ID_EX, EX_MEM and MEM_WB. Several multiplexers were added to accommodate for forwarding, flushing and stalling as well. New defines were added to accommodate for the extra added instructions of the RV32IM instruction set. For the flushing, we only need to flush instructions at the Execution stage, and that is done by selecting whether to propagate the control signals produced by the Control Unit normally or to propagate zeros instead, using the least significant bit of PCSrc signal produced by the Branch Control Unit.

ALU_control_unit: The ALU Control Unit has several case statements to send out control signals to the ALU according to the funct3, funct7(bit 25), bit 30 of the instruction, After processing all those signals, data is sent to the alu_operation module to execute the specific operation that needs to be done.

ALU_Operation: This is the ALU and in it, a fix for a bug that was in all the shift instructions was implemented. It also now takes input from the pipelining register instead of taking the inputs directly out of the outputs of the previous stage. It also now has the implementation of the MUL instruction whether signed or unsigned.

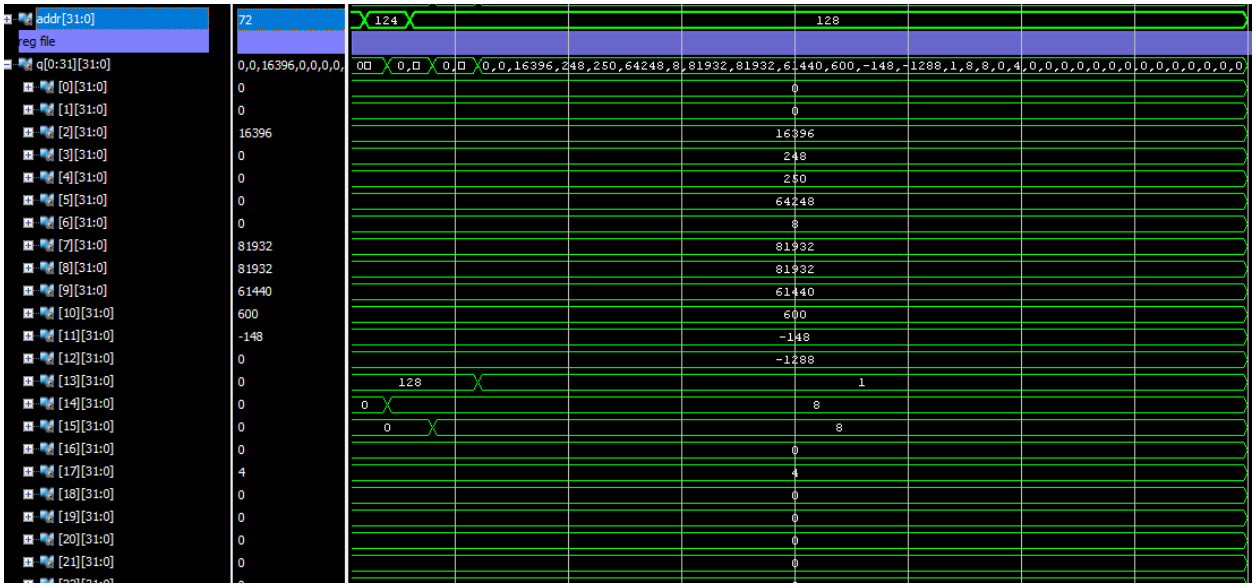
single_memory: This module replaces the modules that were called **data_mem** and **instruction_memory.** This is a single ported memory that contains both the data and the instructions. They are separated by using an offset. A single-ported byte addressable memory that replaced both the data and instruction memories. This was done by dividing the memory so that instructions are in the first part of the memory and data is in the other. The clock signal determines whether to read instructions or read/write in the data memory; at the positive edge instructions are accessed and at the negative edge data is accessed. To fetch data from the lower half of the memory, an offset parameter is added to the given address.

Forwarding_Unit: The purpose of the forwarding unit is to forward needed data from the execution or the memory stages to deal with RAW dependencies. The forwarding mechanism works by comparing the source registers' numbers from the decoding stage to the destination registers' numbers in the execution and memory stages. In addition, we check whether we need to write in the register file in the first place or not, and that that destination register is not the zero register. The Forwarding Unit then outputs 2 signals (forwardA and forwardB) used as selectors to what will be the operands going into the ALU.

Testing Results

For testing, we tested each instruction one at a time to check for any unexpected errors. Then add another instruction to check another supported instruction and check for any dependency. Therefore, we ended up with a one test case that covers most of the instructions while others were removed during testing for the flow of the test cases.

Test Case 1

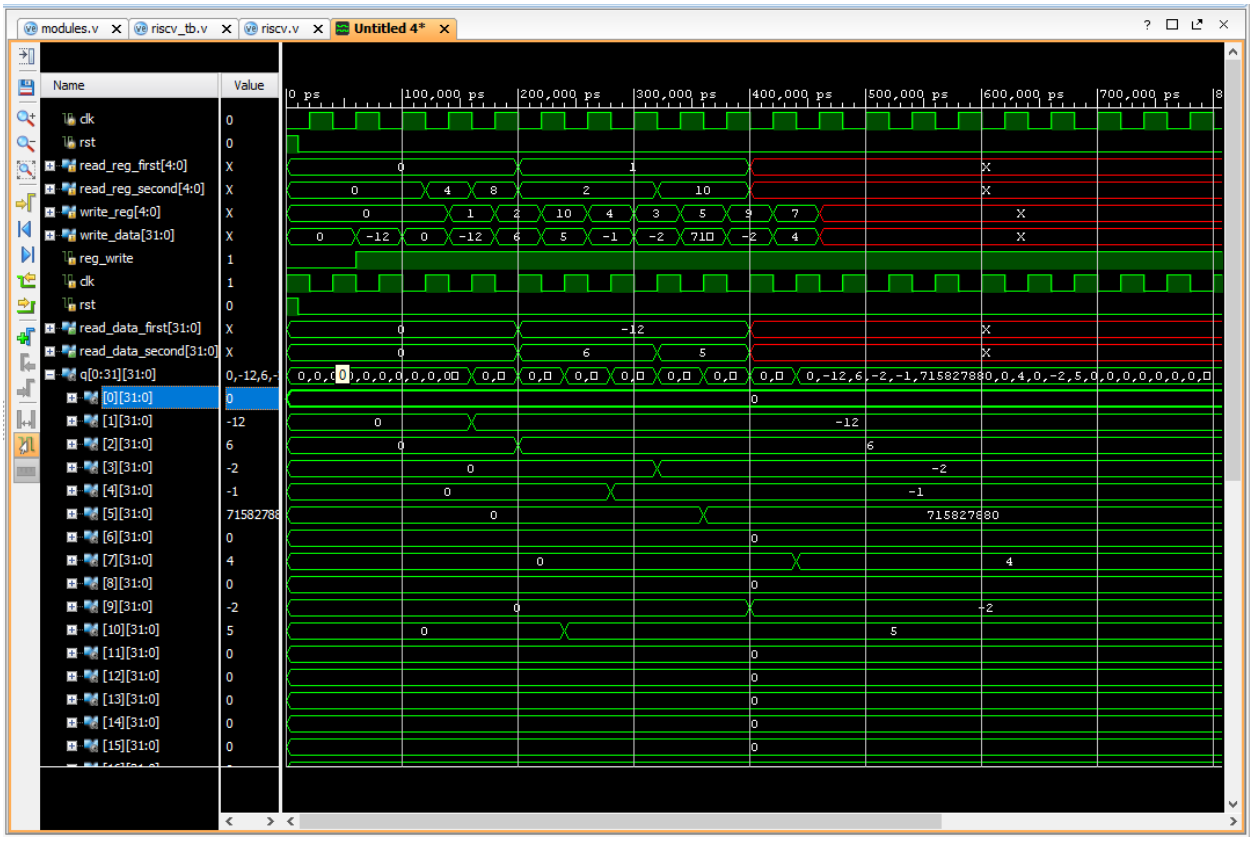


The screenshot above shows validated results for miscellaneous instructions with no specific function for the program as a whole.

Test Case 2

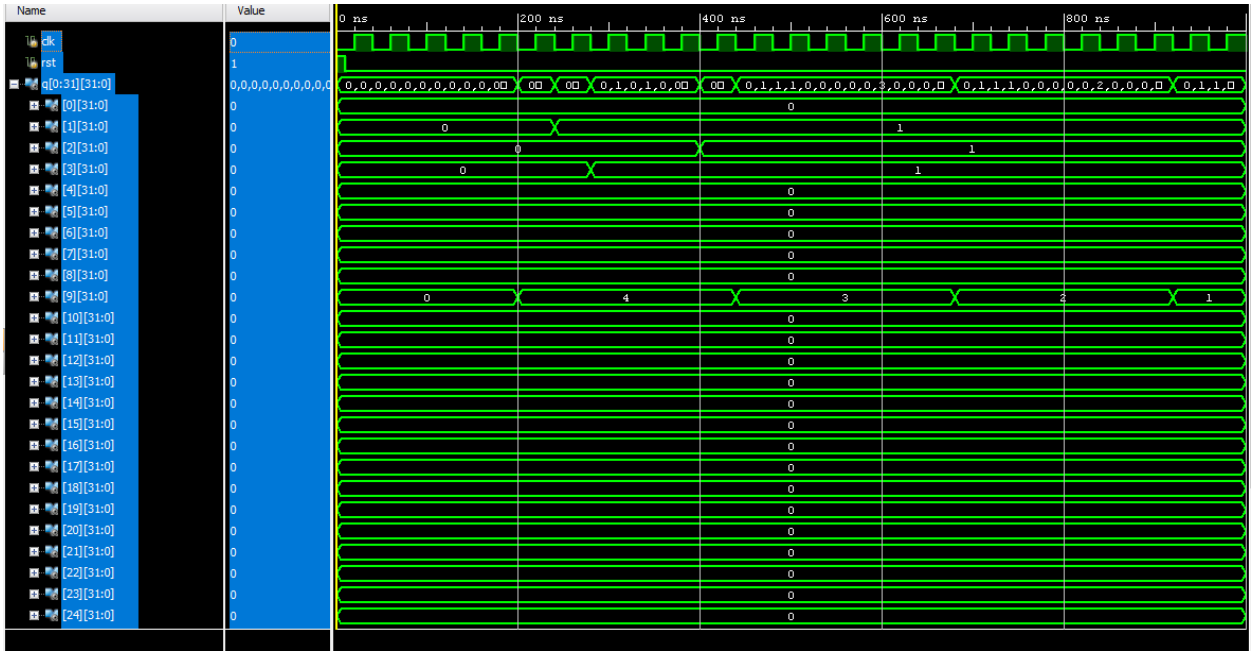
The screenshot above shows results of MUL and MULHSU

Test Case 3



The screenshot above shows the results of the MULHU, DIV, DIVU, REM, REMU

Test Case 4



The above screenshot tests a program implementing a loop that is taken.

Points for Improvement:

- Support the instructions of JAL and JALR, as we managed to know that the register will be updated with the correct value; however, the jump does not happen nor the required flushing.
- Create a cleaner version of the implemented full datapath using more cautious implementation of hardware components.
- Update the Forward Unit to receive the source registers from the previous instruction only, not from both the previous and the one before as in the single memory module only a dependency will occur between two consecutive instructions.

Datapath Block Diagram