CSCE 3301 - computer Architecture

Project (1): RV32IC implementation and testing

Fall 2019

Submitted by:

Mariam Abul-Ela 900141674

Nada Badawy 900171975

Mohamed Al-Awadly 900163100

Instructor: Dr. Sherif Salama

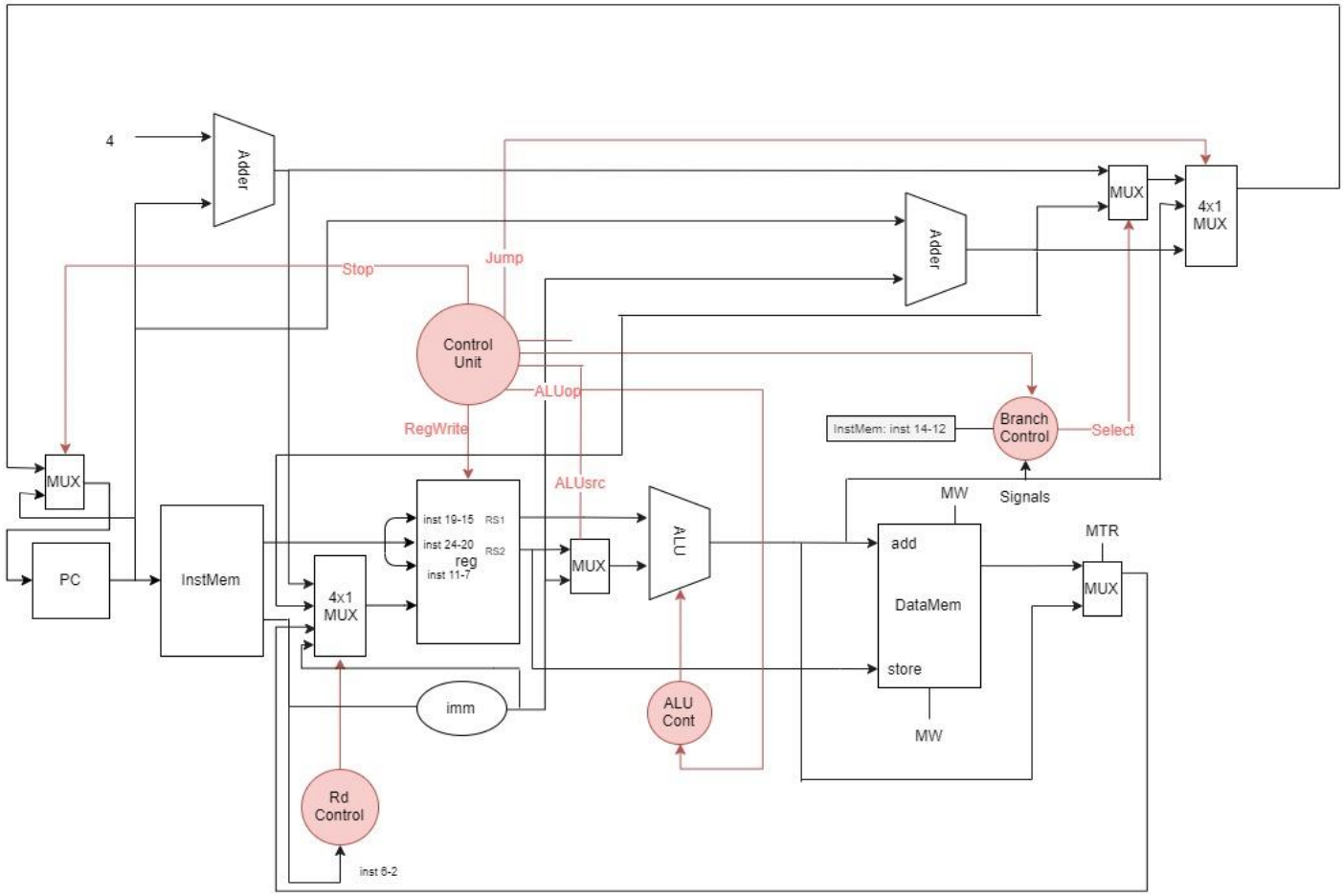
Date: 29/10/19

Project Description:

The target of this project is to implement a RISC-V processor that supports the RV32I base integer instruction set. Yet, the ECALL, FENCE, FENCE.I, and CSR instructions should be implemented as NOP instructions. Also, the EBREAK should terminates the PC.

How to use the processor:

All the module will be the same, except for the instruction memory module and data memory module. In the instruction memory, we change the instruction based on the type of instruction which is being tested; it took the instruction in binary or hex and decode it to read and execute the result. In the data memory we initialize the data in an exact memory location that we need to load to test the instructions.



Modules Description:

The main module is the DataPath where we call all of our modules to generate the flow of the single cycle datapath for the Risc-v processor.

We call the **RIGF** which gets the target_PC and the output will be the address of the instruction. Then we will take this address to the **instruction memory** to be fetched and get the instruction that will be decoded in the **register file**. Also, part of the instruction will be sent to the **Control unit** to get the control signals like Branch signal and ALUop signal. Another different parts of the instruction and the ALUop will determine the ALU_sellect from the **ALU_Control** module. After that we will get the immediate from the **Immediate_Generator**. After the decoding stage, we need to know what the input of the ALU will be. Because it will be different form Format to another, we but a **MUX** to select the second input to the ALU based on the ALU_sellect. Then the **ALU** will have different outputs like zero flag and the output of the operation that is done in the ALU. Then part of this output of the ALU will be select the memory location that the data will be store on it. And this based on the instruction itself and the signal from the Control unit. The output of the **Data_Memory** and the ALU_output will also enter to a MUX to select which one is going to be written in the Rd_Register. Because we have different data that need to be written in the Rd_Register we create a module **Mux_4** to select which one will be written using the select line form the **Rd_Control** module. Also, we create a **branch_control** module that based on the signals of Func_3, jump, branch, zero_flag, carry, overflow, and the sign it decides which branch will be executed. Last part is in the DataPath module is about the PC and how it changes. Because we might have different PCs based on the instruction that we executed like (Add and or) pc = pc+4, and (branch) pc = pc +2*imm, we use a MUX to choose. Also, to continue changing the PC or to terminate the code we create **PC_Controle** module which its output will be a select line in a MUX that select if it is going to continue or to stop.

Note:

You can add from test folder test that you want to run to the Instruction memory.