Zeyad Tolba Kamal | 900192983 & AbdelAziz Yehia | 900203361

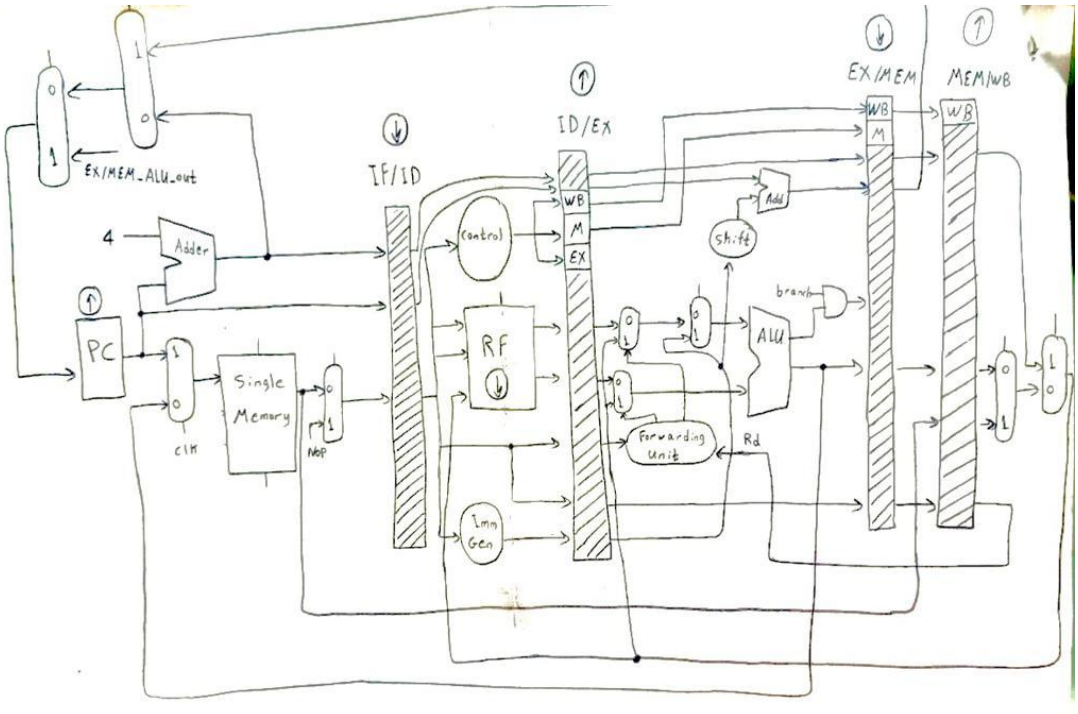
# Milestone 4 Report

Nov 23, 2022

## Technical Summary

In Milestone 4, we created a new memory instead of the Data Memory and the Instruction memory. Now, we have two memories in one module, so we divided our new memory into two parts. First half of the memory was for data and the other half was for instructions. Then, in the integration, we added a mux to select whether to fetch an instruction or to deal with data. If the clock is 1, then we fetch an instruction. On the other hand, if the clock is 0, then we deal with data. Moreover, we handled structural hazards by fetching 1 instruction every two cycles of the clock; this happens by alternating between positive and negative edges of the clock. We also detect the branch and flush the instructions to avoid hazards. In this milestone, we mainly solve the errors and handle all types of hazards to make sure the 40 instructions are working successfully on the simulation and on the FPGA.

## The Final Data path [modified single cycle processor]



Above is the data path we used to implement the 40 instructions in the single cycle RISC V processor USING pipelining.

## Technical Modifications in the code

- We import the code of the pipelined data path from the lab work
- Now we have the registers between every stage so we can update the outputs of the previous stage, and passing them as inputs to the next stage
- We have 4 registers IF_ID, ID_EX, EX_MEM, and MEM_WB.
    - IF_ID will take the 32-bit instruction, PC,  and PC+4 to handle AUIPAC and JALR
    - ID_EX will take output of the register files, PC value, PC+4 , the immediate value, 2 source registers and destination register.
    - EX_MEM will take the signals of the control unit, Pc, PC+4, PC+imm, zero flag, ALU output
    - MEM_WB will take WB, regwrite,memToreg, memory output, PC+4, the destination register, ALU output,.

## Handling Hazards

As the pipelining system saves much time, it leads to many types of hazards. For Data hazards, it may happen when we try to read a register that's supposed to be written by the previous instruction. For Control Hazards,  it may happen when we are executing a branching instruction. Below are the details how we handle each hazard in specific.

- **Data Hazards**

    **RAW(Read After Write) hazards**

Consider a scenario in which an add instruction writes to Register x6 and is followed by two instructions that read from the same Register x6. The changed value of Register x6 will be written back into the register in the WB stage of the standard pipelined datapath. The two instructions after it will simultaneously be in the MEM stage and the EX stage, indicating that it previously used the wrong value for the register. We must thus convey the two instructions. The following instruction will be in the EX stage when the changed value of the register is written back, and the instruction after them will be in the IF stage because our system fetches one instruction every two cycles. Thus, the instruction that comes right after the add instruction is the only one that needs to be forwarded.

### Load-use hazards

When we have a load instruction, we usually stall for one cycle then forward. However, we will do one forwarding from the WB stage to take the updated value from there and use it in the EX stage.

### Forwarding Unit

It should be the forwarding unit's responsibility to determine whether a Read After Write situation exists, regardless of whether it was caused by an R-type, I-type, or load-use instruction. While the remaining instructions can make the forwarding from either the MEM-WB registers or the EX-MEM registers as they will not utilise the memory, the load-use instructions must make the forwarding from the MEM-WB registers since the updated value comes from the memory output. Therefore, we limit the forwarding unit to checking the MEM-WB registers in order to keep things simple and handle all data dangers in one instance. Forward the modified value by setting either forwardA or forwardB to equal 01 if the instruction in the MEM-WB stage writes to the same registers that the instruction in the EX stage reads from. In order to choose the inputs of the ALU if they become the same or will be replaced by the forwarded value based on forwardA and forwardB, we added two additional MUXes.

- **Control Hazards**

### Branching Instruction hazard

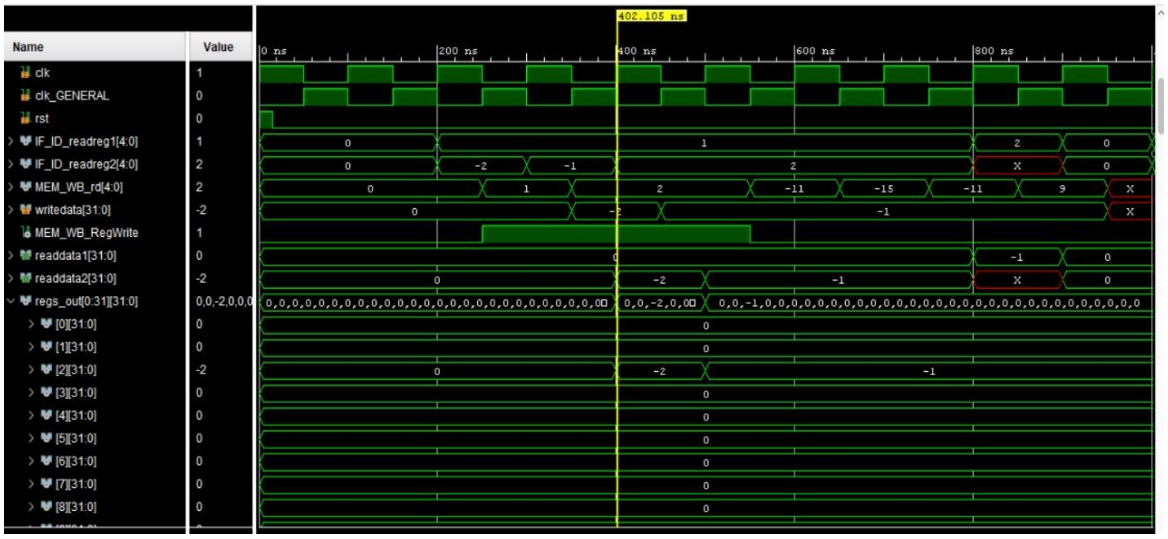
When we need to jump to a specific PC which is not the next instruction (PC+4) as normal flow of the CPU, , there will be a number of instructions already fetched and entered in the stages of the processor. So we need to first flush all the registers to delete these data which will allow us to enter our proposed instruction in the pipeline. At the conclusion of the EX step, we check the zero flag from the ALU output and the branch signal to see if the PC will be altered or not. We will already have one set of instructions in the IF stage at this point. As a result, we implemented an additional MUX in the IF stage that determines whether to pass the enforced NOP instructions (add x0, x0, x0) or the conventional fetched instruction bits to the decoding stage based on the value of AND out, which verifies the branch signal and zero flag.

## Verilog Description files

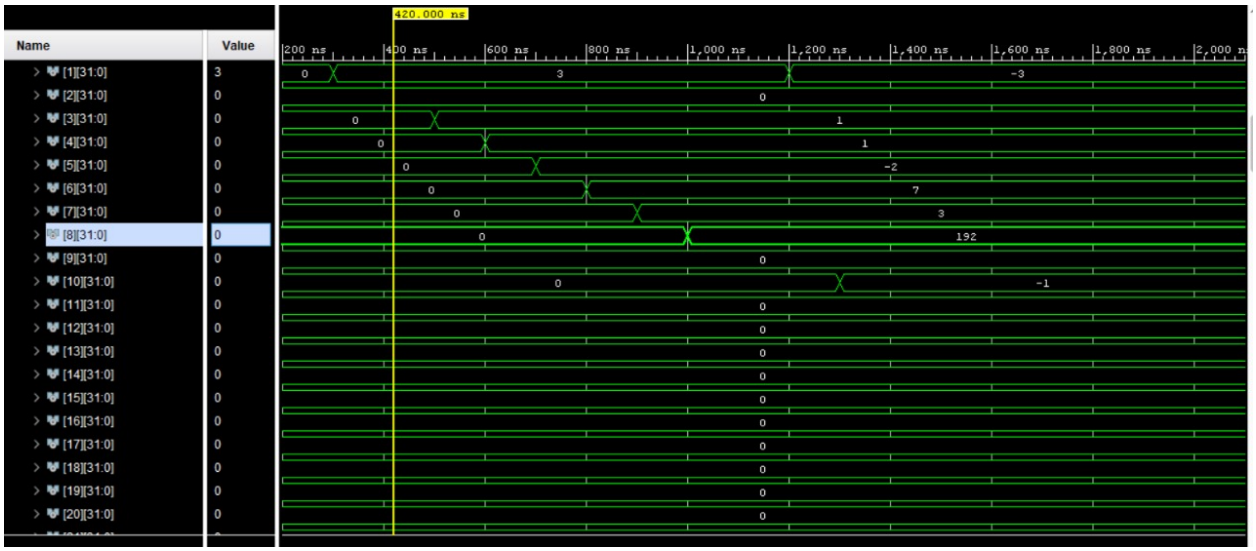
The Verilog descriptions supporting all of the RV32I instructions are included in the zip file.

## Simulation and Testing for 40 instructions

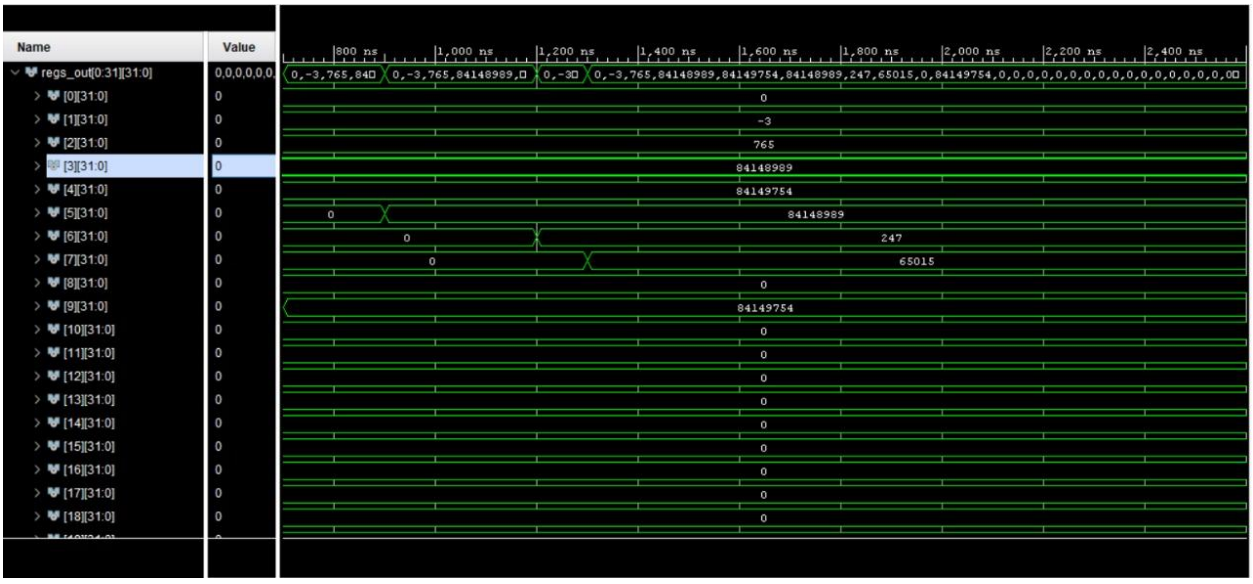
- Branches Instructions
    - Simulation Testing and screenshots



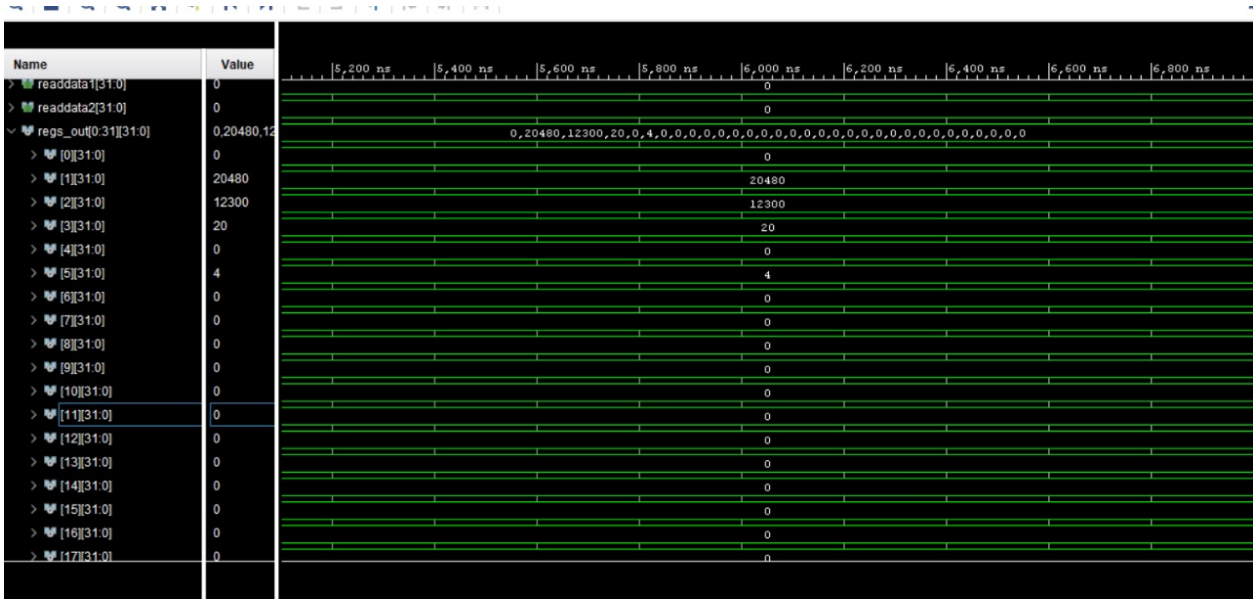

- I Instructions
    - Simulation Testing and screenshots



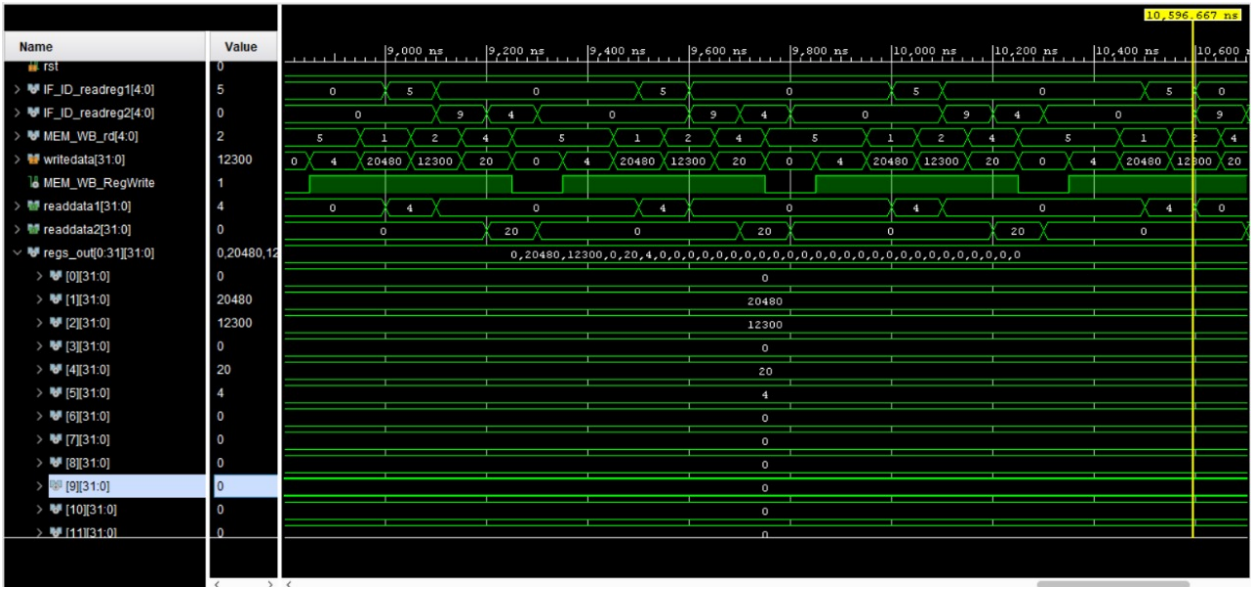
- Load and Store Instructions
    - Simulation Testing and screenshots



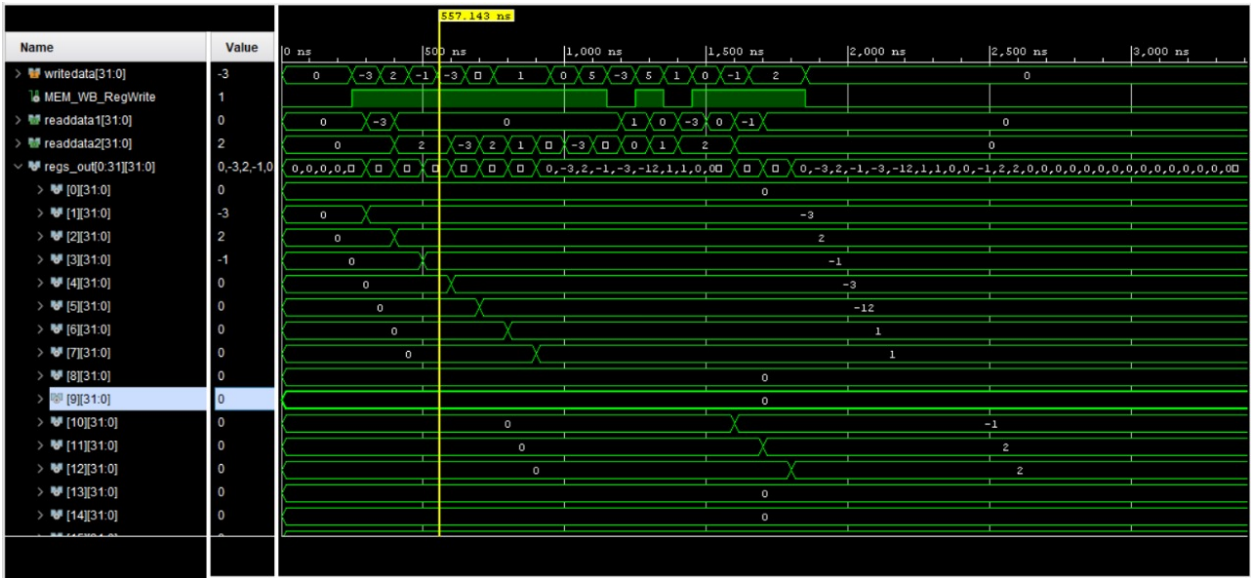

- LUI,AUIPC,JAL Instructions
    - Simulation Testing and screenshots



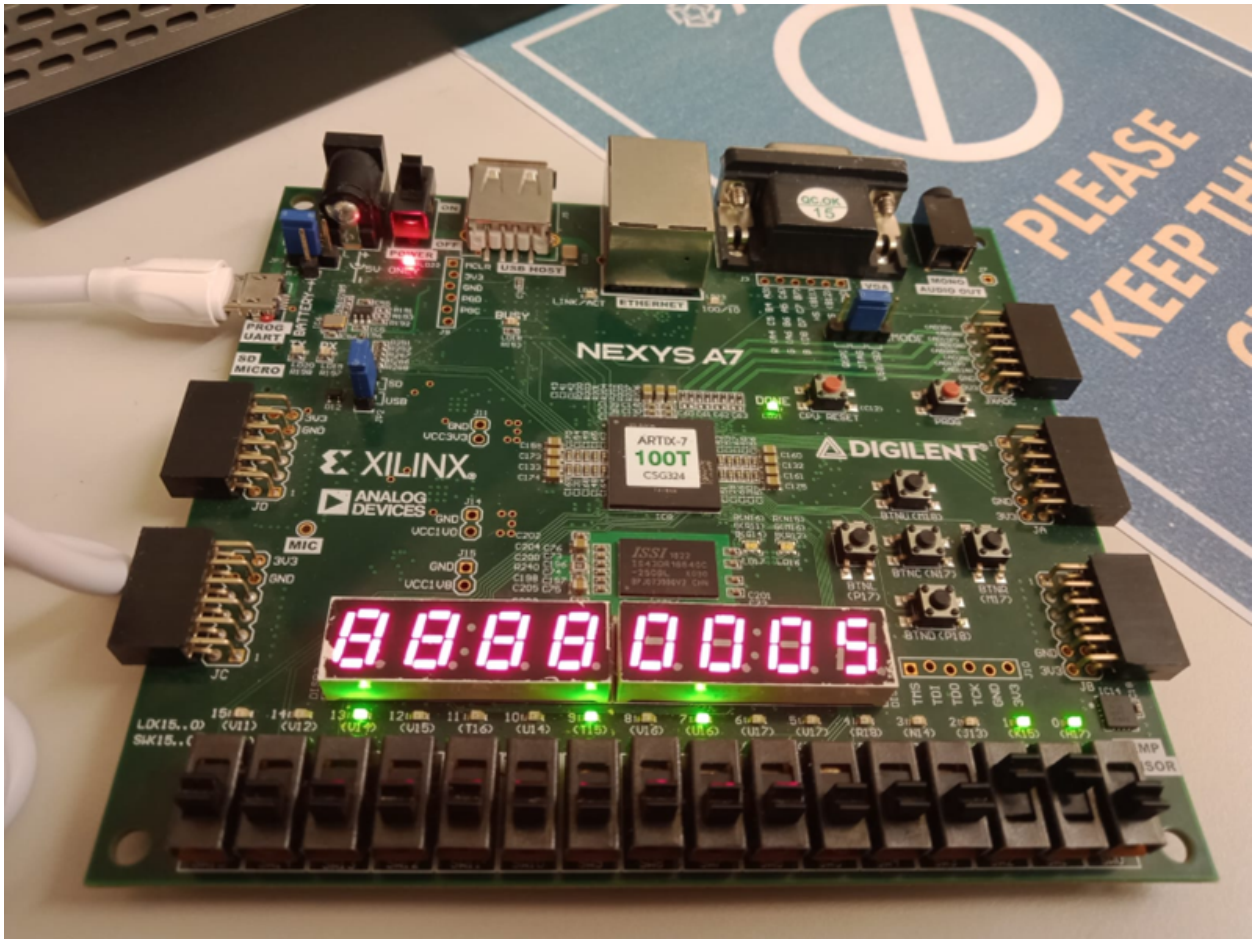
- LUI,AUIPC,JALR Instructions
    - Simulation Testing and screenshots




- R Instructions
    - Simulation Testing and screenshots



## FBGA Testing and screenshots




**THIS IS THE END OF THIS DOCUMENT**