CS2323- Early Implementation and Challenges Report

Name- Vemula Siddhartha
Roll number- EE23BTECH11063
No group, done solo.

Implementation Status

- Implemented a RISC-V 64/32 (configurable using parameters) I single-cycle processor.
- Implemented the following set of instructions: R-type, I-type, S-type.
- Implemented the following blocks:
    - Control Unit:
        - Takes the instruction as an input.
        - Gives outputs:
            - ALUCtrl (The ALU Control signals, which are encoded based on the instruction and which operation the ALU has to perform based on it).
            - branch (high whenever there is a branch type of instruction, whenever PC is added to an immediate value).
            - MemRead (high whenever a value has to be read from the data memory).
            - MemtoReg (high whenever a value from the data memory has to be written to a register, mux select).
            - MemWrite (high whenever a value is written to memory).
            - RegWrite (high whenever a value is to be written to a register).
            - MemSign (high when the value to be accessed from memory is unsigned and low when signed).
            - MemWidth (different encoding for the length of the memory to be accessed- byte, half, word and double).
    - ALU:
        - Takes register (rs1 and rs2) inputs and an immediate value input.
        - Takes ALUCtrl input.
            - ALUCtrl input specifies which operation the ALU has to perform for a given instruction.
        - Gives two outputs: The ALU output (rs1 operator rs2 or rs1 operator imm) and a Zero (rs1 == rs2?, which is used for branch instructions) output.
        - Performs the following operations: add, sub, xor, or, and, sll, srl, sra, slt, sltu, addi, xori, ori, andi, slli, srli, srai, slti, sltui.
    - Immediate Generate Block:
        - Takes the instruction as an input.
        - Gives the signed immediate value from the instruction encoding as an output.

        - Generates the correct immediate value from different instruction encodings as an output of size REG_WIDTH (configurable width of the registers).
    - Register File:
        - Takes inputs of clock (synchronization signal) and reset (reset the register file values to 0 at the start).
        - Takes input writeEnable (signal which is high when a value is to be written to a register), waddr (register address to which the value is to be written to), wdata (the data to be written to the register).
        - Takes inputs raddr1 (address of the first read register), raddr2 (address of the second read register).
        - Gives outputs rs1 and rs2, the registers which are being read.
        - Implicitly does not allow any changes to register x0.
    - Data Memory (Auxiliary Code to emulate data memory):
        - Takes input clock (synchronization signal).
        - Takes the MemRead output of the Control unit as an input (to check if the Memory is being accessed for reading).
        - Takes the MemWrite output of the Control unit as an input (to check if a value has to be written to the Memory).
        - Takes the MemSign and MemWidth outputs of the Control unit as inputs (functionality as mentioned previously).
        - Takes a wdata input (the value which has to be written to the memory when MemWrite is high).
        - Takes full_addr input (the address at which the memory is to be accessed for read and write operations).
        - Gives as output the data which is accessed by read instructions.
        - A 1KB memory is emulated for simulation purposes.

Verification Procedure

- All the modules had individual testbenches written through which their working was verified.
- There was a top module which connected all the before mentioned blocks. It consists of all the proper interconnections for the correct working of the processor.
- A top module testbench was also used, through which the working of the processor as a whole was verified.
- All the instructions implemented were verified.
- The instructions were read from an external file, which had a set of instructions in their machine code.
- The machine code was obtained from the Ripes Simulator.
- A bunch of corner test cases were also checked, for the correctness of the processor.

Further implementation

The following things are to be implemented:

- Emulate the instruction memory, and test the working of branch type instructions.
- Implement the M-extension, and integrate it with the existing processor.

- Flash the single-cycle processor onto the FPGA and verify its working on hardware. Get the clock period and resource utilization.
- Implement the Pipelined version of the processor, and verify its working using simulations.
- Implement Data Forwarding and Hazard Detection.
- Verify their working using testbenches.
- Flash the final code onto the FPGA and verify its working on hardware. Get the clock period improvement and resource utilization.

Expected Challenges

- The processor might not work as expected on the FPGA. Will have to take some time to debug.
- A few corner cases might be missed while trying to implement data forwarding and hazard detection. Will have to come up with a robust testbench setup to properly verify the working of the processor.