RISC-V FPGA Implementation

Milestone 3

Mohamed Abdelfattah

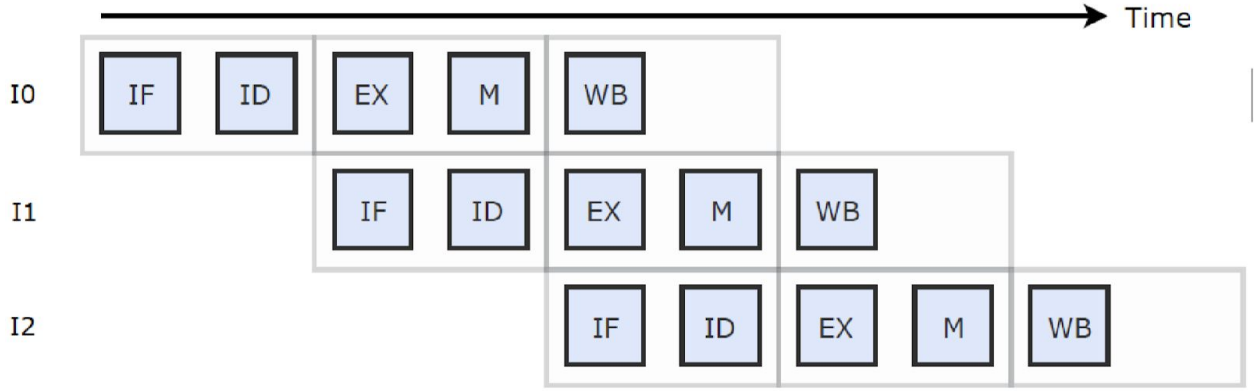
Muhammad Azzazy

Marc Boutros

Abdallah Gabara

Requirements

To create pipelined implementation of the RISCV32I architecture. The implementation supporting all the instructions. It can handle all kind of data dependencies and hazards including the load-use data hazard. The implementation requires us to fetch an instruction every two clock cycles. The implementation is divided into 3 stages: Fetch and Decode, Execute and Memory, Write Back. This is a figure of the pipeline:



This implementation is mainly designed to use a single memory for both instructions and data.

Modules

- Control Unit:

Outputs the control signals controlling different modules in the data path to fit the instruction being executed.

- ALU Control:

Responsible of outputting the selection line for the ALU based on the instruction in the execution phase and the ALUOp signals from the Control Unit.

- Data Memory:

A byte addressable memory divided in bytes. In the reading mode, the memory outputs a word based on 3 signals it receives from the Control Unit (by, half, unsign). In the case of, byte or half reading the memory outputs the required byte or half with sign extension or not depending on the unsign signal. Otherwise, the memory outputs a word conventionally.

- Branching Unit:

A specialized unit that receives the flags of the ALU and determines whether to go for branching or not provided that the instruction in execution is a jump or branch.

- Memory Control Unit:

Consists of multiplexers to pass a low signal when an instruction is being fetched instead of the load, byte, and unsigned signals passed by the control unit in order to load the instruction correctly as a 32-bit word

- Forwarding Unit:

Responsible of forwarding results from previous instructions to newly executed instructions whenever necessary to avoid potential errors. The unit is also responsible for solving any hazards through producing a stall signal that stops the PC from fetching any new instruction.

Work procedure

We started by adding the pipeline registers to the single cycle implementation. The next step consisted of multiplexing the inputs to the single ported memory and demultiplexing its output. A forwarding unit was added to account for the hazards added with the pipelining of the datapath with minimal stalling. Since the branching is now implemented in the instruction decode stage, there is only one hazard which occurs when one of the compared registers is being loaded from memory by the previous instruction. The hazard was accounted for by the forwarding unit to reduce the amount of modules and wires in the top module since a separate hazard detection unit will require some of the inputs already given to the forwarding unit. To allow a single port memory to be used for both instructions and data, an instruction is issued each 2 clock cycles as required; this is done by creating a secondary clock with half the frequency of the main clock, assigned to the program counter preventing it from updating during the clock cycle when the memory is used for data.

Testing strategy

To test all of the supported instructions we loaded a test program into the memory alongside its data. We stored an initial program alongside some data in some positions in a memory file that we read in the initialization of the memory. We ran the program and made sure all instructions are fetched, decoded, executed and written back successfully. We managed to be sure of the results by inspecting the signals after simulating the Verilog descriptions.

Block diagram