MULTICYCLE-RISC-IITB

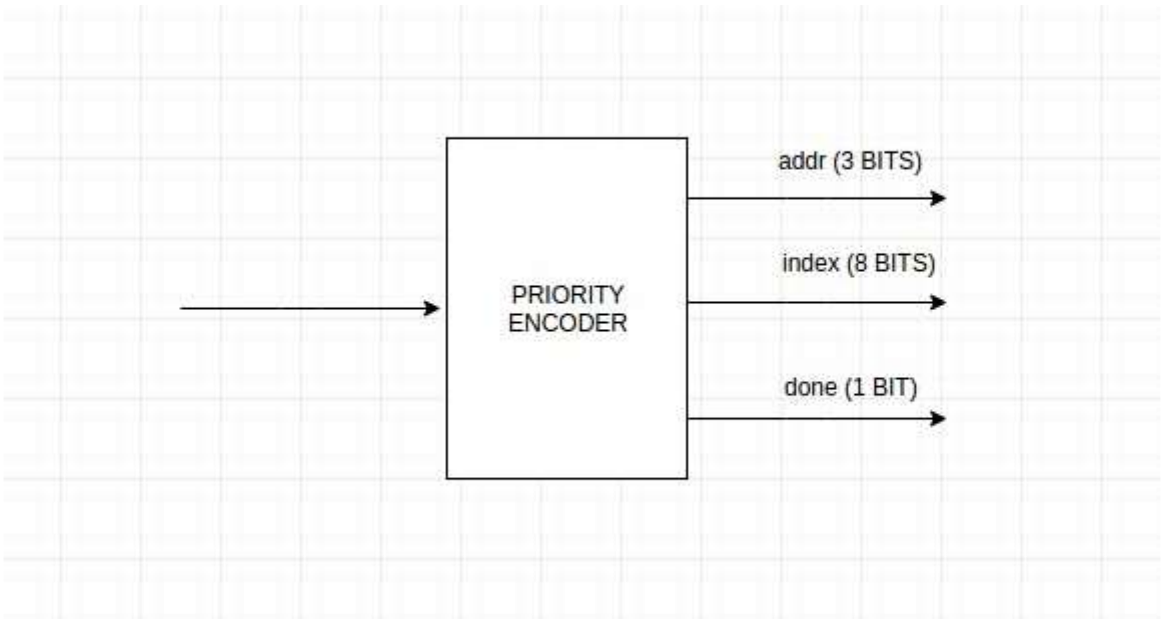
AKASH DOSHI 140010008
DEBARNAB MITRA 140070037
ADITYA GOLATKAR 14B030009
RUDRAJIT DAS 140020012

Part of the project that was challenging/ interesting:

- **R7 serving as both a normal register and PC** : As per our implementation, the increment of PC is done in S1 and the incremented value of PC is written back in S2. S2 is executed only in case of non-jump instructions. Therefore in non jump instructions, if R7 is the destination, the new value would overwite PC+1 in the register writeback stage. If R7 is the source, however, the value of PC should not be incremented, else the incorrect value would have been used. For this purpose, we made an additional state S0 and an additional register T5 such that every non-jump instruction would terminate in S0 where only the value of PC would be incremented.

- **Delay in setting of zero flag**: Zero flag is set by ALU, Hence its value can be used only if ALU operation is completed. In BEQ, we were checking equality in the same state where the ALU equality operation was carried out. As a result, the zero flag value was checked at the rising edge of the clock(t = 0) but the zero flag was written to at the falling edge of the clock(t =T/2). So we created an additional state, S10_1, before which the zero flag had been set, and only in this state did we check its value.

- **LM/SM instruction**: We used a priority encoder, which had the input as the 9 bit immediate data, and three outputs, addr(3 bit) , index(8 bit) and done(1 bit). It would scan the input from the LSB, searching for the first one. As soon as it found it, it would set addr to the position of the one, index to a binary value whose only zero bit would be the position of the one. Addr would be used to find the register to be loaded/stored into. Index would be "AND" with the input and sent to the input of the Priority encoder again. When the input contained only zeroes, done would be set to 1.



Conversion from behavioral to synthesisable code:

After drawing the datapath, for each state we identified which line would be needed to selected in which multiplexor and came up with a Multiplexor Encoding and corresponding Control Signals(appended to this submission). The value of all control signals in all the states was noted down, and then each row(corresponding to a single state) was simply copied into the Control Path.

We have used a Mealy FSM, where the Control Path is essentially a RTL in which an FSM sets the value of the control signals in each state which are sent to the Data Path, and the next state and output is a function of the input and current state(here the only inputs are clk and reset. There are no outputs of the Microprocessor overall). The Data Path is a Structural architecture consisting of a sequence of port maps which are executed as and when their input changes.

Verification Procedure:

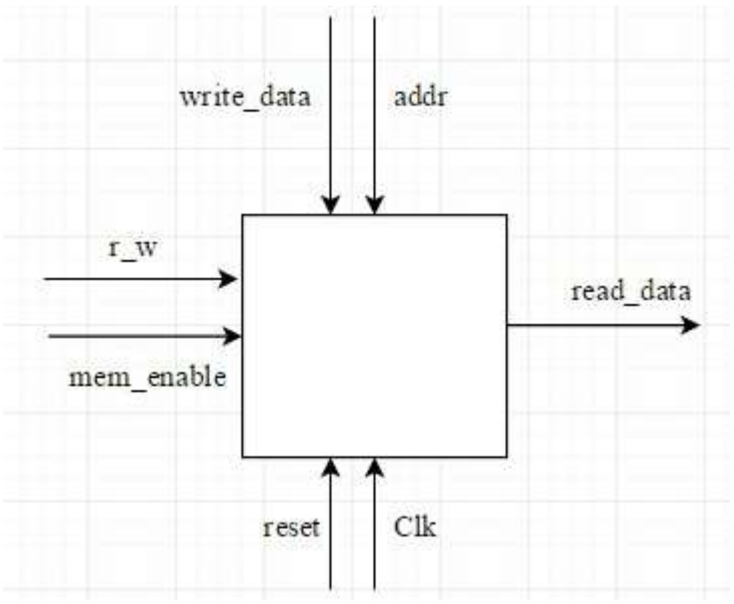
We hardcoded the instructions to be tested in the initialization of the memory. These were loaded into the memory as soon as the reset pin was set low. All the instructions that were tested are in the memory_new. vhd file. We then checked the register and memory values as part of Software simulation in GTKWave, and the same values in SignalTap II Logic Analyzer as part of Hardware Debugging. Both parts were successful.

To run the simulation(GTKwave), simply run the script file in terminal as ./compile_ghdl2.sh . It will generate a run.ghw file in which you can check the signal values.

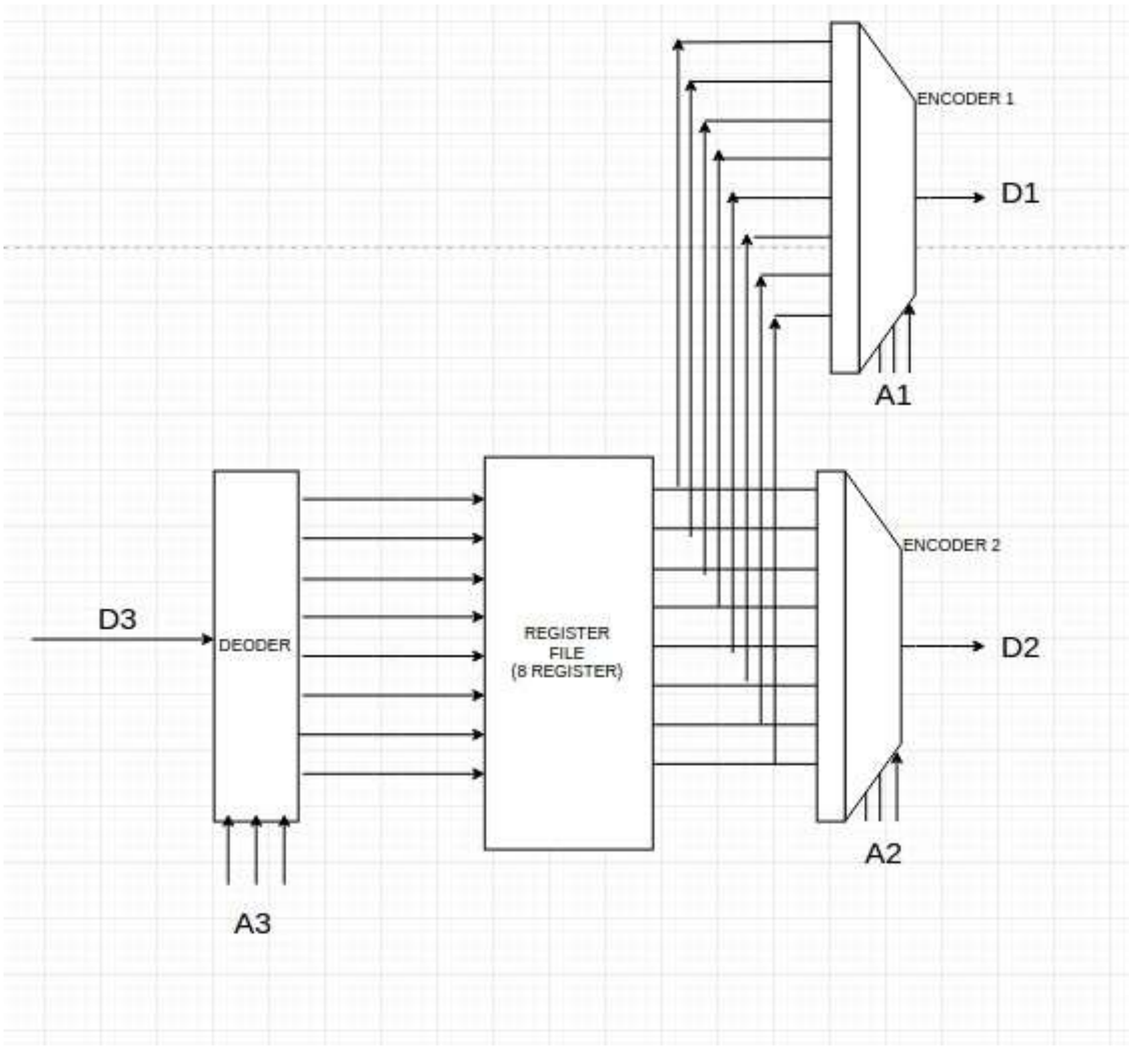
To view the signal tap waveform open the quatrus project 'Toplevel.qpf ' using altera quartus. Compile the project. Upload the .sof file generated onto the FPGA. Open SignalTap II Analyzer to see the required waveforms. (For detailer steps look at the Presentation_de0 PDF uploaded on moodle.)

VHDL Description of entities used in Project:
1. ALU: ALU supports Addition, Equality comparision, Logical AND and Logical NAND . It has an enable signal, two control bits, and produces the result and carry & zero flags as output. These carry and zero flags are written to register only if the corresponding enable signal is high.

2. Memory:



3. Register File: It is a store of eight registers where the 8th register r7 is supposed to be the Program Counter. The RF consists of an decoder, eight registers followed by two encoders. The decoder is used for deciding which register is the write_data to be written to. The output of the decoder is 8 lines of 17 bits each, the 17th bit will tell the registers which line to accept data from. The encoders take the address as the select line, and produce the data at that register address as output.



4. Priority Encoder: (Discussed before)