Indian Institute of Technology Bombay

EE 739 Processor Design(S1)

Course Project

TEAM ID - 8

---

6 Stage Pipelined Processor - IITB_RISC_25

Team Members:

Bala Murugan S (24M1173)
M Lakshmi Deep Chowdary (24M1150)
Aditya Prakash Tare(24M1169)
Tejavanth Kothani (23M1196)

Course Instructor:

Prof. Virendra Singh

Contents

List of Figures

1 Contribution Table

Team Member	**Contribution**
Bala Murugan S (24M1173)	Control Unit, Register Read and Execute stages
M Lakshmi Deep Chowdary (24M1150)	Fetch, Branch Predictor and Memory Access
Aditya Prakash Tare (24M1169)	Control Unit, Register Read and Execute stages
Tejavanth Kothani (23M1196)	Decode, Writeback and Register Files

2 Overview of 6 Stage Pipelined Processor

A 6-stage pipelined processor comprising the following stages: `Fetch`, `Decode`, `Execute`, `Memory Access`, `Writeback`, and `Control Unit`. Initially, without a Branch Predictor, the Control Unit provides the next PC to the Fetch block, which asserts `valid_f` along with the instruction to the Decode block. The Decode stage analyzes the instruction and issues opcode and operand addresses with `valid_d`. The Register Read block accesses the Register File, where READ is combinational and WRITE is sequential. Operands `data_a_rr` and `data_b_rr` are passed to the Control Unit, which handles data forwarding to produce `data_a` and `data_b` for the Execute stage. The ALU processes the instruction, and the result propagates through Memory Access to the Writeback stage. Writeback is combinational to ensure that each instruction completes in one cycle per stage, achieving a 5-cycle latency. With the Branch Predictor integrated, PC values flow from R0 through the Control Unit to the BP stage, and then to Fetch. On a misprediction, the corrected PC is determined using history bits and written to R0, aligning future instructions with the correct program flow.

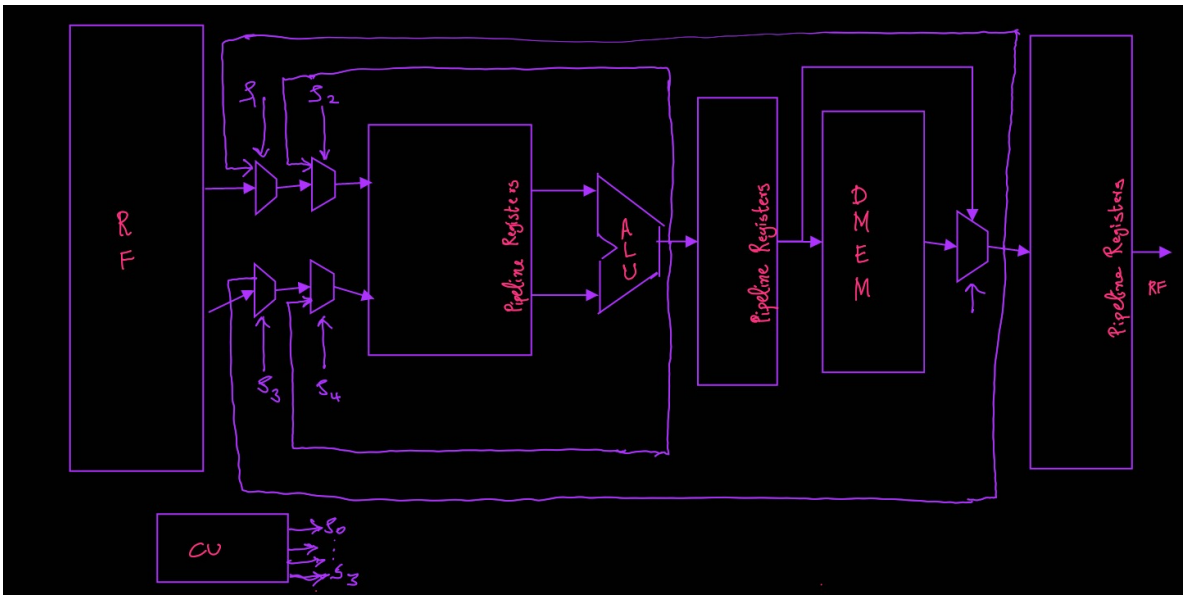

Figure 1: Block Diagram depicting Data Forwarding

3 Synthesized Netlist

The following image shows how how the blocks are synthesized by the Vivado tool, we fixed an FPGA board called `pynq-z2`, to view the synthesize these modules into blocks and to view this netlist.

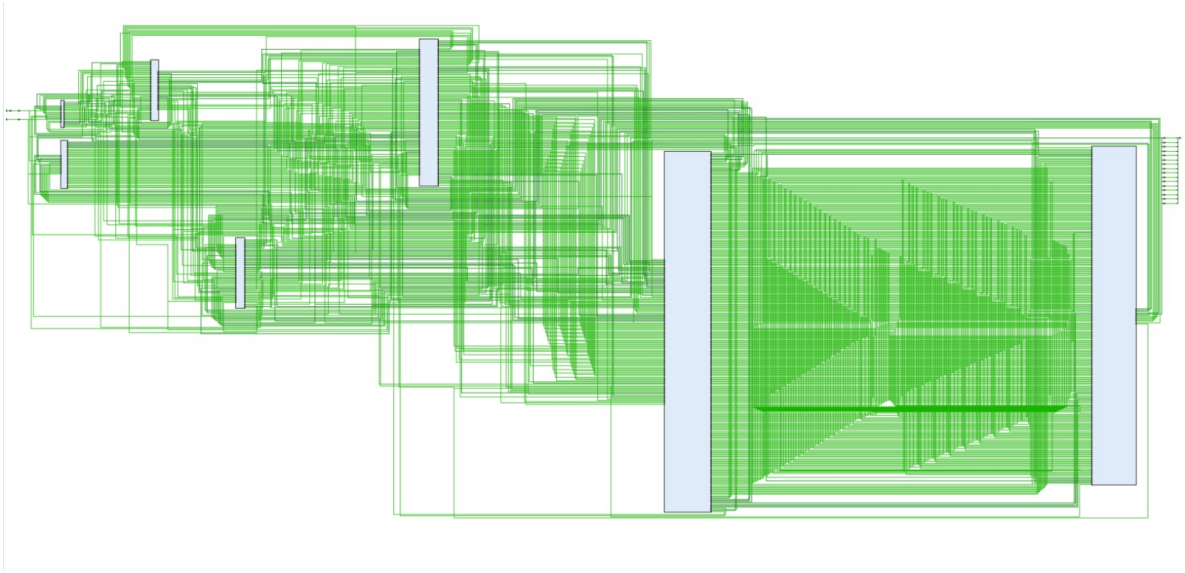

Figure 2: Synthesized Netlist of 6 stage Pipelined Processor

4 LM, SM, ADD and LLI instructions

The following snippet shows how we tested `LLI`,`ADD`,`LM` and `SM`. In the first three instructions, we can see that third instructions is dependent on the first two instructions the control unit resolves it then proceeds to LM instructions. The LM instruction is implemented in such a way that the decode block detects this instruction and sends a `mult_freeze_cu` to the control unit which freezes the fetch and decode block, then RR stage will work upon this instructions loading the registers one by one, the decode block will have a counter which counts from 7 to 0, when the counter reaches 0, the `mult_freeze_cu` signal is kept `LOW,` the enable signals are made `HIGH` for both of the frozen blocks. Then the STORE instruction coming next to the ADD instruction also has a dependency, that is also resolved by the control unit, while the same process goes on for SM instruction.

```
initial begin
//LM and SM testing
rom[0] = 16'b0011011000001000;  //LLI R3 ,8    r(3) = 8
rom[1] = 16'b0011101000001001;  //LLI R5, 9    r(5) = 9
rom[2] = 16'b0011100000001000;  //LLI R4, 10    r(4) = 10
rom[3] = 16'b0000011101110000;  // ADD R3, R5, R6, 000
rom[4] = 16'b0110101011011000;  //LM R5 011011000
rom[5] = 16'b0000011101110000;  // ADD R3, R5, R6, 000
rom[6] = 16'b0111110001111000;  // SM R6 001111000
```

During reset state, we loaded the data memory, in the following way,

```
if(rst) begin
    mem_read_data <= 0;
    ram[0] <= 16'd4;
    ram[1] <= 16'd9;
    ram[9] <= 16'd16;
    ram[10] <= 16'd20;
    ram[11] <= 16'd24;
    ram[12] <= 16'd28;
end
```

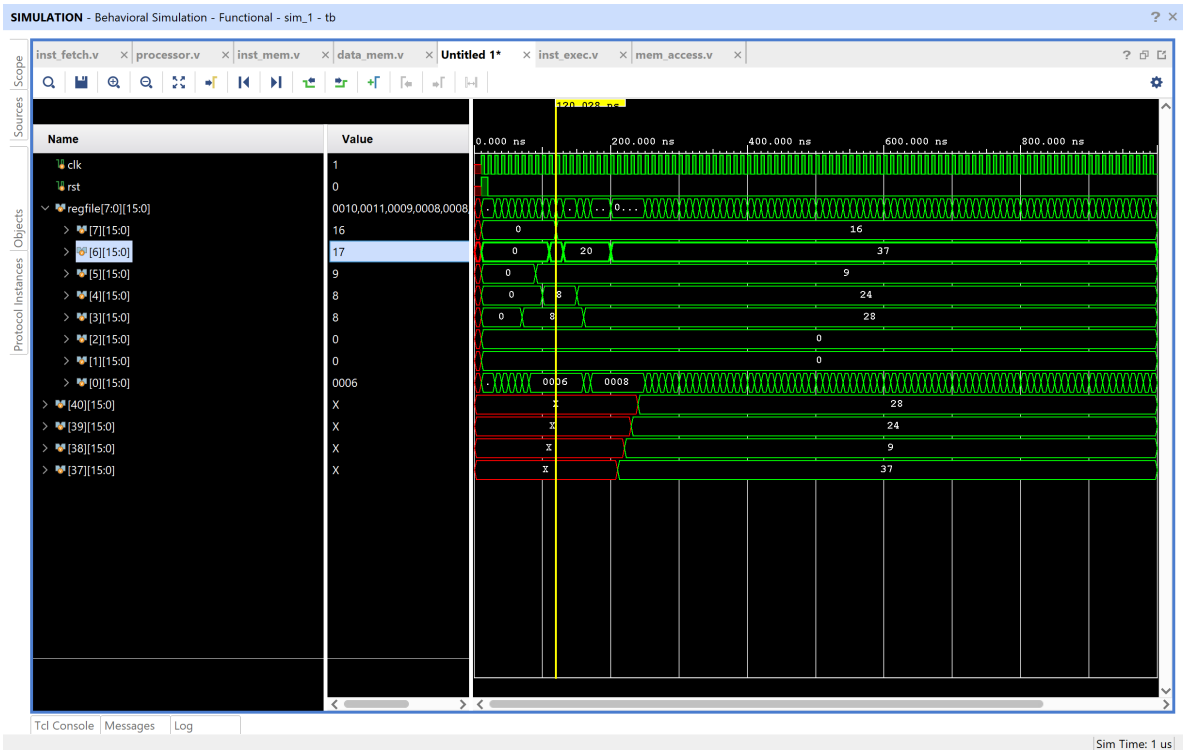

Figure 3: Output Waveform of RF and Data Memory

5 Branch and Jump instructions

Initially, there will be no entry of the respective pc in branch predictor, so there will be a mis-predict for every branch and since the branches taken in the first run the contains of the previous stages are flushed so correct entry is made into reg file giving rise to cycle penalty and since the branches are taken and corresponding branch history bits are updated in the branch history table, in the next after jump instruction there will be no mispredictions (with no cycle penalty). The JAL instruction is detected in `Decode` stage itself, the control unit sends the proper control signals to fetch the correct pc and flush the old one, so in this process 1 cycle will be wasted whereas branch instructions after detected by the Predictor will never give any cycle penalty and no flushing is required. So, JAL will incur 1 cycle penalty other branch and jump instructions which are detected in Execute block will be made an entry into Branch History table and never gives a cycle penalty after the first occurence.

```
initial begin
rom[0] = 16'b0011011000001000; //LLI 0011 011 000 001 000   r(3) = 8
rom[1] = 16'b0011101000001000;  //LLI 0011 101 000 010 000   r(5) = 8
rom[2] = 16'b0011100000001000;  //LLI 0011 100 000 010 000   r(4) = 8
rom[3] = 16'b1000100011000010;  //BEQ 1000 101 011 010010
rom[4] = 16'b0011101000001001;  //LLI 0011 101 000 010 000   r(5) = 9
rom[5] = 16'b0011101000001011;  //LLI 0011 101 000 010 000   r(5) = 11
rom[6] = 16'b1000100011000011; //BEQ 1000 101 011 010010
rom[7] = 16'b0011101000001101;  //LLI 0011 101 000 010 000   r(5) = 12
rom[8] = 16'b0011101000001111;  //LLI 0011 101 000 010 000   r(5) = 15
rom[9] = 16'b0011101000001101; //LLI 0011 101 000 010 000   r(5) = 13
rom[10]= 16'b1000100011000011; //BEQ 1000 101 011 010010
rom[11] = 16'b0011101000001100; //LLI 0011 101 000 010 000   r(5) = 12
rom[12] = 16'b0011101000001111; //LLI 0011 101 000 010 000   r(5) = 15
rom[13] = 16'b0011101000000001; //LLI 0011 101 000 010 000   r(5) = 1
rom[14] = 16'b0011101000001101; //LLI 0011 101 000 010 000   r(5) = 13
rom[15] = 16'b0011101000001111; //LLI 0011 101 000 010 000   r(5) = 15
rom[16] = 16'b0011101000001101; //LLI 0011 101 000 010 000   r(5) = 13
rom[17] = 16'b1011010111110001;  // JAL 1011 010 000000010
end
```

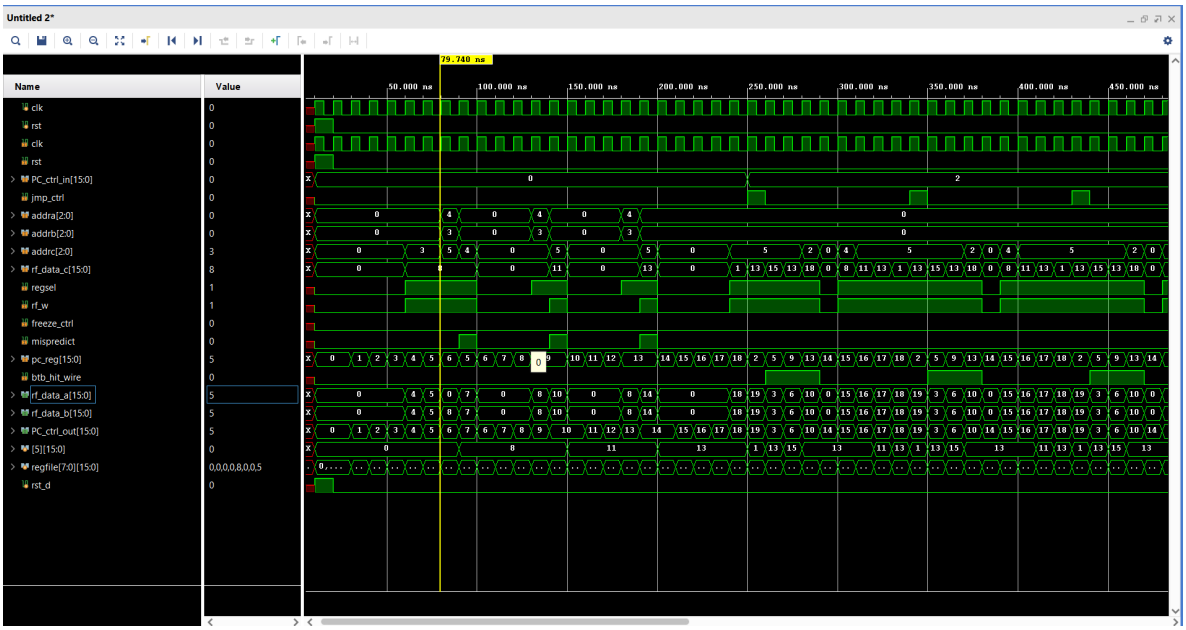

Figure 4: Output Waveform of Register File

6 Conclusion

Thus, the 6 stage Pipelined RISC Processor is designed and the various instructions are verified successfully. Data Fowarding for dependent instructions are implemented. A two bit Branch Predictor with Branch History Table (BHT) is also implemented to reduce the cycle penalties caused by branch instructions.