Computer Architecture Final Project Report

侯奕安 b11202014

December 23, 2024

1 Architecture

1.1 Datapath Design

My overall architecture is basically the same as the one in the textbook. The biggest difference is the newly-added mulDiv module and the relating control signals.

Most of the single-cycle instructions were mentioned in class, so I focus on four instructions that were not: jal, jalr, auipc, and lui.

1. jal: The control signals are

   - Jump = 1
   - RegWrite = 1
   - ALUSrc = 1

   while others are set to 0. The PC address is set as PC_nxt = PC + imm (using the J-type immediate format), and the write-back data is rd_data = PC + 4 (saved return address).

2. jalr: The control signals are

   - Jump = 1
   - RegWrite = 1
   - ALUSrc = 1

   while others are set to 0. The PC address is set as PC_nxt = (rs1_data + imm) & ~1 (LSB set to 0 for alignment), and the write-back data is rd_data = PC + 4 (saved return address).

3. auipc: The control signals are

- RegWrite = 1
- ALUSrc = 1

while others are set to 0. The ALU inputs for auipc are quite different. Input1 is the value of PC, while Input2 is the immediate. The write-back data is rd_data = PC + imm.

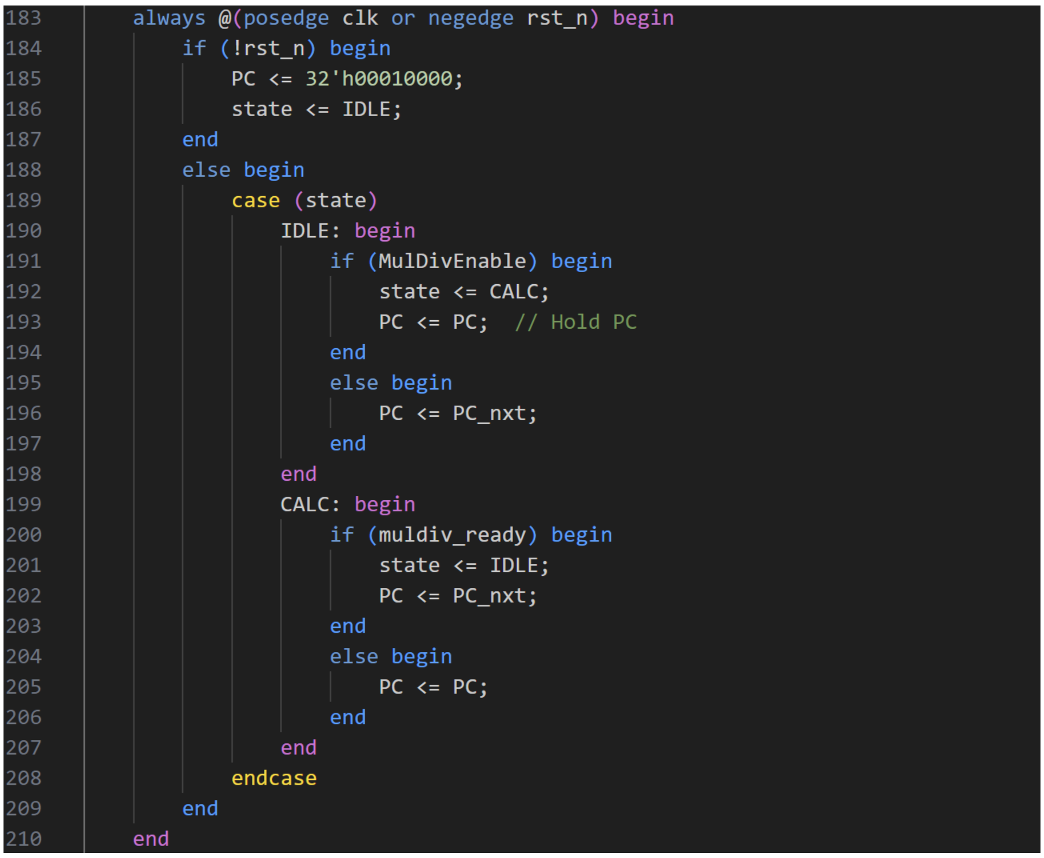
4. lui: The control signals are

    - RegWrite = 1
    - ALUSrc = 1

    while others are set to 0. The ALU inputs are different from other instructions too. Input1 is 0, and Input2 is the immediate. The rd_data is just the shifted immediate.

1.2 Multi-Cycle Instructions Handling

For these instructions, I added a new control signal ”mode”, labeling mul, divu, remu as 2’b01, 2’b10, 2’b11 respectively. Meanwhile, for all three instructions, RegWrite is set to 1, MulDivEnable is set to 1, and others are set to 0.
The way I handle multi-cycle instructions is demonstrated in the figure below. When there is a need to run a multi-cycle instruction, the control unit will raise the signal ”MulDivEnable” to 1, causing the CPU to enter state ” CALC”. During this state, the CPU is stalled, and the PC is kept at the same value. Meanwhile, the module ”mulDiv” is doing the calculation based on the value of ”mode” and will finish after 32 cycles.



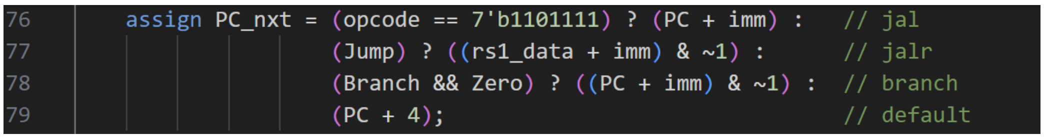
```verilog
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            PC <= 32'h00010000;
            state <= IDLE;
        end
        else begin
            case (state)
                IDLE: begin
                    if (MulDivEnable) begin
                        state <= CALC;
                        PC <= PC;  // Hold PC
                    end
                    else begin
                        PC <= PC_nxt;
                    end
                end
                CALC: begin
                    if (muldiv_ready) begin
                        state <= IDLE;
                        PC <= PC_nxt;
                    end
                    else begin
                        PC <= PC;
                    end
                end
            endcase
        end
    end
```

Figure 1: code of the design of finite state machine

After the calculation is completed, the ”muldiv” signal is raised to 1, and CPU enters the state ”IDLE” and PC is allowed to move forward.

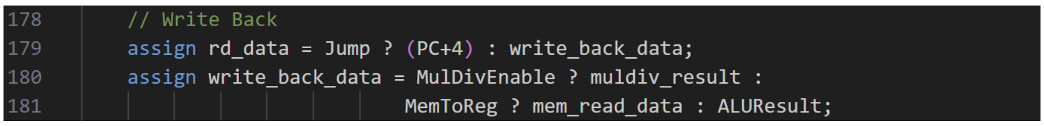
1.3 Some Logic Update

Due to some added instructions, the PC updating logic and the write-back data logic are slightly different. The changes are shown below.



```verilog
    assign PC_nxt = (opcode == 7'b1101111) ? (PC + imm) :   // jal
                    (Jump) ? ((rs1_data + imm) & ~1) :      // jalr
                    (Branch && Zero) ? ((PC + imm) & ~1) :  // branch
                    (PC + 4);                               // default
```

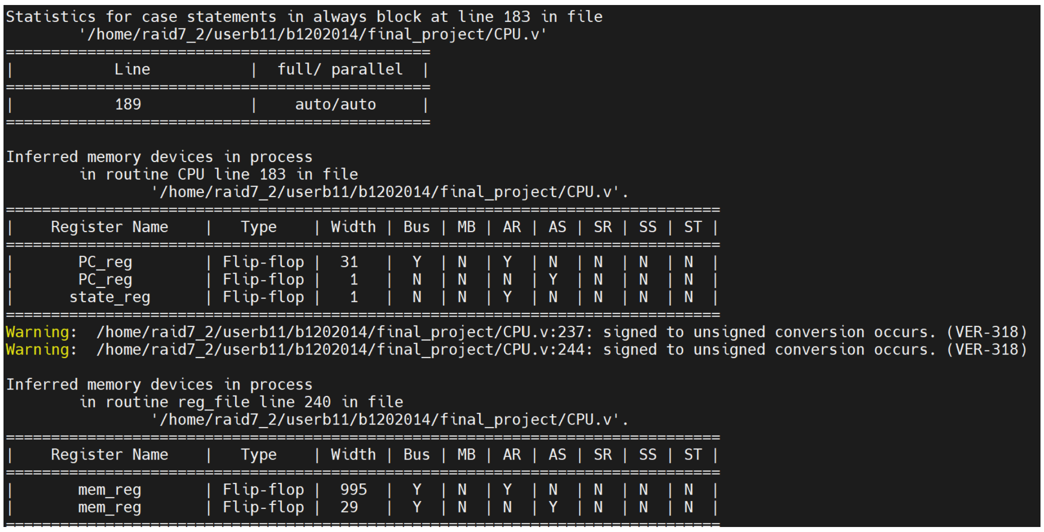
Figure 2: PC Updating Logic



```
    // Write Back
    assign rd_data = Jump ? (PC+4) : write_back_data;
    assign write_back_data = MulDivEnable ? muldiv_result :
                                  MemToReg ? mem_read_data : ALUResult;
```

Figure 3: Write-Back Logic

2 Register Table

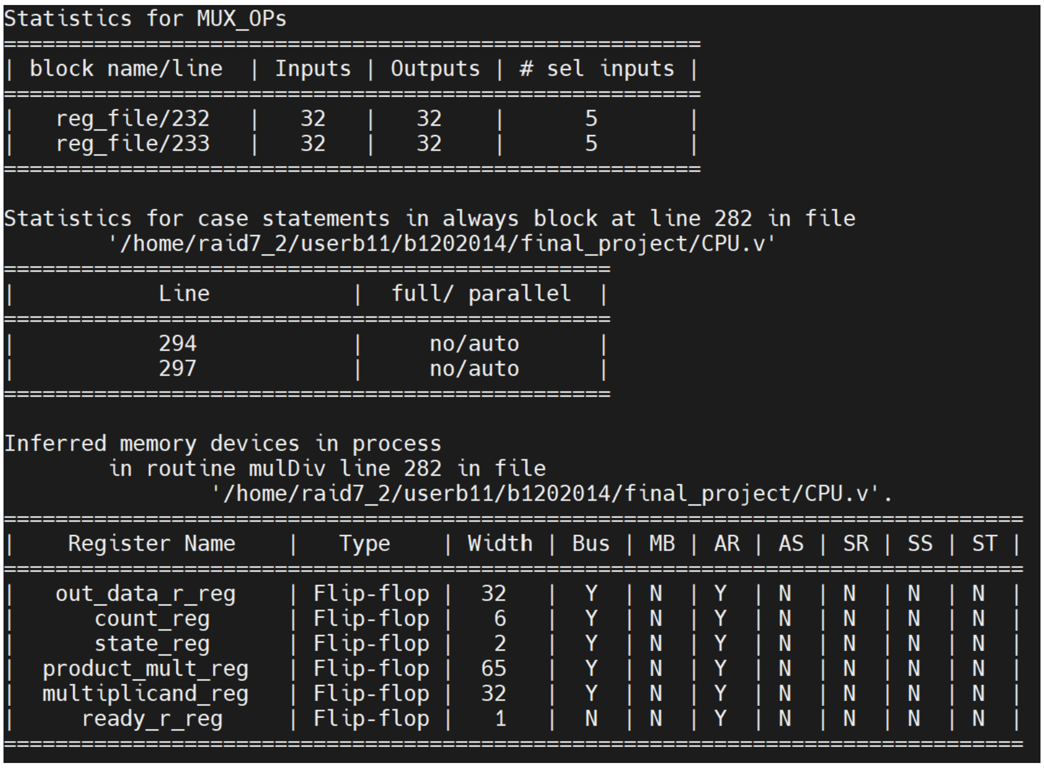


```
Statistics for case statements in always block at line 183 in file
        '/home/raid7_2/userb11/b1202014/final_project/CPU.v'
===============================================
|           Line           |  full/ parallel  |
===============================================
|           189            |     auto/auto    |
===============================================

Inferred memory devices in process
        in routine CPU line 183 in file
                '/home/raid7_2/userb11/b1202014/final_project/CPU.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
|      state_reg      | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
===============================================================================
Warning:  /home/raid7_2/userb11/b1202014/final_project/CPU.v:237: signed to unsigned conversion occurs. (VER-318)
Warning:  /home/raid7_2/userb11/b1202014/final_project/CPU.v:244: signed to unsigned conversion occurs. (VER-318)

Inferred memory devices in process
        in routine reg_file line 240 in file
                '/home/raid7_2/userb11/b1202014/final_project/CPU.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|       mem_reg       | Flip-flop |  995  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem_reg       | Flip-flop |  29   |  Y  | N  | N  | Y  | N  | N  | N  |
===============================================================================
```

Figure 4: Register Table 1



```
Statistics for MUX_OPs
======================================================
| block name/line   | Inputs | Outputs | # sel inputs |
======================================================
|   reg_file/232    |   32   |   32    |      5       |
|   reg_file/233    |   32   |   32    |      5       |
======================================================

Statistics for case statements in always block at line 282 in file
        '/home/raid7_2/userb11/b1202014/final_project/CPU.v'
===============================================
|           Line           |  full/ parallel  |
===============================================
|           294            |     no/auto      |
|           297            |     no/auto      |
===============================================

Inferred memory devices in process
        in routine mulDiv line 282 in file
                '/home/raid7_2/userb11/b1202014/final_project/CPU.v'.
===============================================================================
|    Register Name      |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|   out_data_r_reg      | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|      count_reg        | Flip-flop |   6   |  Y  | N  | Y  | N  | N  | N  | N  |
|      state_reg        | Flip-flop |   2   |  Y  | N  | Y  | N  | N  | N  | N  |
|  product_mult_reg     | Flip-flop |  65   |  Y  | N  | Y  | N  | N  | N  | N  |
|  multiplicand_reg     | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|     ready_r_reg       | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
===============================================================================
```

Figure 5: Register Table 2

3 Obversation

During my implementation, I encountered several challenges:

1. Control Signal Interactions: The ALUOp = 2'b10 for the branch instructions caused subtle bugs because it conflicted with R-type operations.

2. Multi-Cycle Instructions: The mulDiv operations required careful state management to prevent PC from advancing before computation finished.

3. Jump Instructions: Initially missed that jalr needs to clear the LSB of the target address (& ~1).

4. Architecture Designing: I decided to modulize every part of CPU, make my code look cleaner and easier for debugging.