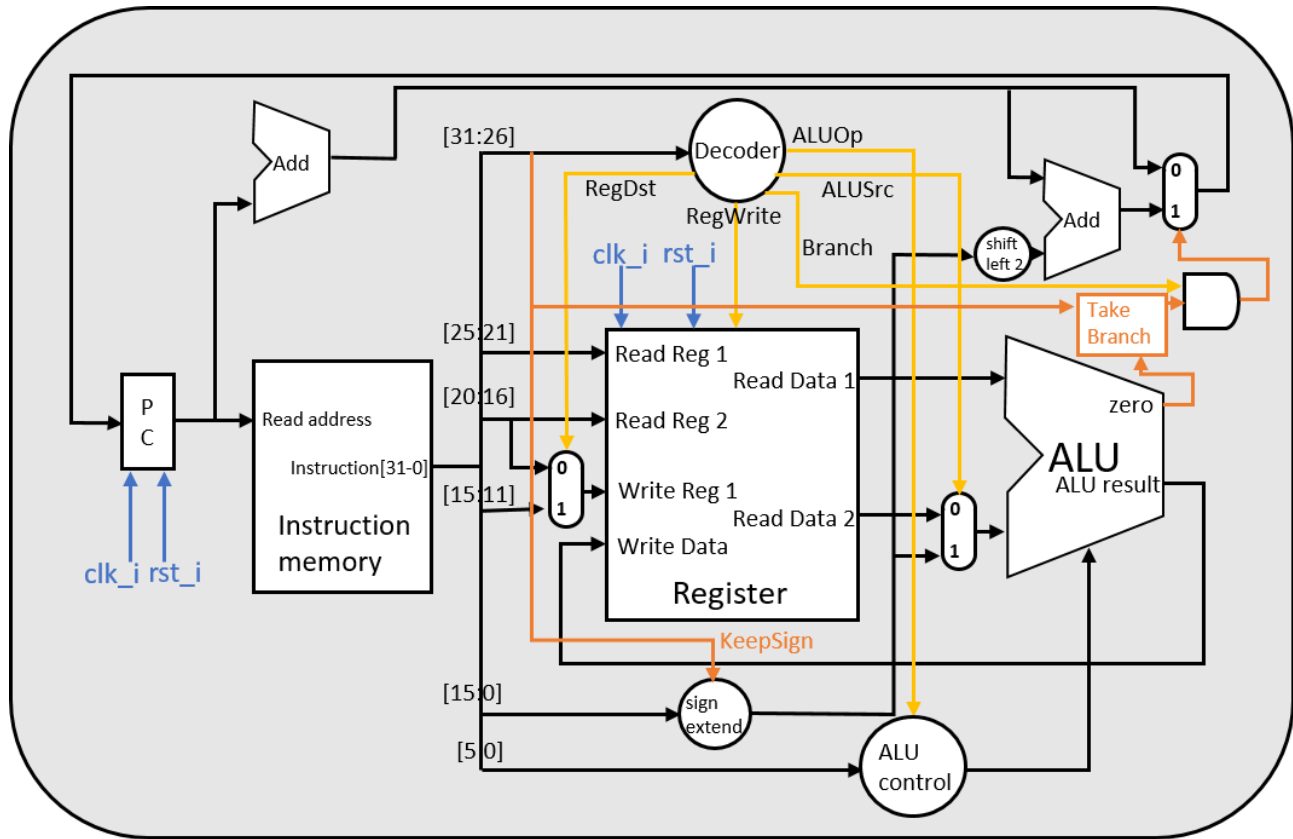


Computer Organization Lab 1

0113110 Po-Han Chen, 0316213 Yu-Wen Pwu

Architecture Diagram



Top module: Simple_Single_CPU

Detailed Description of the Implementation

- ALU Operations
 - 0: `src1_i & src2_i`
 - 1: `src1_i | src2_i`
 - 2: `src1_i + src2_i`
 - 6: `src1_i - src2_i`
 - 7: `src1_i < src2_i`
 - 12: `~(src1_i | src2_i)`
 - 14: `src2_i_signed >>> src1_i_signed`
 - 15: `src2_i << 16`

- ALU Control (input [5:0] funct_i)
Case (funct_i)
 - 3: ALUCtrl_o = 14; // SRA
 - 7: ALUCtrl_o = 14; // SRAV
 - 32: ALUCtrl_o = 2; // ADD
 - 34: ALUCtrl_o = 6; // SUB
 - 36: ALUCtrl_o = 0; // AND
 - 37: ALUCtrl_o = 1; // OR
 - 42: ALUCtrl_o = 7; // SLT
- Decoder Output Signals (input [4:0] instr_op_i;)
 - *RegWrite* = 1, if instr_op_i ∈ {0, 35, 8, 9, 15, 13}
R-type, Load, ADDI, SLTIU, LUI, ORI
-> need to write result to some register
 - *ALU_op* =
(instr_op_i == 35 || instr_op_i == 43) ? 2 : // Load or Store -> add
(instr_op_i == 4 ? 6 : // Branch -> sub
(instr_op_i == 8 ? 2 : // ADDI -> add
(instr_op_i == 9 ? 7 : // SLTIU -> lt
(instr_op_i == 13 ? 1 : // ORI -> or
(instr_op_i == 15 ? 15 : // LUI -> Left shift immediate by 16
4'b1111))))); // Else -> check with funct in instruction
 - *ALUSrc* = 1, if instr_op_i ∈ {35, 43, 8, 9, 15, 13}
Load, Store, ADDI, SLTIU, LUI, ORI
-> multiplexer selects [15:0] as the second ALU source
 - *RegDst* = 1, if instr_op_i == 0
R-type
-> multiplexer selects [15:11] as the register to write to
 - *Branch* = 1, if instr_op_i ∈ {4, 5}
BEQ, BNE
- Simple_Single_CPU
 - *keep_sign* (*KeepSign*) = 1, if inst[31:26] ∈ {9, 13}
Keep sign (Zero-extend) if op is SLTIU, ORI
 - Assign shamt ([10:6]) as the first ALU source if op is SRA

- *TakeBranch* =
BEQ -> Zero
BNE -> !Zero

Problems Encountered and Solutions

- Errors related to timing (eg. results come after 1 clock cycle later)
Solution: Change all nonblocking assignments to blocking assignments.
- General debugging difficulties
Debugging is a pain throughout the implementation.
Solution: Declare an always block and use \$display to see the contents in the variables during simulation (evaluation).

```
always @(*) begin

    $display("%b", inst);

    // $display("addr_nxt1 = %d, addr_nxt2 = %d, addr = %d, addr_nxt = %d", addr_nxt1,
    addr_nxt2, addr, addr_nxt);

    $display("%b", RegWrite);

    // $display("%b %b", RegRead1, reg_read1);

    $display("%d %d => %d ", reg_data1, reg_data2, res_alu);

end
```

Lesson Learned (If Any)

- Debugging is surprisingly difficult with so many modules in 1 place.
Next time it's probably wise to test individual components first, but even that would be kind of troublesome.
- Read the samples first to make sure that I truly understand the MIPS operations.