計算機結構 Lab1 Report

許博翔

November 7, 2023

1 Modules Explanation

1.1 `Control` Module

`Control` module reads the 0..6-th bits of the instruction as input, and outputs `ALUOp`, `ALUSrc`, `RegWrite`. The value of `RegWrite` is always 1 (as every instruction in this homework needs to write register); the value of `ALUSrc` is 1 for `addi`, `srai`, and 0 otherwise, which is calculated by looking at the 5-th bit of the instruction; the value of `ALUOp` is the $4, 5$-th bits of the instruction.

1.2 `ALU_Control` Module

`ALU_Control` module reads the $12..14, 25..31$-th bits of the instruction, `ALUOp` as input, and outputs `ALUOp3`, which will be used by `ALU` later. The module uses `ALUOp` to determine if the instruction is addi or srai, and then uses those 10 bits of the instruction to determine what kind of instruction it is. The value of `ALUOp3` is as follow:
`0:and 1:xor 2:sll 3:add 4:sub 5:mul 6:srai`.

1.3 `Sign_Extend` Module

`Sign_Extend` module reads the 20..31-th bits of the instruction, and outputs `res` (which is named `ext` in `CPU` module). The module sees those 12 bits of the instruction as a sign integer (where the most significant bit is the sign bit), and extends this integer to 32 bits by filling the 12..31-th bits of `res` with the sign bit.

1.4 ALU Module

ALU module reads a, b, op (which is named rs1, rs3, ALUOp3 in CPU module) as input, and outputs c (which is named ALUres in CPU module), Zero. The module uses ALUOp3 to determine which operation should be done to a, b, and then calculates the result as c. The value of Zero is always 0 (as the modules in this homework don't need the value of Zero).

1.5 Adder Module

Adder module reads a, b as input, and outputs a+b as c.

1.6 MUX32 Module

MUX32 module reads a, b, sel as input, and outputs c. The value of c is a if sel is 0, and is b otherwise.

1.7 CPU Module

CPU module reads clk_i, rst_i as input, and does not output anything. The following are the things that is does:

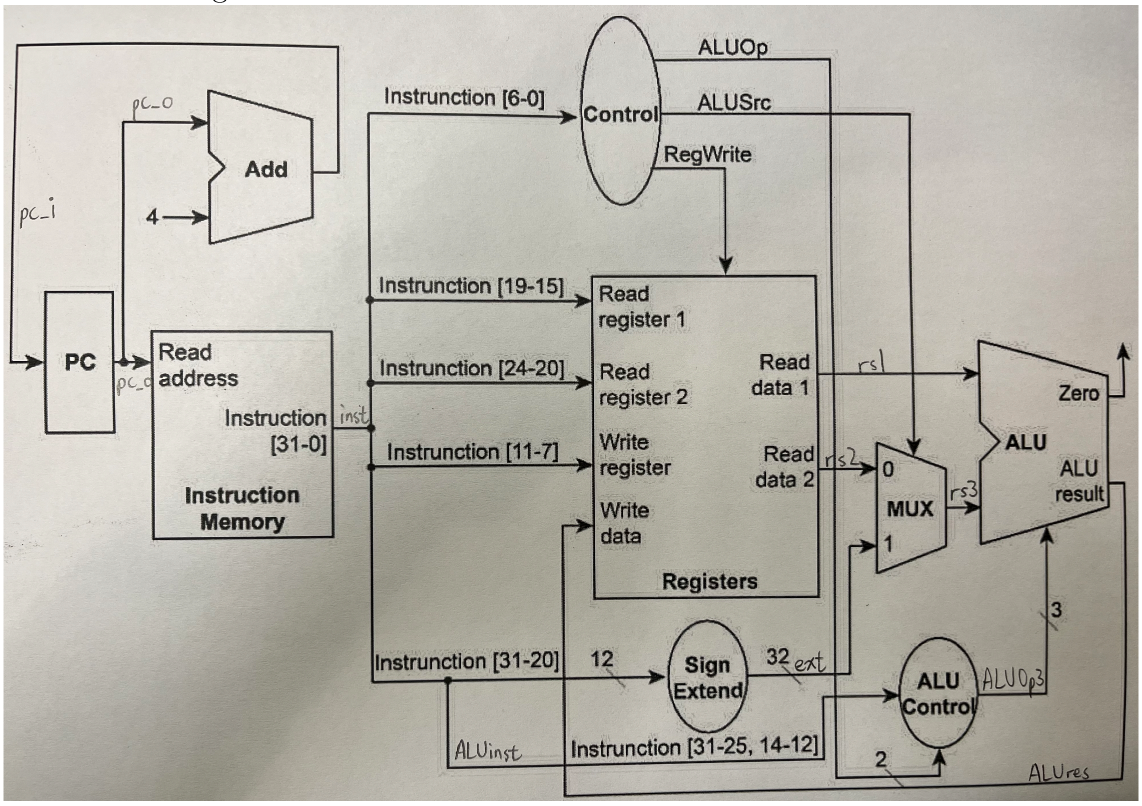
1. It inputs clk_i, rst_i, pc_i to PC module, and gets the output as pc_o.
2. It inputs pc_o to Adder module, and gets the output as pc_i.
3. It inputs pc_o to Intruction_Memory module, and gets the output as inst.
4. It inputs inst[6:0] to Control module, and gets the output as ALUOp, ALUSrc, RegWrite.
5. It inputs rst_i, clk_i, inst[19:15], inst[24:20], inst[11:7], ALUres, RegWrite to Registers module, and gets the output as rs1, rs2.
6. It merges inst[31:25], inst[14:12] as ALUinst, inputs ALUinst, ALUOp to ALU_Control module, and gets the output as ALUOp3.

7. It inputs `inst`[31:20] to `Sign_Extend` module, and gets the output as `ext`.

8. It inputs `rs2`, `ext` to `MUX32` module, and gets the output as `rs3`.

9. It inputs `rs1`, `rs3`, `ALUOp3` to `ALU`, and gets the output as `ALUres`, `Zero`.

The above things can be illustrated as follow:



2 Development Environment

- OS: Ubuntu 22.04.2 LTS
- Compiler: iverilog