HW3 - single cycle CPU Report

卓冠宇 B04901126

Hierarchy

```
b04901126_hw3
|     b04901126_hw3_report.pdf
└──── codes
    |    CPU.v
    |    ALU_Control.v
    |    Control.v
    |    ALU.v
    |    Adder.v
    |    MUX32.v
    |    Sign_Extend.v
    └────
```

Modules Explaination

CPU.v

CPU takes clock, reset and start signals as input. In CPU, we set up several wire to link each modules together for testbench.v to use.

An instruction is broken down into parts like `rd_addr, rs1_addr, rs2_addr, immediate, func73`, and they are linked into register memory, sign extender, ALU control., control. In general, `CPU.v` is for wiring.

ALU.v

ALU takes in two operands, one from Read data 1, and the other is from `MUX32`, to do the arithmetic. `ALU control` would send a signal to tell `ALU` which operation to execute.

There are two outputs, one is `ALU result` and the other is `Zero`, which gives `1` if the `ALU result` is `32'b0`.

ALU_Control.v

ALU control takes in `{funct7|funct3}` and the control signal `ALUOp_i` from `Control`. With these 2 signal, ALU control is able to tell `ALU` which operation to do by sending `ALUCtrl_o`.

The following is the table for the correspondence between `ALUOp_i` and `ALUCtrl_o`.

Input `signal` is chosen to be `{{funct7|funct3}[8], {funct7|funct3}[3:0], ALUOp_i[0]}`.

Signal	ALUCtrl_o	Operation
6'b001001	3'b000	and
6'b001111	3'b110	xor
6'b000011	3'b101	sll
6'b000001	3'b001	add
6'b100001	3'b111	sub
6'b010001	3'b011	mul
6'b101010	3'b100	srai
6'b100001	3'b111	sub
6b'??0000	3'b010	addi

Control.v

`Control` takes in opcode, ie. instruction[6:0], and produce `RegWrite_o` to tell the register memory if any change were to be written, `ALUOp_o` to the `ALU_Control`, `ALUSrc_o` to tell `MUX32` which source of the ALU operand should be.

If `Op_i` changes, we choose:

- `ALUOp_o` to be `Op_i[6:5]`, `01` for R-format and `00` for I-format.
- `ALUSrc_o` to be `~Op_i[5]`
- `RegWrite_o` is set to be `1'b1` for all instructions.

Adder.v

If any of the input `data1_in` or `data2_in` changes, `data_o` would be recomputed as `data1_in + data2_in`.

MUX32.v

`MUX32` has three inputs: `select_i`, `data1_i`, `data2_i`. If any of them changes, `MUX` would read the value of `select_i`, if it's `1'b1` then `data2_i` would be the output, if it's `1'b0`, `data1_i` would be the output.

Sign_Extend.v

`Sign_Extend` takes in `data_i`, which is `instruction[31:20]`. If `data_i` changes, `data_i[11]` would be duplicated `20` times, and concatenated with `data_i` as `data_o`.