HW#3 report

B09901089 黃柏穎

Adder (for PC):

Just simply assign sum of two inputs to output wire.

ALU\_Control:

My implementation first defined the operations (ex. AND) by their correspondence alu opcode in the beginning of the module. Then separate the always part into two parts by ALU\_op, 2’b00 for I-type instructions and 2’b10 for R-type instructions. Then further determine ALUCtrl depends on funct\_i, which is funct7+funct3.

ALU:

The beginning of this module are the definition of alu\_controls copied from ALU\_Control. Then just implement all operations corresponding to alu\_controls and switched to one of them by ALU\_control. Note that the instruction structure of SRAI operation is slightly different from other known operation, thus needs to extract the [4:0] bits from data2\_i because data2\_i here is defaultly a 32-bits number.

Control:

The code in this module sets ALUOp to 10, ALUSrc to 0, RegWrite to 1 if the 7-bits Op\_i region of original instruction equals to 0110011; whereas set ALUOp to 00, ALUSrc to 1 and RegWrite to 1 if Op\_i equals to 0010011.

MUX32:

This module connects the wire to ALU’s second input to registor2’s data if select\_i==1, otherwise connect it to Immediate from Sign\_Extend module, by a ternary operator.

Sign\_Extend:

My implementation of this module is replicate 20 0s and attach to the data\_i, if the MSB of data\_i is 0; otherwise replicate 20 1s and then do the same operation.

CPU:

In this module I just connect all modules following all data paths, and use two value, PCValueOld and PCValueNew, to keep tract of program counter, only update the PCvalue at rising edge of clock.