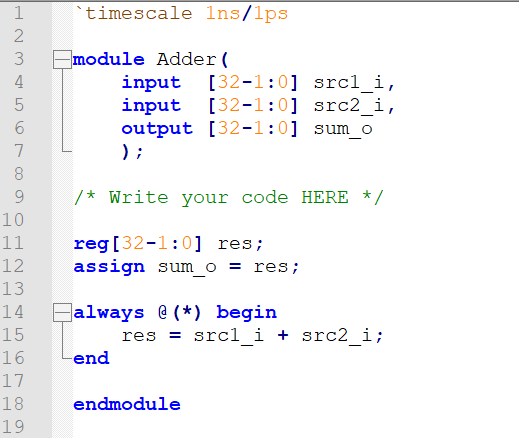
**LAB 3 : Single-Cycle CPU (Simple version)**

**109550135 范恩宇**

**Part I. Detailed description of the implementation :**

1. Adder.v :

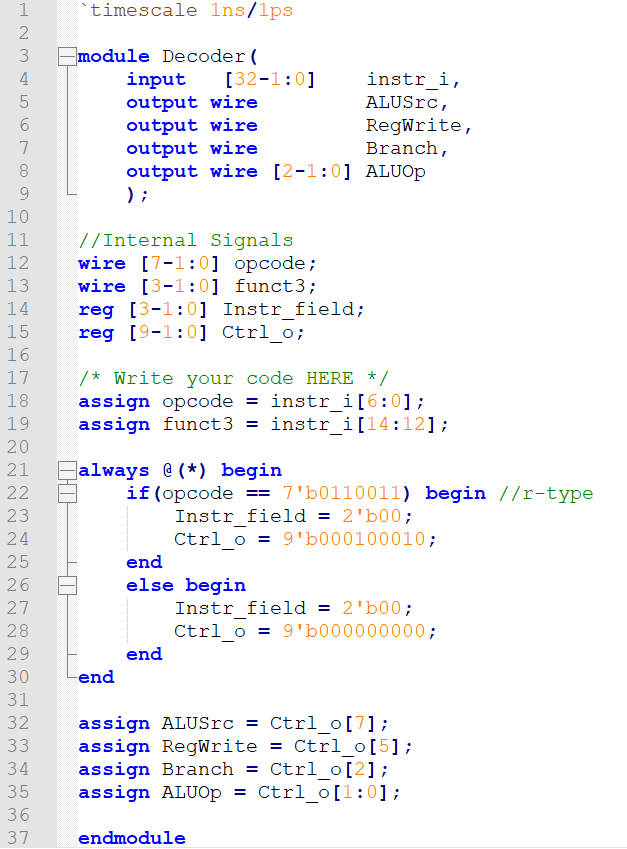
Declare a reg type “res”, which will be the sum of “src1\_i” and “src12\_i” , then “res”will be assign to “sum\_o”in the end .



1. Decoder.v :

First assign the corresponding part in instruction to “opcode”and “funct3”. Then if the opcode represents R-type commands , make “Instr\_field”be 2'b00 and “Ctrl\_o”be 9'b000100010 . Otherwise , those two will be 2'b00 and 9'b000000000 repectively .

Finally , assign the corresponding parts in“Ctrl\_o”to the four required outputs , “ALUSrc”,“RegWrite”,“Branch”, and“ALUOp” .



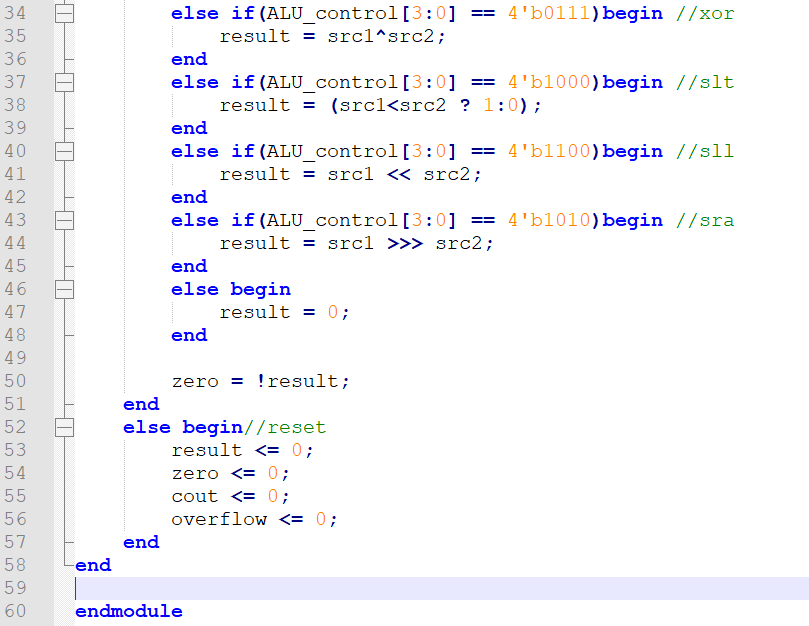
1. ALU.v :

If the the program is reset , then the result , zero flag , carryout , and overflow will all equal to 0 . Otherwise , do corresponding calculations as the slide asked and get the result accordingly , depending on what ALU\_control value we get .

As for the part of addition and subtraction , we need to consider the possibility of overflow , so I check“src1[31]”, “src2[31]”, and “result[31]”. If the first two are equal but the first one and the last one are not equal , it means that overflow happens , making“overflow”be 1 .

Finally , also in the non-reset part , value of zeroflag will be“!result”.

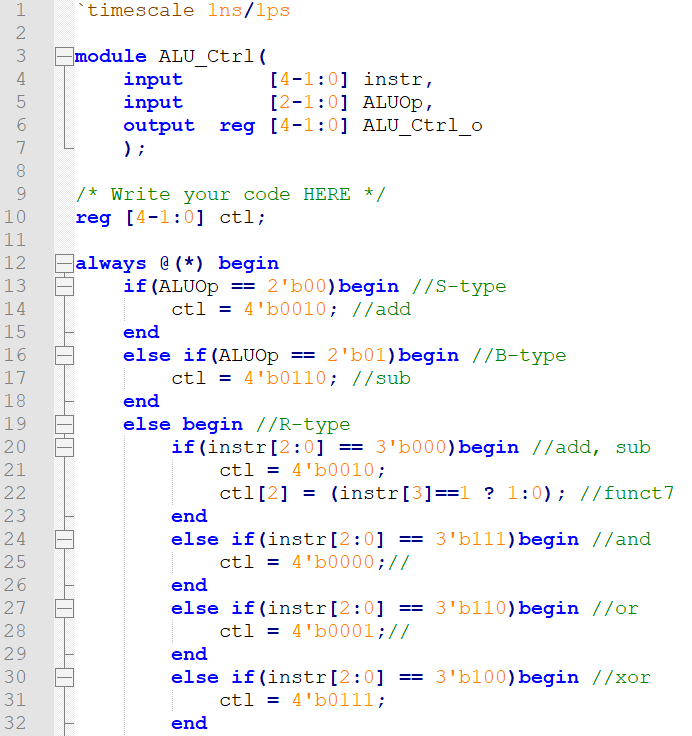


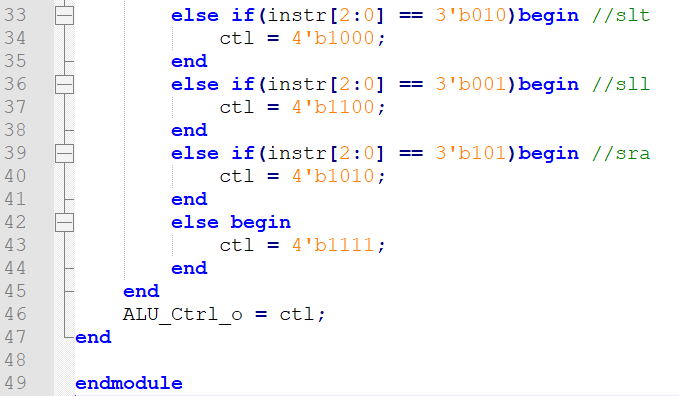


1. ALU\_Ctrl.v :

First , make“ALUOp”do the choosing . If it is S-type , then “ALU\_Ctrl\_o”will do“add”. If it is B-type , then “ALU\_Ctrl\_o”will do“sub”. Otherwise , it will be R-type .

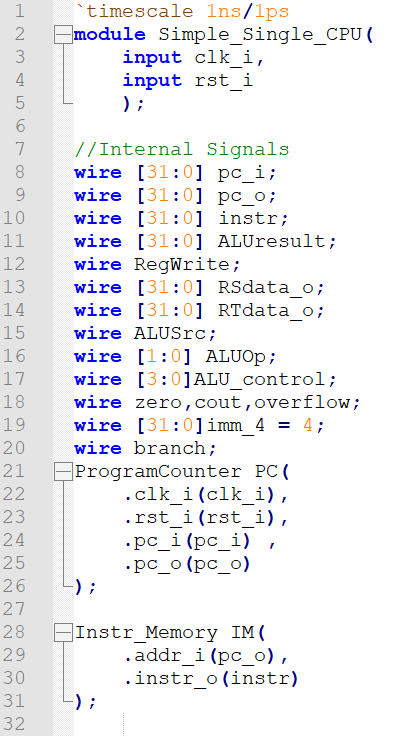
Second , in the R-type part , make“instr[2:0]”do the choosing . When“instr[2:0]”be 3'b000 , since (I30+fun3)[3] equal to alu\_control[2] and (I30 + fun3 = fun7) , I use “ALU\_Ctrl\_o[2] = (instr[3]==1 ? 1:0)“ . For the others , according to the different“instr[2:0]”, we can get the corresponding required operation in the slide of lab3 .

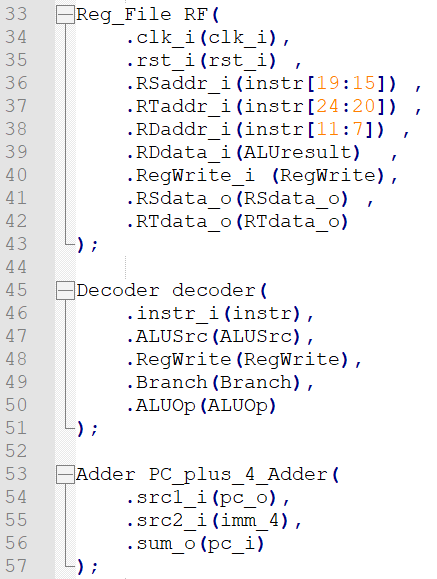


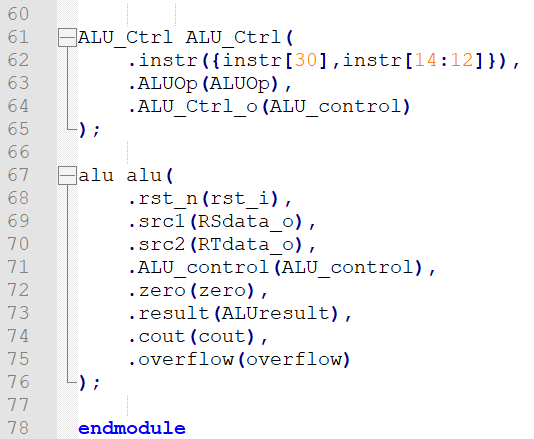


1. Simple\_Single\_CPU.v :

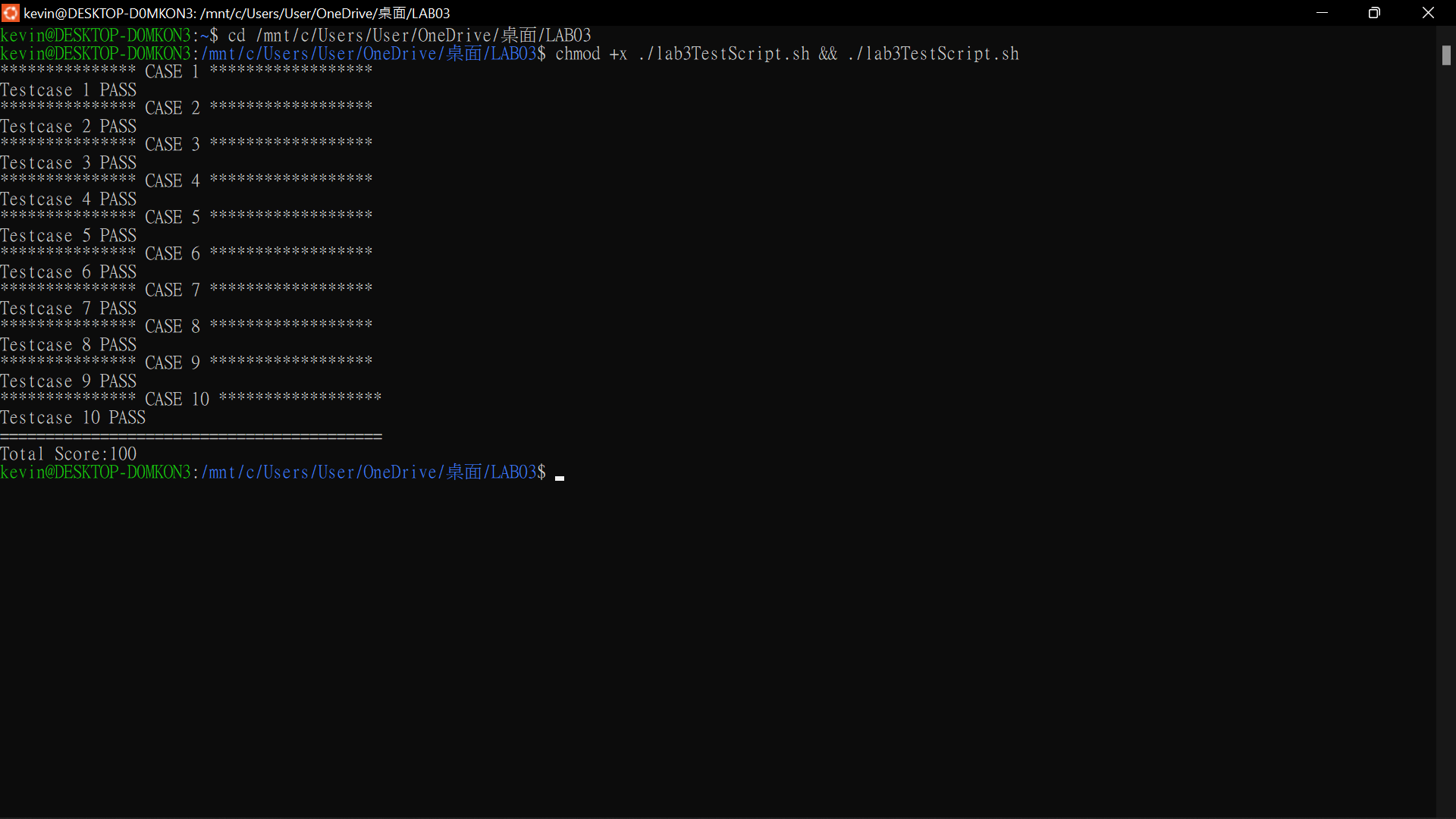
Basically consists of all the other modules except testbench , each of them is given the corresponding variables such as ALUOp or instructions .







**Part II. Implementation results :**

****

**本身電腦在最後跑Ubuntu時突然有些問題跑不了，因此借他人的電腦來跑自己的程式碼**

**Part III. Problems encountered and solutions :**

Actually , I consider“Simple\_Single\_CPU.v”to be the most challenging part in this lab . I think that alu , alu control , and decoder are not difficult to implement once understanding what ALU control value , ALUOP value , and instruction value denote a certain operation . However , the structure of the full cpu is much more complicated than I think in the beginning . As for the solution , it is simply just spend some time figuring out what input and output singnals does the specific module require . After trying different I/O for a few turns , it eventually came up with a desired result .