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   `timescale 1ns/1ps
2
3   module Adder(
4       input  [32-1:0] src1_i,
5       input  [32-1:0] src2_i,
6       output [32-1:0] sum_o
7       );
8
9   /* Write your code HERE */
10
11  reg[32-1:0] res;
12  assign sum_o = res;
13
14  always @(*) begin
15      res = src1_i + src2_i;
16  end
17
18  endmodule
19
```

2. 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  `timescale 1ns/1ps
2
3  module Decoder(
4      input   [32-1:0]    instr_i,
5      output wire         ALUSrc,
6      output wire         RegWrite,
7      output wire         Branch,
8      output wire [2-1:0] ALUOp
9      );
10
11 //Internal Signals
12 wire [7-1:0] opcode;
13 wire [3-1:0] funct3;
14 reg [3-1:0] Instr_field;
15 reg [9-1:0] Ctrl_o;
16
17 /* Write your code HERE */
18 assign opcode = instr_i[6:0];
19 assign funct3 = instr_i[14:12];
20
21 always @(*) begin
22     if(opcode == 7'b0110011) begin //r-type
23         Instr_field = 2'b00;
24         Ctrl_o = 9'b000100010;
25     end
26     else begin
27         Instr_field = 2'b00;
28         Ctrl_o = 9'b000000000;
29     end
30 end
31
32 assign ALUSrc = Ctrl_o[7];
33 assign RegWrite = Ctrl_o[5];
34 assign Branch = Ctrl_o[2];
35 assign ALUOp = Ctrl_o[1:0];
36
37 endmodule
```

3. 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
2   module alu(
3       input                   rst_n,          // negative reset            (input)
4       input signed [32-1:0]   src1,           // 32 bits source 1          (input)
5       input signed [32-1:0]   src2,           // 32 bits source 2          (input)
6       input        [ 4-1:0]   ALU_control,    // 4 bits ALU control input  (input)
7       output reg   [32-1:0]   result,         // 32 bits result            (output)
8       output reg              zero,           // 1 bit when the output is 0, zero must be set (output)
9       output reg              cout,           // 1 bit carry out           (output)
10      output reg              overflow        // 1 bit overflow            (output)
11      );
12
13   /* Write your code HERE */
14  always@(*)begin
15      if(rst_n)begin
16          if(ALU_control[3:0]==4'b0000) begin //and
17              result = src1 & src2;
18          end
19          else if(ALU_control[3:0] == 4'b0001)begin //or
20              result = src1 | src2;
21          end
22          else if(ALU_control[3:0] == 4'b0010)begin //add
23              result = src1 + src2;
24              if(src1[31] != result[31] && src1[31] == src2[31])begin
25                  overflow = 1;
26              end
27          end
28          else if(ALU_control[3:0] == 4'b0110)begin //sub
29              result = src1 - src2;
30              if(src1[31] != result[31] && src1[31] == src2[31])begin
31                  overflow = 1;
32              end
33          end
```

```
34          else if(ALU_control[3:0] == 4'b0111)begin //xor
35              result = src1^src2;
36          end
37          else if(ALU_control[3:0] == 4'b1000)begin //slt
38              result = (src1<src2 ? 1:0);
39          end
40          else if(ALU_control[3:0] == 4'b1100)begin //sll
41              result = src1 << src2;
42          end
43          else if(ALU_control[3:0] == 4'b1010)begin //sra
44              result = src1 >>> src2;
45          end
46          else begin
47              result = 0;
48          end
49
50          zero = !result;
51      end
52      else begin//reset
53          result <= 0;
54          zero <= 0;
55          cout <= 0;
56          overflow <= 0;
57      end
58  end
59
60  endmodule
```

4. 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] equals 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  `timescale 1ns/1ps
2
3  module ALU_Ctrl(
4      input       [4-1:0] instr,
5      input       [2-1:0] ALUOp,
6      output  reg [4-1:0] ALU_Ctrl_o
7      );
8
9  /* Write your code HERE */
10 reg [4-1:0] ctl;
11
12 always @(*) begin
13     if(ALUOp == 2'b00)begin //S-type
14         ctl = 4'b0010; //add
15     end
16     else if(ALUOp == 2'b01)begin //B-type
17         ctl = 4'b0110; //sub
18     end
19     else begin //R-type
20         if(instr[2:0] == 3'b000)begin //add, sub
21             ctl = 4'b0010;
22             ctl[2] = (instr[3]==1 ? 1:0); //funct7
23         end
24         else if(instr[2:0] == 3'b111)begin //and
25             ctl = 4'b0000;//
26         end
27         else if(instr[2:0] == 3'b110)begin //or
28             ctl = 4'b0001;//
29         end
30         else if(instr[2:0] == 3'b100)begin //xor
31             ctl = 4'b0111;
32         end
33         else if(instr[2:0] == 3'b010)begin //slt
34             ctl = 4'b1000;
35         end
36         else if(instr[2:0] == 3'b001)begin //sll
37             ctl = 4'b1100;
38         end
39         else if(instr[2:0] == 3'b101)begin //sra
40             ctl = 4'b1010;
41         end
42         else begin
43             ctl = 4'b1111;
44         end
45     end
46     ALU_Ctrl_o = ctl;
47 end
48
49 endmodule
```

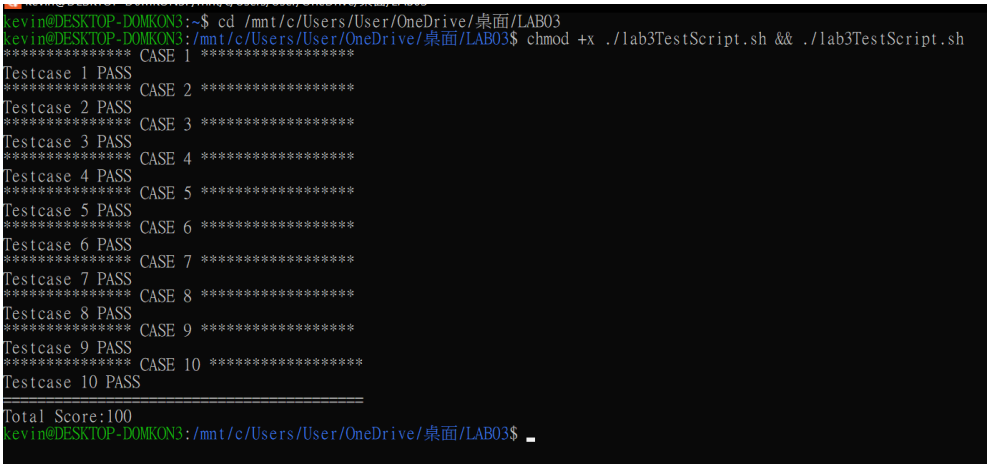
5. Simple_Single_CPU.v :

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

```
1   `timescale 1ns/1ps
2   module Simple_Single_CPU(
3       input clk_i,
4       input rst_i
5       );
6
7   //Internal Signals
8   wire [31:0] pc_i;
9   wire [31:0] pc_o;
10  wire [31:0] instr;
11  wire [31:0] ALUresult;
12  wire RegWrite;
13  wire [31:0] RSdata_o;
14  wire [31:0] RTdata_o;
15  wire ALUSrc;
16  wire [1:0] ALUOp;
17  wire [3:0]ALU_control;
18  wire zero,cout,overflow;
19  wire [31:0]imm_4 = 4;
20  wire branch;
21  ProgramCounter PC(
22      .clk_i(clk_i),
23      .rst_i(rst_i),
24      .pc_i(pc_i) ,
25      .pc_o(pc_o)
26  );
27
28  Instr_Memory IM(
29      .addr_i(pc_o),
30      .instr_o(instr)
31  );
32
```

```
33  Reg_File RF(
34      .clk_i(clk_i),
35      .rst_i(rst_i) ,
36      .RSaddr_i(instr[19:15]) ,
37      .RTaddr_i(instr[24:20]) ,
38      .RDaddr_i(instr[11:7]) ,
39      .RDdata_i(ALUresult)   ,
40      .RegWrite_i (RegWrite),
41      .RSdata_o(RSdata_o) ,
42      .RTdata_o(RTdata_o)
43  );
44
45  Decoder decoder(
46      .instr_i(instr),
47      .ALUSrc(ALUSrc),
48      .RegWrite(RegWrite),
49      .Branch(Branch),
50      .ALUOp(ALUOp)
51  );
52
53  Adder PC_plus_4_Adder(
54      .src1_i(pc_o),
55      .src2_i(imm_4),
56      .sum_o(pc_i)
57  );
60
61  ALU_Ctrl ALU_Ctrl(
62      .instr({instr[30],instr[14:12]}),
63      .ALUOp(ALUOp),
64      .ALU_Ctrl_o(ALU_control)
65  );
66
67  alu alu(
68      .rst_n(rst_i),
69      .src1(RSdata_o),
70      .src2(RTdata_o),
71      .ALU_control(ALU_control),
72      .zero(zero),
73      .result(ALUresult),
74      .cout(cout),
75      .overflow(overflow)
76  );
77
78  endmodule
```

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 .