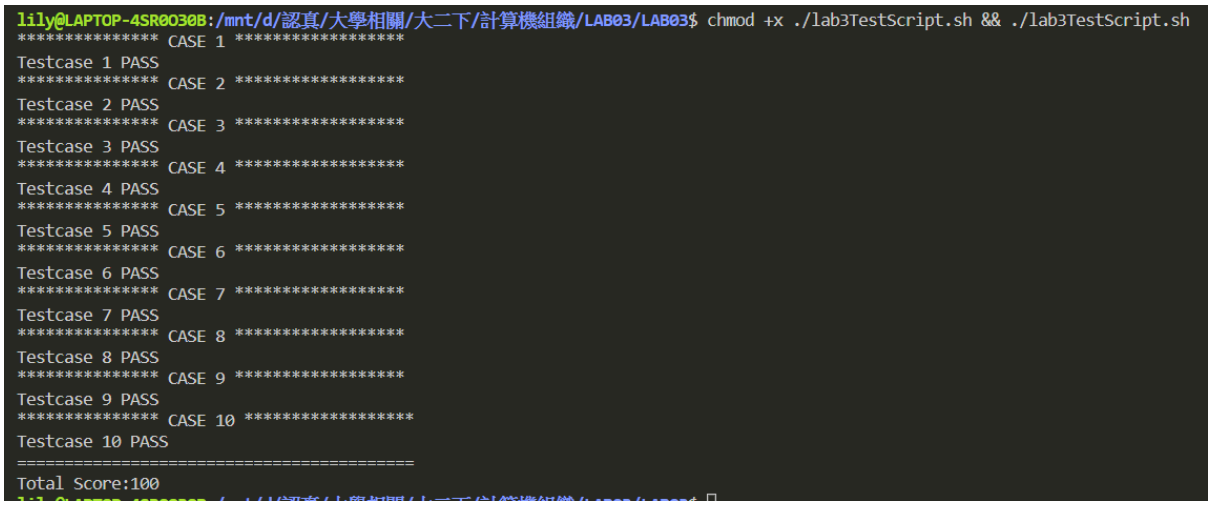
Lab03 Single cycle CPU (Simple version)

109550053楊立嘉

PART 1. Result with Ubuntu

- using WSL in VS code to do implementation



PART 2 . Implementation Detail

PC + 4 Adder

Adder.v

```v
`timescale 1ns/1ps
module Adder(
    input  [32-1:0] src1_i,
    input  [32-1:0] src2_i,
    output [32-1:0] sum_o
    );

assign sum_o = src1_i + src2_i;

endmodule
```

ALU

- It is almost same with I did in Lab 2, just add line 40~46 to do XOR, SLL and SRA using operator.

```
1   module alu(
2       input                   rst_n,         // negative reset           (input)
3       input signed [32-1:0]   src1,          // 32 bits source 1         (input)
4       input signed [32-1:0]   src2,          // 32 bits source 2         (input)
5       input        [ 4-1:0]   ALU_control,   // 4 bits ALU control input  (input)
6       output reg   [32-1:0]   result,        // 32 bits result           (output)
7       output reg              zero,          // 1 bit when the output is 0, zero must be set (output)
8       output reg              cout,          // 1 bit carry out          (output)
9       output reg              overflow       // 1 bit overflow           (output)
10      );
11
12  /* Write your code HERE */
13  //助教要求除了XOR、sll、sra可以直接用 operator以外，其他運算都要套用 lab2的程式碼
14      wire [32-1:0]    res;
15      wire             per_cout;
16      wire             slt;       //就是傳回去的set，但其實set不只有slt，我簡化成只做slt
17      wire [32-1:0]    eq;
18      wire             sign_check, add, sign;
19      wire             over;
20
21      alu32 a1(.src1(src1),
22               .src2(src2),
23               .less(slt),
24               .Ainvert(ALU_control[3]),
25               .Binvert(ALU_control[2]),   //ALU_ctrl同時代表 Binvert 跟 cin 的原因在老師PPT有教
26               .cin(ALU_control[2]),
27               .operation(ALU_control[1:0]),
28               .result(res),
29               .cout(per_cout),
30               .eq(eq),
31               .overflow(over),
32               .Sign(sign)     //sign回傳的是計算結果的正負號，也就是MSB是0還是1。Sign = eq[31] ^ over[6];
33      );
34
35       assign slt = eq[31] ? src1[31] : sign;
36       assign add = ALU_control[1] & ~ALU_control[0];       //確認是否正在做加減法
```

```
38      always @ (*) begin
39          if (rst_n) begin
40              if (ALU_control == 4'b0011)         //特判xor
41                  result <= src1 ^ src2;
42              else if(ALU_control == 4'b1110)     //特判sll
43                  result <= src1 << src2;
44              else if(ALU_control == 4'b1111)     //特判sra
45                  result <= src1 >>> src2;
46              else
47                  begin
48                      result <= res;
49                      zero <= ~|res;                      //將 result 的所有位數拿出來做 nor
50                      cout <= per_cout & add;              //由 cla adder 拿出 carry out 之後，再確認是否是作加減法
51                      overflow <= over & add;              //加了& ad，因為做加剪髮才會需要判斷overflow
52                  end
53          end
54      end
55
56  endmodule
```

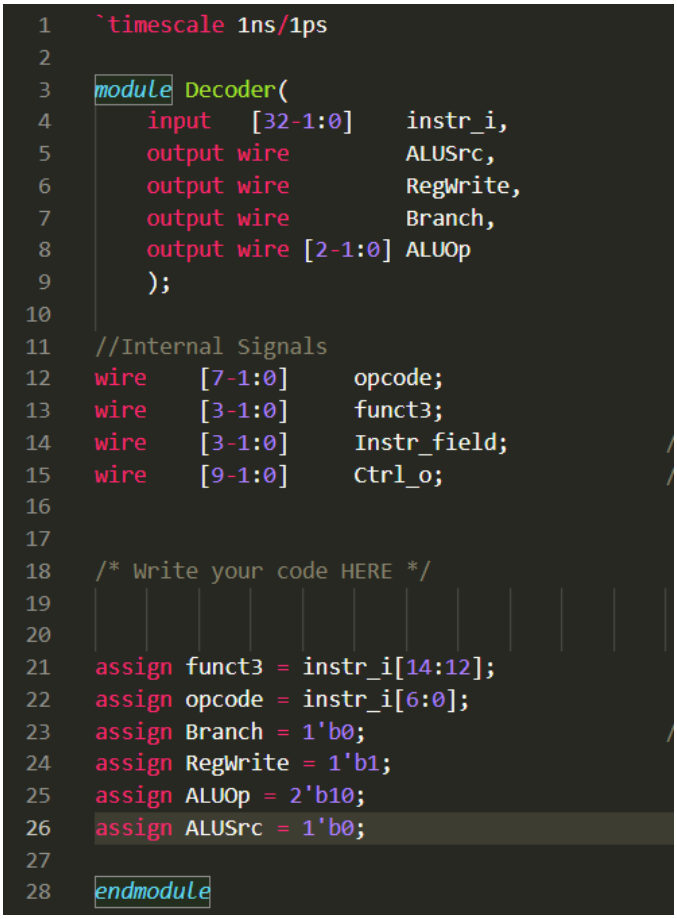
ALU control

- The `ALU_control_input` I defined for XOR, SLL and SRA are 0011, 1110 and 1111.

```
1  `timescale 1ns/1ps
2
3  module ALU_Ctrl(
4      input       [4-1:0] instr, //[30,14-12]，也就是先 func7才 func3
5      input       [2-1:0] ALUOp, //本次作業全部都`R type ，根本不會用到 ALU_op去判斷，所以之後要用再來改
6      output  reg [4-1:0] ALU_Ctrl_o
7      );
8
9  /* Write your code HERE */
10 always @(*)
11     begin
12         case(instr)
13             4'b0000: ALU_Ctrl_o = 4'b0010; //add
14             4'b1000: ALU_Ctrl_o = 4'b0110; //sub
15             4'b0111: ALU_Ctrl_o = 4'b0000; //and
16             4'b0110: ALU_Ctrl_o = 4'b0001; //or
17             4'b0010: ALU_Ctrl_o = 4'b0111; //slt
18             4'b0100: ALU_Ctrl_o = 4'b0011; //xor 我自己定義的 ALU_control_input
19             4'b0001: ALU_Ctrl_o = 4'b1110; //sll 我自己定義的 ALU_control_input
20             4'b1101: ALU_Ctrl_o = 4'b1111; //sra 我自己定義的 ALU_control_input
21
22             default: ALU_Ctrl_o = 4'b0000;
23         endcase
24     end
25
26
27
28 endmodule
29
```

Decoder

- note that I implement `Decoder.v` under the situation that this lab only need to solve R-type instruction, so I didn't use `ALUOp` to determine current instruction type, just set up every control line for R-type.



```
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 wire    [3-1:0]     Instr_field;
15 wire    [9-1:0]     Ctrl_o;
16
17
18 /* Write your code HERE */
19
20
21 assign funct3 = instr_i[14:12];
22 assign opcode = instr_i[6:0];
23 assign Branch = 1'b0;
24 assign RegWrite = 1'b1;
25 assign ALUOp = 2'b10;
26 assign ALUSrc = 1'b0;
27
28 endmodule
```

Simple Single CPU

- Just screenshot parts that we need to fill.
- note that the input of decoder `instr_i` is whole 32 bits `instr` but not 7 bits, because the input TAs gave in `Decoder.v` is 32 bits. As for the input of `ALU_Ctrl` is only 4 bits , `I[30]` and `func3`, also follow TAs' setting in `ALU_Ctrl.v`.

```
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]),     //rs1
37          .RTaddr_i(instr[24:20]),     //rs2
38          .RDaddr_i(instr[11:7]),      //rd
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          );
58
```

```
59  ALU_Ctrl ALU_Ctrl(
60          .instr({instr[30],instr[14:12]}),
61          .ALUOp(ALUOp),
62          .ALU_Ctrl_o(ALU_control)
63          );
64
65  alu alu(
66          .rst_n(rst_i),
67          .src1(RSdata_o),
68          .src2(RTdata_o),
69          .ALU_control(ALU_control),
70          .result(ALUresult),
71          .zero(zero),
72          .cout(cout),
73          .overflow(overflow)
74          );
75
76
77  endmodule
```

PART 3. Experience

How to define `ALU_control_input` for XOR, SLL and SRA confuses me for a long time, I want to implement using “normal” definition, but I google and find : some people define `ALU_control_input` just using 3 bits, some people define SRA as 1011, some 1101, just like screen shot below from google searching “SRA alu control input”.

All in all, I define this `ALU_control_input` by myself, but I still do not understand why there is not a formal definition.

ALU Control	Instruction	Module input/output
0010	addu	**module** ALU (input1, input2, alu_control, shift_amount, ALUout, zero) output [31:0] ALUout; output zero; input [31:0] input1, input2; input [3:0] alu_control; input [4:0] shift_amount; // your code here (behavioral code is OK) **endmodule**
0110	subu	
0000	and	
0001	or	
1001	xor	
1010	sll	
1011	sra	
1100	srl	
0111	slt	
1110	sltu	
xxxx	jr	

Operation	ALU Control Line
AND	0000
OR	0001
SLL	0011
SRL	0100
ADDU	1000
SUBU	1001
XOR	1010
SLTU	1011
NOR	1100
SRA	1101
LUI	1110

Finally, I think I done `Decoder.v` use tricky method, because I done everything just for R-type instruction. Hope that when needed in following lab, I can fix this quickly.