

這次有遇到幾個問題

第一個是 case 沒有寫完整，之後在 Decoder.v 檔裡面再加了一個 default case 來處理就可以了。

第二個是 write back 的 data 好像都印不出來，看了 testbench.v 裡面的寫法，應該是因為都是 0，所以都沒有 write 到 result.txt 檔裡面，檢查各個.v 檔和電路之後發現最後要 write back 的 MUX0,1 順序和其他的 decoder 相反，所以拉回去的值錯了，對調 Simple_Single_CPU.v 檔裡面那一個 MUX 的 port 就可以了。

第三個是我們發現 jal 指令做完之後並沒有 jump，而是直接跳到下一行，檢查.v 檔之後發現是在原本的寫法裡面少給 jal 的 imm 最後 1' b0，導致值是錯的。

再來是 sw 指令都寫進錯誤的 address(0)，檢查 ALU_Ctrl 之後發現，在原本的設計裡面 sw 和 slt 掉到同一個 case，所以導致 alu 算出來的值是錯的，在 ALU_Ctrl 的判斷式裡面多加上 ALU_OP 來做判斷就可以了。

第四個就是 alu 的 input 要加 signed，我們覺得可能是因為 alu 後來是直接改成用對應的 operator 去實作，所以若是 input 沒有 signed 的話他會報錯。由於第 1700 行的指令是 slt，然後好像是 0 跟 -1 比，最後結果應該要是 1，後來是在 simple_cycle 裡面，Vscode 會提示各個 port 的型態，才突然想到 alu 的 input 應該要是 signed，後來改完就對了~

執行結果：

```
kkmelon@DESKTOP-TEEM7HG:~/ComputerOrganization/LAB4$ cd ComputerOrganization/LAB4/
kkmelon@DESKTOP-TEEM7HG:~/ComputerOrganization/LAB4$ ls
ALU_Ctrl.v  Decoder.v  Instruction.txt  MUX_2to1.v  Reg_File.v  alu.v  result.txt
Adder.v     Imm_Gen.v  Lab4_Slide.pdf  Makefile    Simple_CPU.lxt  answer.txt  test
Data_Memory.v  Instr_Memory.v  Lab4-Spec.pdf  ProgramCounter.v  Simple_Single_CPU.v  demo.sh  testbench.v
kkmelon@DESKTOP-TEEM7HG:~/ComputerOrganization/LAB4$ chmod +x demo.sh
kkmelon@DESKTOP-TEEM7HG:~/ComputerOrganization/LAB4$ ./demo.sh
iverilog *.v -o Simple_CPU.vvp
vvp Simple_CPU.vvp -fst -sdf-verbose -lxt2
WARNING: Instr_Memory.v:15: $readmemb(Instruction.txt): Not enough words in the file for the requested range [0:64].
LXT2 info: dumpfile Simple_CPU.lxt opened for output.
CONGRATULATION!!
MMMMMMMMMMMMMMMMMMMMWKKOOKKOOOXMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMW0dc:::cc11111lcc:::coONMMMMMMMMMMMM
MMMMMMMMMMMMWk:::lkONNNNNNNNNNNNNNNNNKko:::dNNMMMMMM
MMMMMMMMMMO'cONNNNNNNNNNNNNNNNNNNNNKl'xWMMMMMMMM
MMMMMMMMMc'ONNNNNNNNNKZNNNNNNNNNNXNNNNNNNO::XMMMMMM
MMMMMMW;KNNNNNNNNNO.xNNNNNNNNNNl;NNNNNNNNXc'NMMMMMM
MMMMWKl;:dXNNlNNNd.KNNNNNNNNNx.KNNdkNNNxc.:KWNMM
MMXl,c1...ONNd:::,:,:,:,:,:,:cNNX'..cc;cKMM
W'o'xNNk'..NNd'WWWXXXXXXXXXXXXXXXXXc:NNNc.dNNNx'cN
O:::ONX,::NNNo,Wk:NNNNNNNNNNNNNNNNNNWl;NNNo.'ONK,;k
MMO.Ox,c.cNNNo,Wl'XNNNNNNNNNNNNNNNNNNWl;NNNx.:oK'xMM
MMX...:O.lNNNo,Wo'XNNNNNNNNNNNNNNNNNNWl'ONNO.do..OMM
MMMMNN.ox.oXNNNo,WWWXXXXXXXXXXXXXXXXXl.kKNX.lk.ONMM
MMMMO.kl.xKNNNo'WWWNNNNNNNNNNNNNNNNNNWl,kONN;K.dMM
MMMMd.K;:xoKNd'WWWNNNNNNNNNNNNNNNNNNWc,kxKo'N:MM
MMMM;N'.'.'Od.:codxxxxkkO00kxdoc;:d'...Kd.WMM
MMMM.xNkOXk'.'..kOkxd..looo'.oxkO0'...:xNOxXO.OMM
MMNx'XNNNNNO....:oxO0:.dx;kOxo:...'kNNNNNN:1MM
MMW;lNZOKOx...:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMO.cc'.ox'..:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMNOOKl'x,,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMMMW';:OkO,,:,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMMMMWxoc,.lx,,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMMMMMmk.Ok,,:,:,:,:,:,:,:,:,:,:,:,:,:,:
MMMMMMMMMX'..:,:,:,:,:,:,:,:,:,:,:,:,:,:
kkmelon@DESKTOP-TEEM7HG:~/ComputerOrganization/LAB4$
```

decoder 的筆記：

	RW	B	J	WB	MR	MW	AB	Src	Op	Code
R	1	0	0	00	0	0	0	x0	10	0110011
addi	1	0	0	00	0	0	x1	00		0010011
lw	1	0	0	01	1	0	x1	00		0000011
sw	0	0	0	xx	0	1	x1	00		0100011
beq	0	1	0	xx	0	0	00	01		1100011
jal	1	0	1	1x	0	0	0x	xx		1101111
jalr	1	0	1	1x	0	0	1x	xx		1100111

$\text{rd} = \text{PC} + 4$
 jal x1 $\xrightarrow{-40}$ set PC = PC + offset
 jalr $\text{rd} = \text{PC} + 4$ $\text{rs1} + \text{off} = \text{PC}$ beq x7 x0 16 sw x1 8(x2) rs2 rs1
 addi rd rs1 imm lw x14 8(x2) rd rs1

Implement detail:

架構圖其實就是照 slide 做的~

Imm_Gen 的 sign-extend 是直接複製幾個的方式，不太確定有沒有其他方法。

```
/* Write your code HERE */
always @* begin
    casez(opcode)
        7'b0010011, 7'b0000011, 7'b1100111: begin //addi, load
            Imm_Gen_o = {{20{instr_i[31]}}, instr_i[31:20]};
        end
        7'b0100011: begin //store
            Imm_Gen_o = {{20{instr_i[31]}}, instr_i[31:25], instr_i[11:8], instr_i[7]};
        end
        7'b1100011: begin //Branch
            Imm_Gen_o = {{19{instr_i[31]}}, instr_i[7], instr_i[30:25], instr_i[11:8], 1'b0};
        end
        7'b1101111: begin //jal, jalr
            Imm_Gen_o = {{12{instr_i[31]}}, instr_i[19:12], instr_i[20], instr_i[30:25], instr_i[24:21], 1'b0};
        end
        default: begin
            Imm_Gen_o = 0;
        end
    endcase
end

endmodule
```

ALU_control 是直接沿用上次的~

```
3 wire [6:1:0] instr_ALUOp;
4 assign instr_ALUOp = {instr, ALUOp};
5
6 // 6'b????00 : I type, S type
7 // 6'b????01 : B type
8 // 6'b????10 : R type
9 always @(*) begin
10     casez(instr_ALUOp)
11         6'b????00: ALU_Ctrl_o = 4'b0010; // ld sd, add
12         6'b????01: ALU_Ctrl_o = 4'b0110; // beq, sub
13         6'b000010: ALU_Ctrl_o = 4'b0010; // add
14         6'b100010: ALU_Ctrl_o = 4'b0110; // sub
15         6'b011110: ALU_Ctrl_o = 4'b0000; // and
16         6'b011010: ALU_Ctrl_o = 4'b0001; // or
17         6'b001010: ALU_Ctrl_o = 4'b0111; // slt
18         6'b110110: ALU_Ctrl_o = 4'b1000; // sra
19         6'b000110: ALU_Ctrl_o = 4'b1001; // sll
20         6'b010010: ALU_Ctrl_o = 4'b1010; // xor
21         default: ALU_Ctrl_o = 4'bxxxx;
22     endcase
23 end
24 endmodule
```

Decoder 是類似這樣給所有 output 值，數字是依照前面ㄉ table

```
/* Write your code HERE */
always @(*) begin
    casez(instr_i)
        7'b0110011: begin //R-type
            RegWrite = 1;
            Branch = 0;
            Jump = 0;
            WriteBack1 = 0;
            WriteBack0 = 0;
            MemRead = 0;
            MemWrite = 0;
            ALUSrcA = 1'bx;
            ALUSrcB = 0;
            ALUOp = 2'b10;
        end
        7'b0010011: begin //addi
```