0810749 張君實 Report

1. Finish the truth table for Decoder.

Instr	RW	B	J	WB[1,0]	MR	MW	Src[A,B]	Op[1,0]
R-type	1	0	0	00	0	0	X0	1X
addi	1	0	0	00	0	0	X1	00
Load	1	0	0	01	1	0	X1	00
Store	0	0	0	XX	0	1	X1	00
Branch	0	1	0	XX	0	0	00	01
JAL	1	0	1	1X	0	0	0X	XX
JALR	1	0	1	1X	0	0	1X	XX

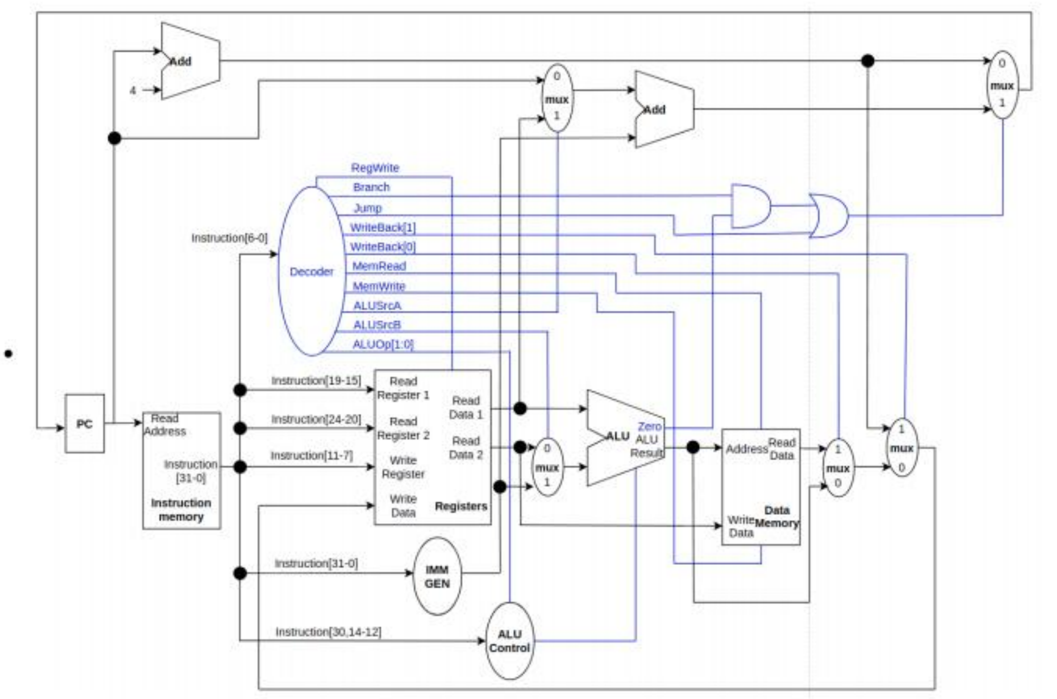
2. What's the problem you encounter?

大概是花了許多時間才把表生出來吧，我到網路上查表，有些表找不到，只好直接打開測資讀每一行指令的意思，再看他們指令的 opcode 或 func7，func3。最後確認表沒錯才開始寫 code。因為我寫完 code 後把 debug 交給另一個組員，所以才一直確認表有沒有錯。

3. Your implement detail.

以下是實作細節，助教寫好的部分就不贅述。

(1) simple_single_cpu



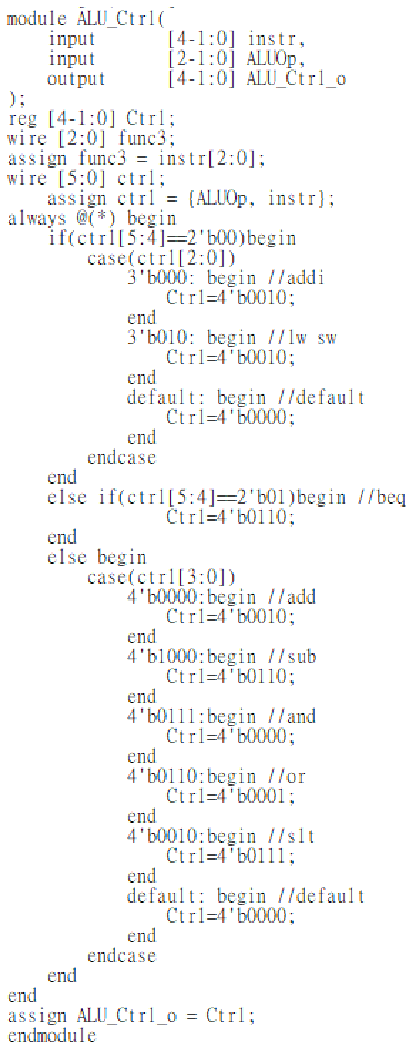
照著助教給的圖接線

(2) alu

```
module alu(
    input                   rst_n,         // negative reset            (input)
    input signed [32-1:0]   src1,          // 32 bits source 1          (input)
    input signed [32-1:0]   src2,          // 32 bits source 2          (input)
    input        [ 4-1:0]   ALU_control,   // 4 bits ALU control input  (input)
    output reg   [32-1:0]   result,        // 32 bits result            (output)
    output                  Zero          // 1 bit when the output is 0, zero must be set (output)
);

/* Write your code HERE */
    always@(*)begin
        if(rst_n) begin
            case(ALU_control)
                4'b0000: begin // and
                    result <= src1 & src2;
                end
                4'b0001: begin // or
                    result <= src1 | src2;
                end
                4'b0010: begin // add
                    result <= src1 + src2;
                end
                4'b0110: begin // sub
                    result <= src1 - src2;
                end
                4'b0111: begin // slt
                    result <= (src1<src2)? 32'b1 : 32'b0;
                end
                default: begin
                    result <= 32'b0 ;
                end
            endcase
        end
        else begin
            result <= 32'b0;
        end
    end
    assign Zero = !result;
endmodule
```

這次 lab 的運算只需要用到 and,or,add,sub,slt。根據不同的 alu_control，就對 alu 做不同運算，並透過 result 來算 zero。

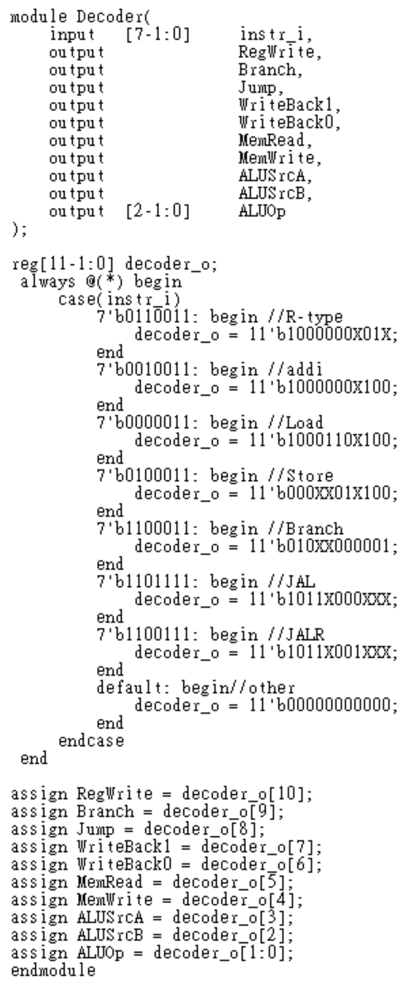
(3) alu_ctrl



```
module ALU_Ctrl(
    input       [4-1:0] instr,
    input       [2-1:0] ALUOp,
    output      [4-1:0] ALU_Ctrl_o
);
reg [4-1:0] Ctrl;
wire [2:0] func3;
assign func3 = instr[2:0];
wire [5:0] ctrl;
    assign ctrl = {ALUOp, instr};
always @(*) begin
    if(ctrl[5:4]==2'b00)begin
        case(ctrl[2:0])
            3'b000: begin //addi
                Ctrl=4'b0010;
            end
            3'b010: begin //lw sw
                Ctrl=4'b0010;
            end
            default: begin //default
                Ctrl=4'b0000;
            end
        endcase
    end
    else if(ctrl[5:4]==2'b01)begin //beq
                Ctrl=4'b0110;
    end
    else begin
        case(ctrl[3:0])
            4'b0000:begin //add
                Ctrl=4'b0010;
            end
            4'b1000:begin //sub
                Ctrl=4'b0110;
            end
            4'b0111:begin //and
                Ctrl=4'b0000;
            end
            4'b0110:begin //or
                Ctrl=4'b0001;
            end
            4'b0010:begin //slt
                Ctrl=4'b0111;
            end
            default: begin //default
                Ctrl=4'b0000;
            end
        endcase
    end
end
assign ALU_Ctrl_o = Ctrl;
endmodule
```

先看 aluop 來確認指令的 type。再從 instr 判斷是甚麼指令。判斷了甚麼指令後，就 output 對應的 alu_ctrl_o 給 alu 去做正確的運算。

(4) decoder

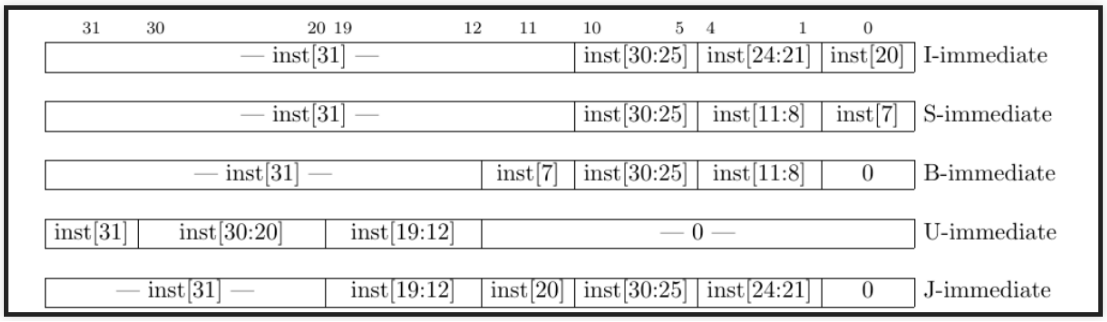


```
module Decoder(
    input   [7-1:0]     instr_i,
    output              RegWrite,
    output              Branch,
    output              Jump,
    output              WriteBack1,
    output              WriteBack0,
    output              MemRead,
    output              MemWrite,
    output              ALUSrcA,
    output              ALUSrcB,
    output  [2-1:0]     ALUOp
);

reg[11-1:0] decoder_o;
 always @(*) begin
     case(instr_i)
         7'b0110011: begin //R-type
             decoder_o = 11'b1000000X01X;
         end
         7'b0010011: begin //addi
             decoder_o = 11'b1000000X100;
         end
         7'b0000011: begin //Load
             decoder_o = 11'b1000110X100;
         end
         7'b0100011: begin //Store
             decoder_o = 11'b000XX01X100;
         end
         7'b1100011: begin //Branch
             decoder_o = 11'b010XX000001;
         end
         7'b1101111: begin //JAL
             decoder_o = 11'b1011X000XXX;
         end
         7'b1100111: begin //JALR
             decoder_o = 11'b1011X001XXX;
         end
         default: begin//other
             decoder_o = 11'b00000000000;
         end
     endcase
 end

assign RegWrite = decoder_o[10];
assign Branch = decoder_o[9];
assign Jump = decoder_o[8];
assign WriteBack1 = decoder_o[7];
assign WriteBack0 = decoder_o[6];
assign MemRead = decoder_o[5];
assign MemWrite = decoder_o[4];
assign ALUSrcA = decoder_o[3];
assign ALUSrcB = decoder_o[2];
assign ALUOp = decoder_o[1:0];
endmodule
```

在一開始我們已經寫好了表，根據 opcode，decoder 照著表輸出不同的指令。

(5) Imm_Gen(code 太長就不附在 report 了)

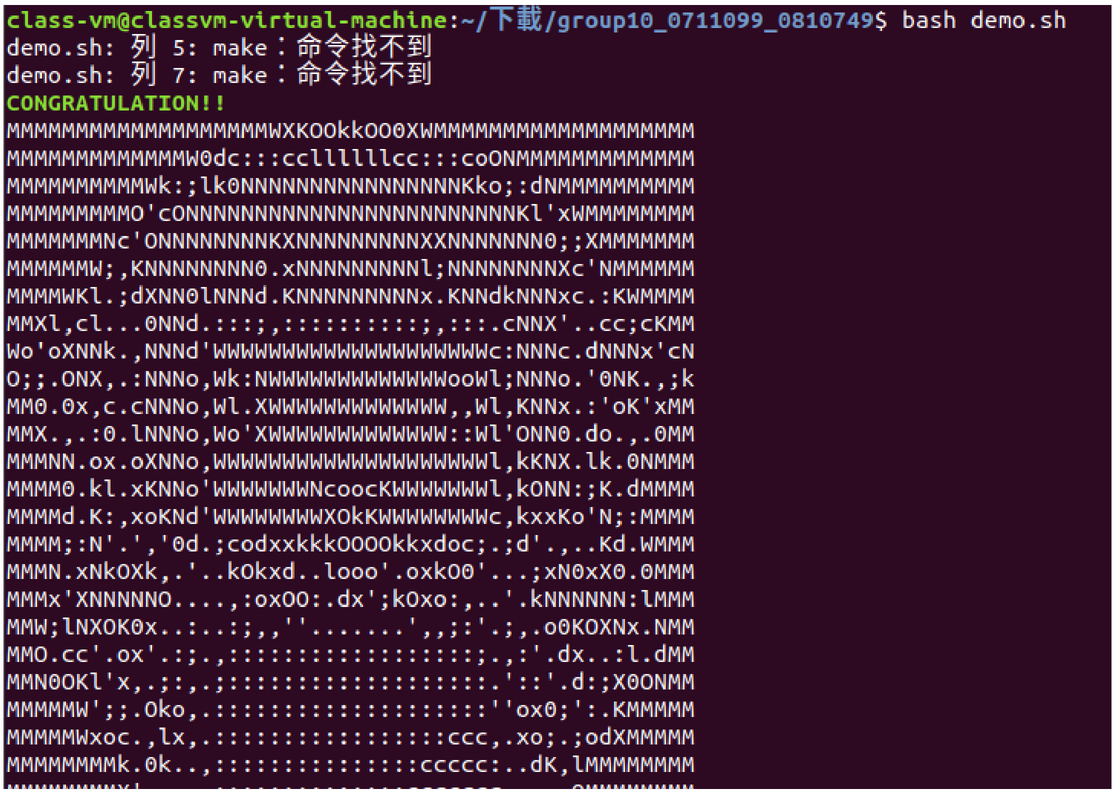
Sign-extend



31	30 20	19 12	11	10 5	4 1	0	
— inst[31] —				inst[30:25]	inst[24:21]	inst[20]	I-immediate
— inst[31] —				inst[30:25]	inst[11:8]	inst[7]	S-immediate
— inst[31] —			inst[7]	inst[30:25]	inst[11:8]	0	B-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —				U-immediate
— inst[31] —		inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0	J-immediate

照著表寫，根據 input opcode 判斷，r-type 就歸零，addi/load/jalr 就輸出 I-immediate，store 輸出 S-immediate，branch 輸出 B-immediate，jal 輸出 J-immediate。

(6) 運行



```
class-vm@classvm-virtual-machine:~/下載/group10_0711099_0810749$ bash demo.sh
demo.sh: 列 5: make：命令找不到
demo.sh: 列 7: make：命令找不到
CONGRATULATION!!
MMMMMMMMMMMMMMMMMMMMWXKOOkkOO0XWMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMW0dc:::cclllllllcc:::coONMMMMMMMMMMMMMM
MMMMMMMMMMWk:;lk0NNNNNNNNNNNNNNNNNNKko;:dNMMMMMMMMMM
MMMMMMMMMO'cONNNNNNNNNNNNNNNNNNNNNNNNNNKl'xWMMMMMMMM
MMMMMMMNc'ONNNNNNNNNKXNNNNNNNNNNXXNNNNNNNN0;;XMMMMMMM
MMMMMMW;,KNNNNNNNNN0.xNNNNNNNNNNl;NNNNNNNNNXc'NMMMMMM
MMMMWKl.;dXNN0lNNNd.KNNNNNNNNNNx.KNNdkNNNxc.:KWMMMM
MMXl,cl...0NNd.:::;,::::::::::;,:::.cNNX'..cc;cKMM
Wo'oXNNk.,NNNd'WWWWWWWWWWWWWWWWWWWWWc:NNNc.dNNNx'cN
O;;.ONX,.:NNNo,Wk:NWWWWWWWWWWWWWWWooWl;NNNo.'0NK.,;k
MM0.0x,c.cNNNo,Wl.XWWWWWWWWWWWWWW,,Wl,KNNx.:'oK'xMM
MMX.,.:0.lNNNo,Wo'XWWWWWWWWWWWWWW::Wl'ONN0.do.,.0MM
MMMNN.ox.oXNNo,WWWWWWWWWWWWWWWWWWWWWWl,kKNX.lk.0NMMM
MMMM0.kl.xKNNo'WWWWWWWNcoocKWWWWWWWWl,kONN:;K.dMMMM
MMMMd.K:,xoKNd'WWWWWWWWWXOkKWWWWWWWWWc,kxxKo'N;:MMMM
MMMM;:N'.','0d.;codxxkkkOOOOkkxdoc;.;d'.,..Kd.WMMM
MMMN.xNkOXk,.'..kOkxd..looo'.oxkO0'...;xN0xX0.0MMM
MMMx'XNNNNNO....,:oxOO:.dx';kOxo:,..'.kNNNNNN:lMMM
MMW;lNXOK0x..:..:;,,''.......',,;:'.;,.o0KOXNx.NMM
MMO.cc'.ox'.:;.,::::::::::::::::::;.,:'.dx..:l.dMM
MMN0OKl'x,.;:,.;:::::::::::::::::::.'::'.d:;X0ONMM
MMMMMW';;.Oko,.::::::::::::::::::::''ox0;':.KMMMMM
MMMMMWxoc.,lx,.:::::::::::::::::ccc,.xo;.;odXMMMMM
MMMMMMMMk.0k..,:::::::::::::::ccccc:..dK,lMMMMMMMM
```

成功運行

4. Anything you want to say.

真棒，從這次的 lab 中，了解到 cpu 結構真的可以很複雜，這跟平常上課畫的不太一樣。