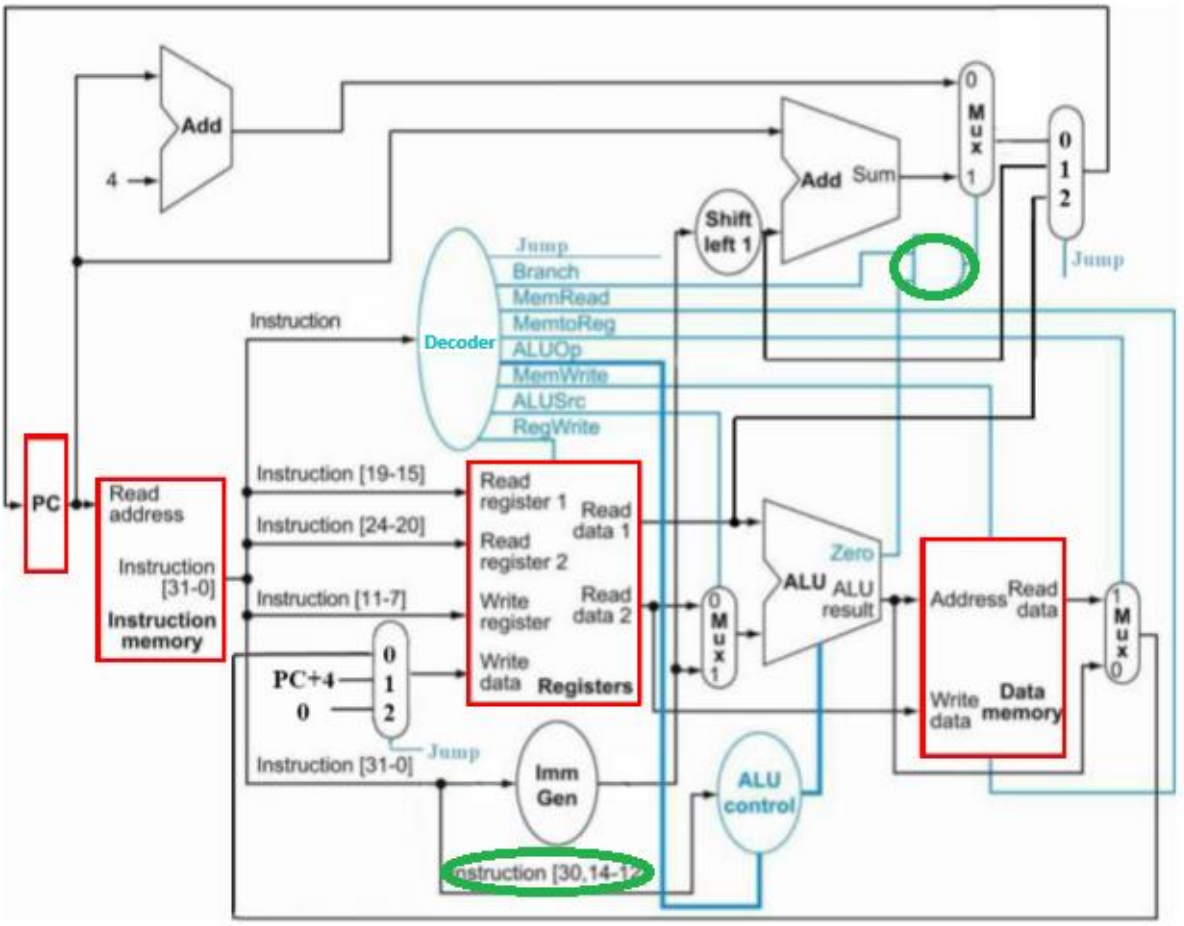
Computer Organization

Architecture diagram:



alu_control 更改了設計，讀取整個 **instruction**，利用整個 opcode 接進去比只用 alu_op 來得方便

宣告 **3mux1** 處理有 **jump** 的 **mux**

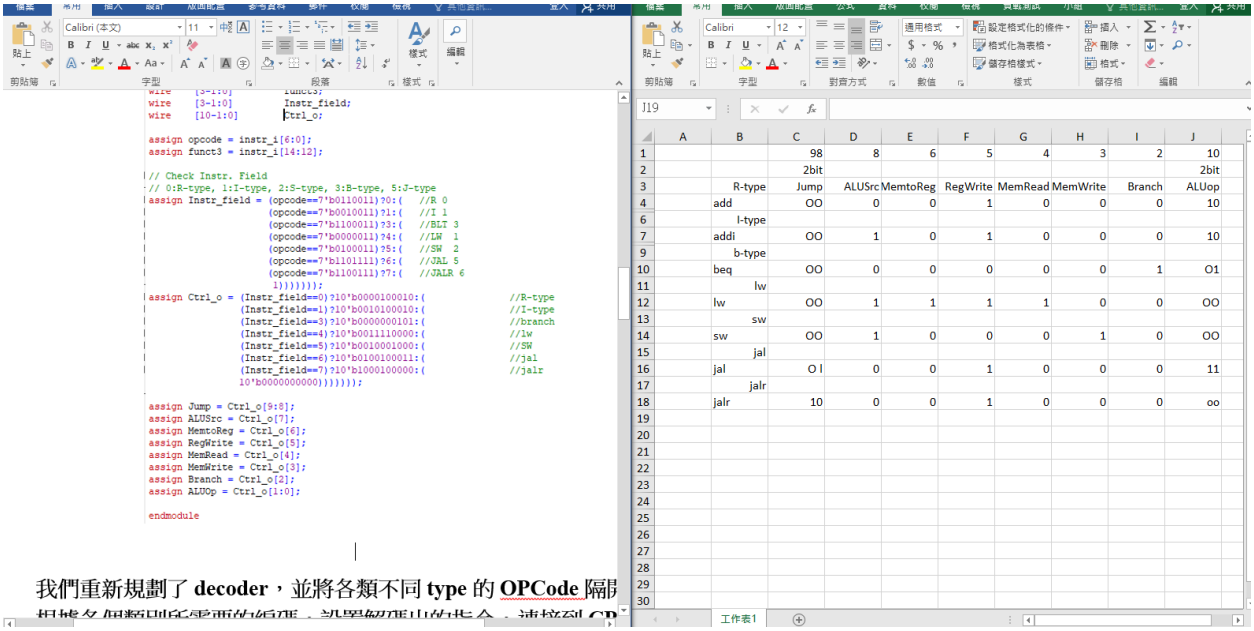
宣告 **and** 的 **wire** 來接線處理 **Branch** 和 **Zero**

Detailed description of the implementation:

```
/**************************************************
Student Name: 王培碩 施家齊
Student ID: 0816137 0716241
**************************************************/

`timescale 1ns/1ps

module Imm_Gen(
    input  [31:0] instr_i,
    output [31:0] Imm_Gen_o
    );

/* Write your code HERE */
reg [31:0] temp;
assign Imm_Gen_o=temp;

always@(*)
begin
    case (instr_i[6:0])
        7'b0000011://i-type lw
            temp={ {21{instr_i[31]}}    ,   instr_i[30:20]  };
        7'b0100011://S-type sw
            temp={ {21{instr_i[31]}}, instr_i[30:25], instr_i[11:7] };
        7'b1100011://sb-type beq bne blt bge
            temp={ {20{instr_i[31]}}    ,   instr_i[7]  ,   instr_i[30:25]  ,   instr_i[11:8] };
        7'b1101111://jal
            temp={ {13{instr_i[31]}}, instr_i[19:12], instr_i[20], instr_i[30:21]};
        default://i-type
            temp={ {21{instr_i[31]}} ,  instr_i[30:20] };//i-type addi sub andi xori slti
    endcase
end
endmodule
```

Imm 立即值處理部份，我們透過 OpCode 分出各個 type ，並參照 risc-V 的格式切割指令，這次除了舊有的還新增加 lw,sw,jal。

```
module Decoder(
        input [31:0]    instr_i,
        output          ALUSrc,
        output          MemtoReg,
        output          RegWrite,
        output          MemRead,
        output          MemWrite,
        output          Branch,
        output [1:0]    ALUOp,
        output [1:0]    Jump
        );

/* Write your code HERE */

//Internal Signals
wire    [7-1:0]         opcode;
wire    [3-1:0]         funct3;
wire    [3-1:0]         Instr_field;
wire    [10-1:0]        Ctrl_o;

assign opcode = instr_i[6:0];
assign funct3 = instr_i[14:12];

// Check Instr. Field
// 0:R-type, 1:I-type, 2:S-type, 3:B-type, 5:J-type
assign Instr_field = (opcode==7'b0110011)?0:(   //R 0
                     (opcode==7'b0010011)?1:(   //I 1
                     (opcode==7'b1100011)?3:(   //BLT 3
                     (opcode==7'b0000011)?4:(   //LW  1
                     (opcode==7'b0100011)?5:(   //SW  2
                     (opcode==7'b1101111)?6:(   //JAL 5
                     (opcode==7'b1100111)?7:(   //JALR 6
                      1)))))));
assign Ctrl_o = (Instr_field==0)?10'b0000100010:(               //R-type
                (Instr_field==1)?10'b0010100010:(               //I-type
                (Instr_field==3)?10'b0000000101:(               //branch
                (Instr_field==4)?10'b0011110000:(               //lw
                (Instr_field==5)?10'b0010001000:(               //SW
                (Instr_field==6)?10'b0100100011:(               //jal
                (Instr_field==7)?10'b1000100000:(               //jalr
                10'b0000000000)))))));

assign Jump = Ctrl_o[9:8];
assign ALUSrc = Ctrl_o[7];
assign MemtoReg = Ctrl_o[6];
assign RegWrite = Ctrl_o[5];
assign MemRead = Ctrl_o[4];
assign MemWrite = Ctrl_o[3];
assign Branch = Ctrl_o[2];
assign ALUOp = Ctrl_o[1:0];

endmodule
```



```
wire    [3-1:0]         Instr_field;
wire    [10-1:0]        Ctrl_o;

assign opcode = instr_i[6:0];
assign funct3 = instr_i[14:12];

// Check Instr. Field
// 0:R-type, 1:I-type, 2:S-type, 3:B-type, 5:J-type
assign Instr_field = (opcode==7'b0110011)?0:(   //R 0
                     (opcode==7'b0010011)?1:(   //I 1
                     (opcode==7'b1100011)?3:(   //BLT 3
                     (opcode==7'b0000011)?4:(   //LW  1
                     (opcode==7'b0100011)?5:(   //SW  2
                     (opcode==7'b1101111)?6:(   //JAL 5
                     (opcode==7'b1100111)?7:(   //JALR 6
                      1)))))));

assign Ctrl_o = (Instr_field==0)?10'b0000100010:(               //R-type
                (Instr_field==1)?10'b0010100010:(               //I-type
                (Instr_field==3)?10'b0000000101:(               //branch
                (Instr_field==4)?10'b0011110000:(               //lw
                (Instr_field==5)?10'b0010001000:(               //SW
                (Instr_field==6)?10'b0100100011:(               //jal
                (Instr_field==7)?10'b1000100000:(               //jalr
                10'b0000000000))))))));

assign Jump = Ctrl_o[9:8];
assign ALUSrc = Ctrl_o[7];
assign MemtoReg = Ctrl_o[6];
assign RegWrite = Ctrl_o[5];
assign MemRead = Ctrl_o[4];
assign MemWrite = Ctrl_o[3];
assign Branch = Ctrl_o[2];
assign ALUOp = Ctrl_o[1:0];

endmodule
```

	A	B	C	D	E	F	G	H	I	J
1			98	8	6	5	4	3	2	10
2			2bit							2bit
3		R-type	Jump	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop
4		add	OO	0	0	1	0	0	0	10
6		I-type								
7		addi	OO	1	0	1	0	0	0	10
9		b-type								
10		beq	OO	0	0	0	0	0	1	O1
11		lw								
12		lw	OO	1	1	1	1	0	0	OO
13		sw								
14		sw	OO	1	0	0	0	1	0	OO
15		jal								
16		jal	O I	0	0	1	0	0	0	11
17		jalr								
18		jalr	10	0	0	1	0	0	0	oo

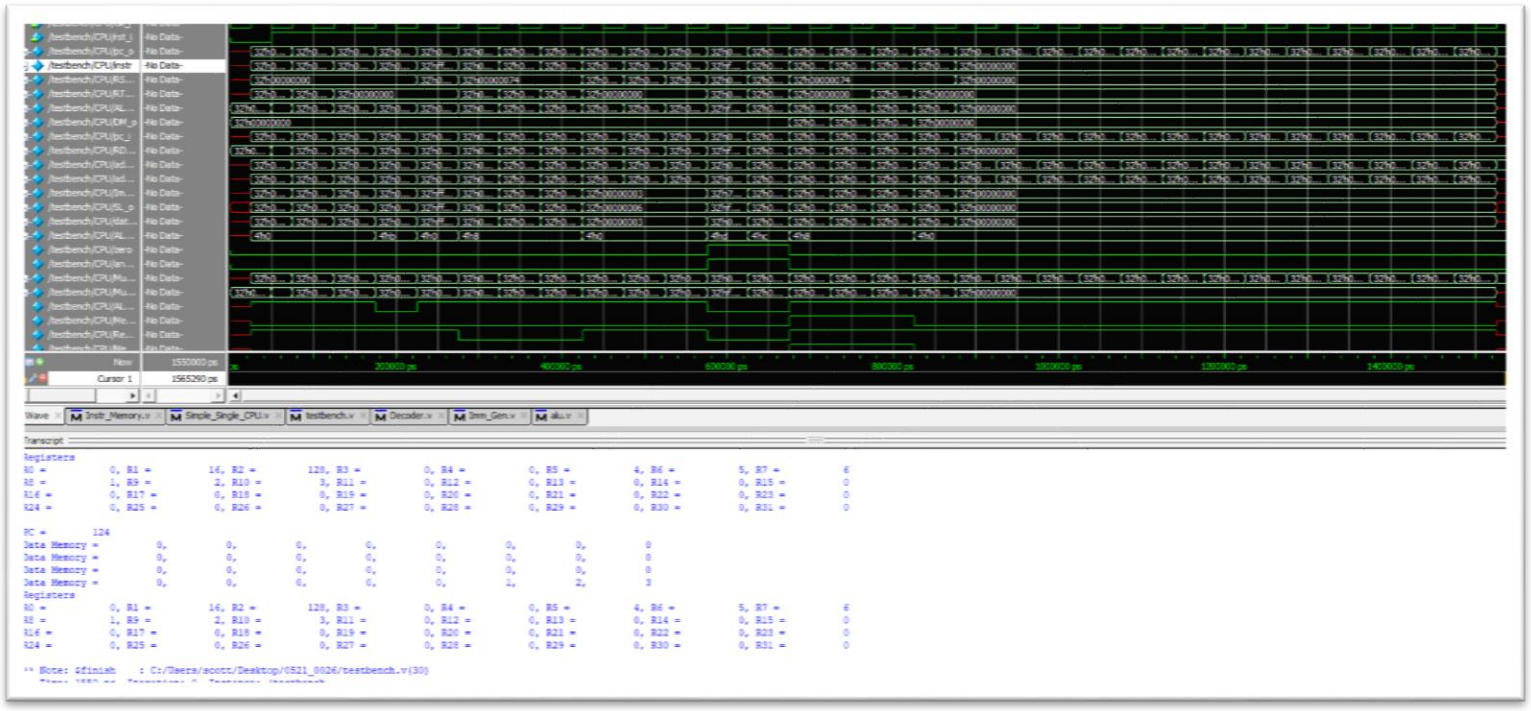
我們重新規劃了 **decoder**，並將各類不同 **type** 的 **OPCode** 隔開，再根據各個類別所需要的編碼，設置解碼出的指令，連接到 **CPU** 各處。

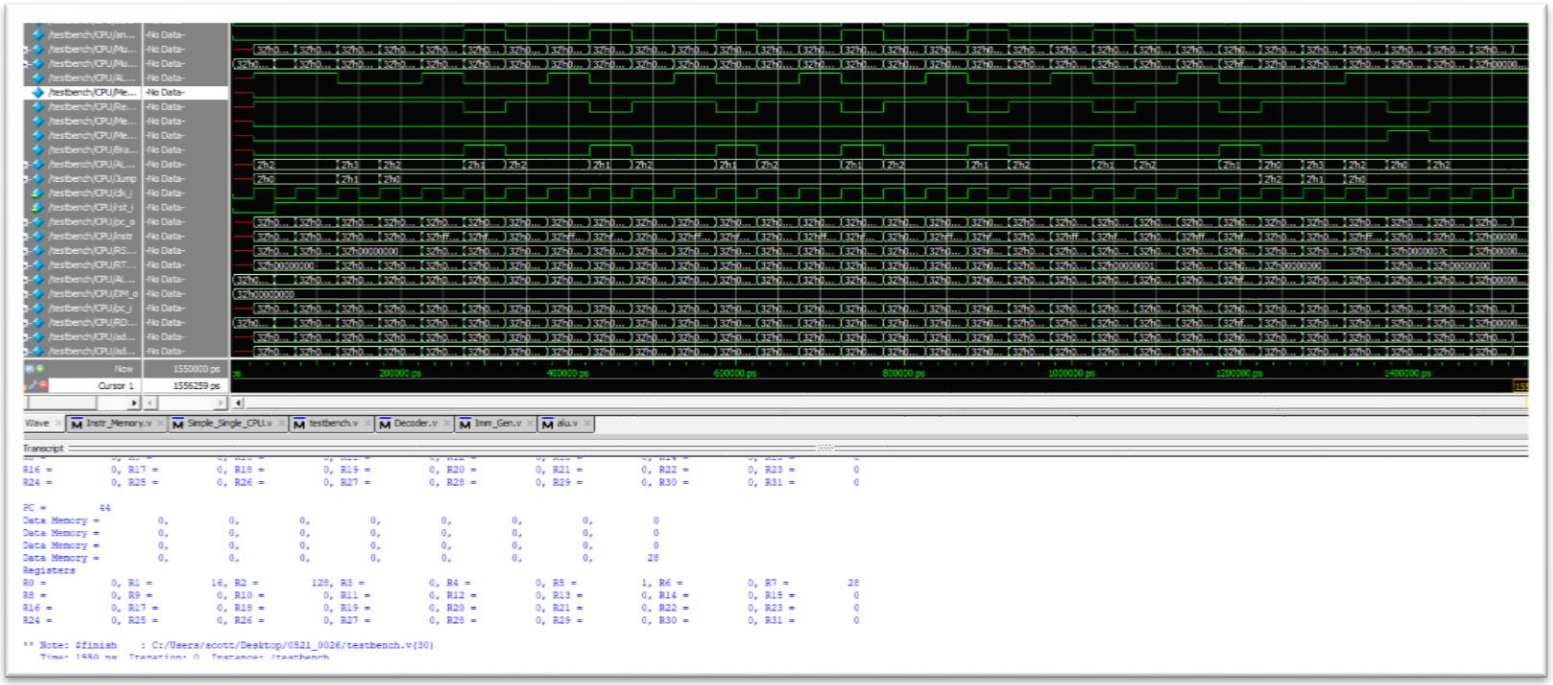
為求方便將 **jalr,lw,sw** 的 **ALUOp** 設成 **00**、剩餘 **R-type** 和 **I-type** 都設成 **10**、**jal** 設成 **11**。

```
module ALU_Ctrl(
        input   [4-1:0] instr,
        input   [2-1:0] ALUOp,
        input [6:0]             Opcode,
        output  [4-1:0] ALU_Ctrl_o
        );

/* Write your code HERE */
reg [4-1:0]temp;
assign ALU_Ctrl_o=temp;
always@(*)
begin
        case (ALUOp)//according to chap4-part1.pdf
                2'b10://r-type i-type
                        case(instr[2:0])
                                3'b000:begin
                                        if(Opcode==7'b0010011)
                                                temp<=4'b0000;//addi
                                        else if(instr[3]==1'b0)
                                                temp<=4'b0000;//add
                                        else
                                                temp<=4'b0111;//sub
                                end
                                3'b111:temp<=4'b0001;//and andi
                                3'b110:temp<=4'b0010;//or  ori
                                3'b100:temp<=4'b0011;//xor xori
                                3'b010:temp<=4'b0100;//slt slti
                                3'b001:temp<=4'b0101;//sll slli
                                3'b101:temp<=4'b0110;//sra srli
                        endcase
                2'b00://lw sw
                        case(instr[2:0])
                                3'b010:begin
                                        if(instr[3]==0)
                                                temp<=4'b1000;//lw
                                        else
                                                temp<=4'b1001;//sw
                                end
                                3'b000:temp<=4'b1010;//jalr
                        endcase
                2'b11:
                                temp<=4'b1011;//jal
                2'b01://beq bne
                        case(instr[2:0])
                                3'b000:temp<=4'b1110;//beq
                                3'b001:temp<=4'b1111;//bne
                                3'b100:temp<=4'b1101;//blt
                                3'b101:temp<=4'b1100;//bge
                        endcase

        endcase
end

endmodule
```

因為 **R-type** 和 **I-type** ，**ALU** 執行的動作相似度高，於是寫在一起。再根據各個不同的指令需求傳遞指令給 **ALU**，**ALU** 再根據收到的指令執行對應的操作。

Implementation results:





Problems encountered and solutions:

為完成這次作業，需要對這份 CPU 中的每個功能都有透徹的理解，為此就花了不少時間在這上面。結果出錯，問題有兩種可能；線接錯或程式有 bug。為了找到錯誤的源頭就必須細看波形圖，慢慢尋出端倪。

Comment:

這次我們卡在不少 verilog 的語法問題所造成的錯誤，下次使用 verilog 時要再更加注意、小心。