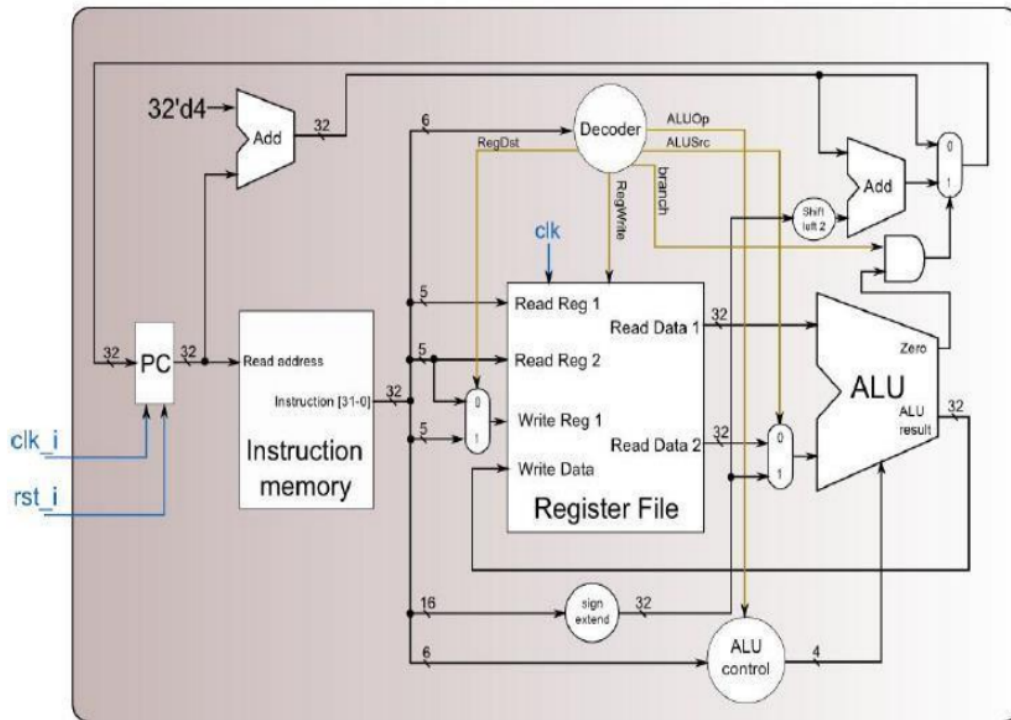


# Computer Organization

Architecture diagrams:



電路是依照這張圖寫的，一個 block 是一個.v 檔，檔案解說如下。

Hardware module analysis:

- 直覺的 module:

✧ adder.v

```
assign sum_o = src1_i + src2_i;
```

✧ Shift\_Left\_Two\_32.v

```
assign data_o= {data_i[29:0], 2'b00};
```

✧ Sign\_Extend. v

將 most significant bit extend

```
always@(*) begin
    data_o[15:0] <= data_i[15:0];
    if(data_i[15]==1'b1) begin
        data_o[31:16]<=16'b1111111111111111;
    end
    else begin
        data_o[31:16]<=16'b0000000000000000;
    end
end

end
```

0510008 藍挺毓

0510026 陳司瑋

✧ MUX\_2to1.v

```
data_o = (select_i)? data1_i: data0_i;
```

✧ ALU.v

依照助教給的 code 寫的，只改了變數的名稱。

```
assign zero_o = (result_o==0);
always @(ctrl_i, src1_i, src2_i) begin
    case (ctrl_i)
        0: result_o<=src1_i&src2_i;
        1: result_o<= src1_i | src2_i;
        2: result_o<=src1_i + src2_i;
        6: result_o<=src1_i - src2_i;
        7: result_o<=src1_i < src2_i ? 1:0 ;
        12: result_o<= ~(src1_i | src2_i);
        default: result_o<=0;
    endcase
end
```

## ● Control

✧ Decoder.v

```
always@(*) begin
    R_format <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);
    addi <= (~instr_op_i[5]) & (~instr_op_i[4]) & (instr_op_i[3]) & (~instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);
    slti <= (~instr_op_i[5]) & (~instr_op_i[4]) & (instr_op_i[3]) & (~instr_op_i[2]) & (instr_op_i[1]) & (~instr_op_i[0]);
    beq <= (~instr_op_i[5]) & (~instr_op_i[4]) & (~instr_op_i[3]) & (instr_op_i[2]) & (~instr_op_i[1]) & (~instr_op_i[0]);

    RegWrite_o <= R_format | addi | slti;
    ALUSrc_o <= addi | slti;
    RegDst_o <= R_format;
    Branch_o <= beq;
    ALU_op_o[0] <= beq | slti;
    ALU_op_o[1] <= R_format;
    ALU_op_o[2] <= addi | slti;
end
```

✧ ALU\_control.v

```
always@(*) begin
    ALUCtrl_o[3] = 0;
    ALUCtrl_o[2] = ALUOp_i[0] || (funct_i[1] && ALUOp_i[1]);
    ALUCtrl_o[1] = ALUOp_i[2] || ALUOp_i[0] || (ALUOp_i[1] && ~funct_i[2]);
    ALUCtrl_o[0] = (ALUOp_i[2]&& ALUOp_i[0]) || ((funct_i[0] || funct_i[3]) && ALUOp_i[1]);
end
```

control 是照下面的表格設計的，R-format 和 Beq 是造 MIPS 原本的設計，而 Addi 和 Slti 的 ALU OP 是自己設定的，再依這個設定把電路推出來。

而電路設計方式很簡單，都是用一個大 and 再把結果為 1 的都 or 起來。

	R-format	ADDI	SLTI	Beq
	0	0	0	0
	0	0	0	0
	0	1	1	0
	0	0	0	1
	0	0	1	0
	0	0	0	0
<hr/>				
RegDst	1	0	0	X
ALUSrc	0	1	1	0
Regwrite	1	1	1	0
Branch	0	0	0	1
<hr/>				
ALUop0	0	0	1	1
ALUop1	1	0	0	0
ALUop2	0	1	1	0

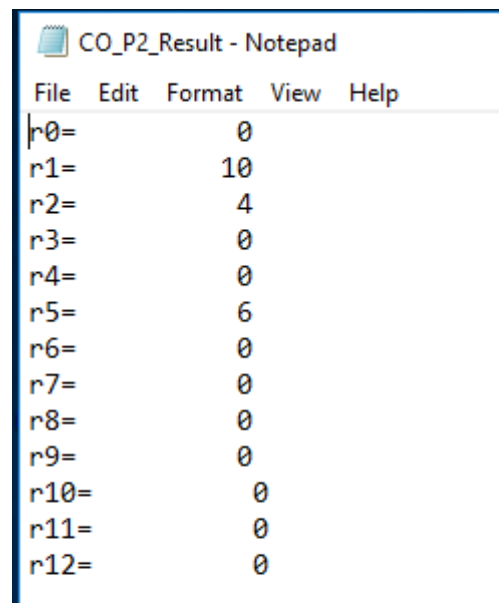
	ALUOP	funct	ALU control input
Add	010	100000	0010
Sub	010	100010	0110
AND	010	100100	0000
OR	010	100101	0001
Slt	010	101010	0110
ADDI	100	000000	0010
SLTI	101	000000	0110
Beq	001	000000	0110

Finished part:

CO\_P2\_test\_data1.txt result

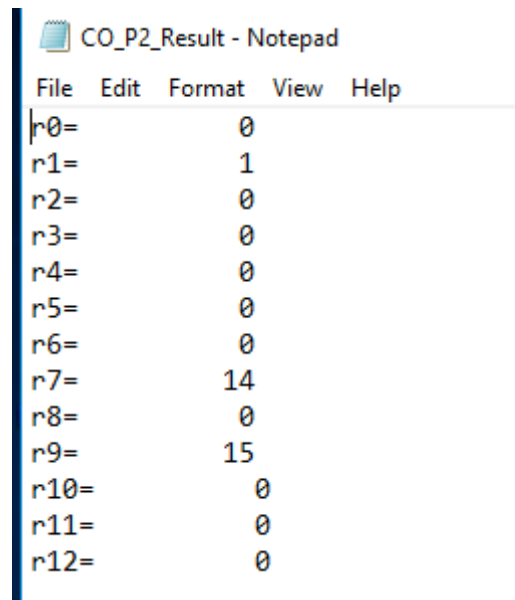
0510008 藍挺毓

0510026 陳司瑋



```
File Edit Format View Help
r0= 0
r1= 10
r2= 4
r3= 0
r4= 0
r5= 6
r6= 0
r7= 0
r8= 0
r9= 0
r10= 0
r11= 0
r12= 0
```

CO\_P2\_test\_data2.txt result



```
File Edit Format View Help
r0= 0
r1= 1
r2= 0
r3= 0
r4= 0
r5= 0
r6= 0
r7= 14
r8= 0
r9= 15
r10= 0
r11= 0
r12= 0
```

## Problems you met and solutions:

因為修過 Dlab 的關係，這次的 lab 也沒有遇到甚麼問題。

不過一開始因為沒有了解題目，所以 control 的部分照課本寫了有 lw, sw 的版本且沒有寫到 addi 和 slti。所以是第二次修的時候才畫了上面的 1, 0 表格推導電路。

## Summary:

Lab2 就是用 verilog 模擬實作 single cycle 的電路，透過此次 lab 的實作讓我

0510008 藍挺毓

0510026 陳司瑋

們更確定 single cycle 的運作跟它會遇到的 condition 選擇。最大的電路有點像連連看，只是要一直設變數，會搞得有一點點暈頭轉向的，但整體來說還蠻順利的，只有 decoder 跟 control 的部分需要多想一下，其他都是蠻簡單的小塊邏輯電路。