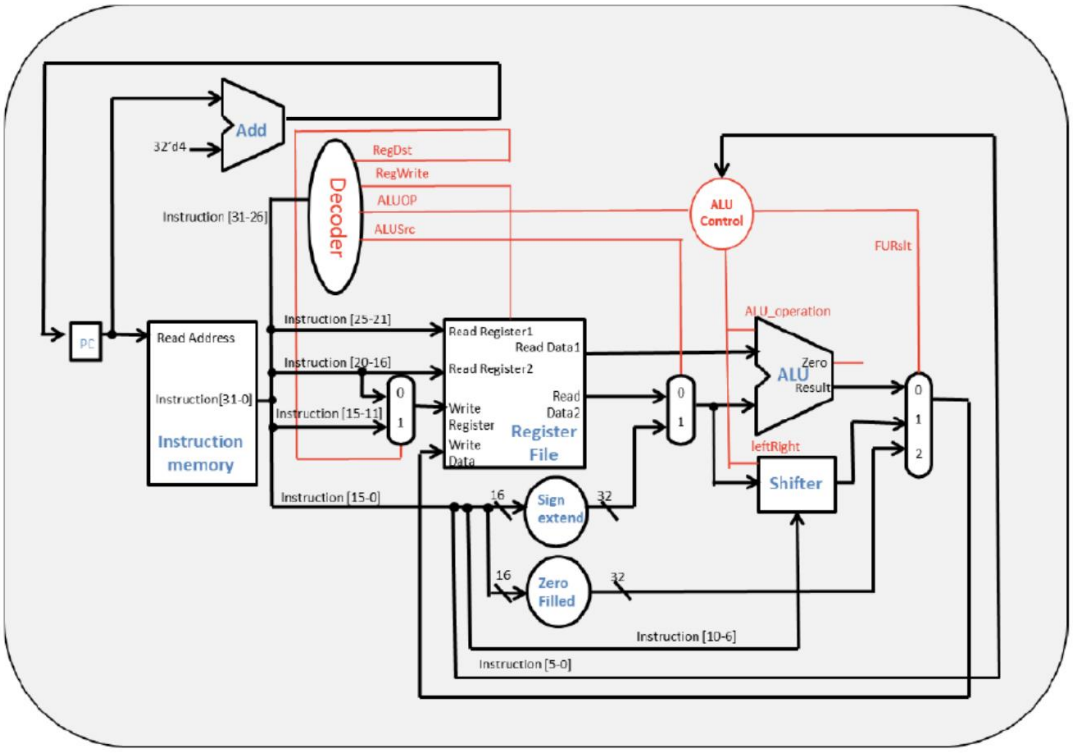
Computer Organization

Architecture diagrams:



照著圖片想辦法把線和模組都接在一起就好

Hardware module analysis:

Adder: 就是簡單的將兩個數字相加，用在找 pc+4 的位置

ALU: 就是最重要的計算單位，我用了 spec 上提供的 32-bit ALU，所以沒有使用到 ALU-1-bit 以及 Full Adder 的 module

ALU_Control: 根據指令種類決定給 ALU 的輸入

Decoder: 根據指令種類決定輸出

Mux2to1: 根據 select 輸入選擇輸出兩個 input 其中一個

Mux3to1: 根據 select 輸入選擇輸出三個 input 其中一個

Shifter: 根據 leftright 決定左移或右移 shamt 個單位

Sign_Extend: 把數字擴充成 32 位，要注意正負數

Zero_Filled: 用 0 填充到 32 位

Finished part:

Adder.v

```
//Main function
/*your code here*/
assign sum_o = src1_i + src2_i;
```

ALU.v

```
//Main function
/*your code here*/
assign zero_o = ~(|result);

always@(*)begin
    case(ALU_operation_i)
        4'b0000: result = aluSrc1 | aluSrc2;
        4'b0001: result = aluSrc1 & aluSrc2;
        4'b0010: result = $signed(aluSrc1) + $signed(aluSrc2);
        4'b0110: result = $signed(aluSrc1) - $signed(aluSrc2);
        4'b0111: result = ($signed(aluSrc1) < $signed(aluSrc2)) ? 32'b1 : 32'b0;
        4'b1100: result = ~(aluSrc1 | aluSrc2);
        default: result = 0;
    endcase
end
```

ALU_Ctrl.v:

```
assign FURslt_o = ((funct_i == 6'b000000) || (funct_i == 6'b000010)) ? 2'b01 : 2'b00;
//Select exact operation
always@(*)begin
    if(ALUOp_i == 3'b000) begin
        ALU_operation_o = 4'b0010; // addi
    end
    else begin
        case(funct_i)
            6'b010010: ALU_operation_o = 4'b0010; // add
            6'b010000: ALU_operation_o = 4'b0110; // sub
            6'b010100: ALU_operation_o = 4'b0001; // and
            6'b010110: ALU_operation_o = 4'b0000; // or
            6'b010101: ALU_operation_o = 4'b1100; // nor
            6'b100000: ALU_operation_o = 4'b0111; // slt
            6'b000000: ALU_operation_o = 4'b0001; // sll
            6'b000010: ALU_operation_o = 4'b0000; // srl
            default: ALU_operation_o = 4'b0000;
        endcase
    end
end
```

Decoder.v

```
always@(*) begin
    case(instr_op_i)
        6'b000000:          // R-Type
        begin
            ALUOp_o = 3'b010;
            RegDst_o = 1'b1;
            ALUSrc_o = 1'b0;
            RegWrite_o = 1'b1;
        end

        6'b001000:          // addi
        begin
            ALUOp_o = 3'b000;
            RegDst_o = 1'b0;
            ALUSrc_o = 1'b1;
            RegWrite_o = 1'b1;
        end

    endcase
end
```

Mux2to1.v

```
//Main function
/*your code here*/
assign data_o = select_i ? data1_i : data0_i;
```

Mux3to1.v

```
//Main function
always@(*) begin

    if(select_i == 2'b00) data_o = data0_i;
    else if(select_i == 2'b01) data_o = data1_i;
    else if(select_i == 2'b10) data_o = data2_i;

end
```

Shifter.v

```
//Main function
/*your code here*/
assign result = leftRight ? sftSrc << shamt : sftSrc >> shamt ;
```

Sign_Extend.v

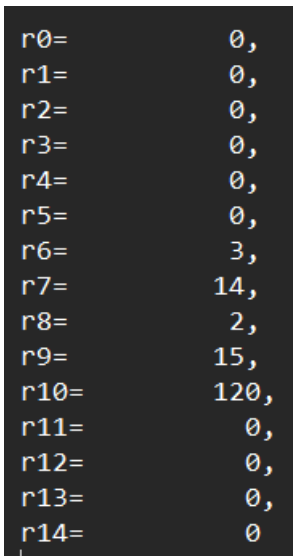
```
always@(*)
begin
    data_o[32-1:0] = { {16{data_i[15]}}, data_i[15:0] };
end
```

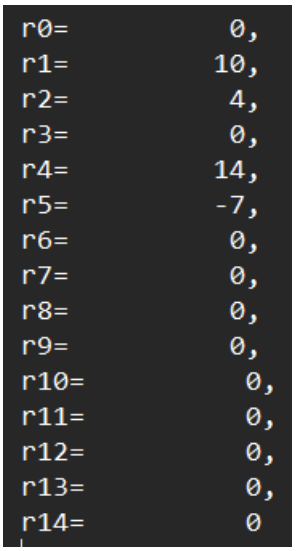
Zero_Filled.v

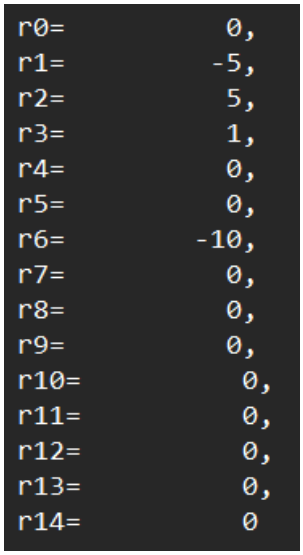
```
assign data_o[15:0] = data_i;
assign data_o[31:16] = 16'b0;
```

Result:

```
==========================================
   Congratulation. You pass  TA's pattern
==========================================
```







Problems you met and solutions:

一開始碰到的問題是沒看到討論區提出的問題，以為不需要改動 simple single cpu 裡面的 code，所以就遇到了 PC Address 錯誤的問題。在接好 cpu 裡面的線路之後，跑 simulation 又發現輸入的值都沒有被存進register，找了好久才發現我的ALU control 和 Decoder都有問題，根本不太可能發現，花了我很久的時間。

Summary:

這次的作業相較於上一次要做的部分更多了，但因為有上一次的經驗，所以也沒有想像中困難。在完成這份 Project 的過程中，我對使用 Vivado 和 verilog 來完成電路設計更加熟悉，也慢慢感受到寫硬體的樂趣。