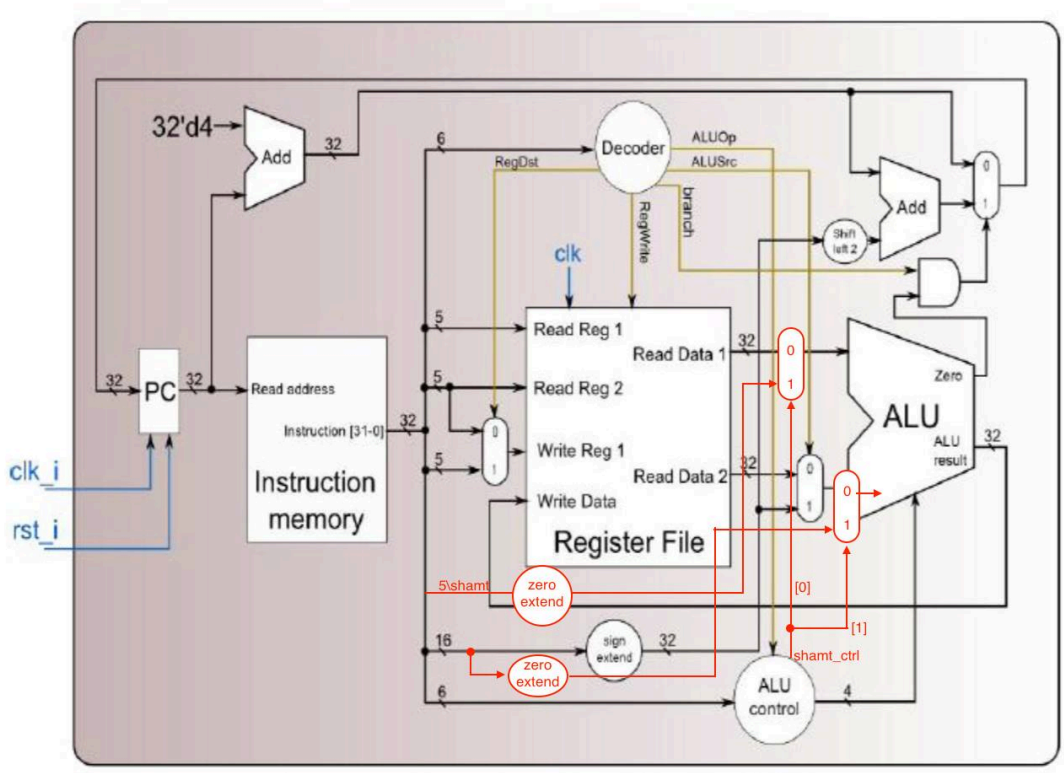
Computer Organization

0316313 張逸群 0316055 許庭嫣

Architecture diagram:



Top module: Simple_Single_CPU

Detailed description of the implementation:

一、Zero-Extension

$$\begin{cases} output[\,size_{input}-1:0\,] = input \\ output[\,31:\,size_{input}\,] = 0 \end{cases}$$

二、Sign-Extension

$$\begin{cases} output[\,size_{input}-1:0\,] = input \\ output[\,31:size_{input}\,] = input[size_{input}-1\,] \end{cases}$$

三、Shift-left 2

$$\text{output[i]} = \begin{cases} input[\,i-2], & 31 \geq i \geq 2 \\ 0, & 2 > i \geq 0 \end{cases}$$

四、Decoder

```
case( instr_op_i )
    INSTR_ZERO:begin // r-type instr: add, sub, and, or, slt
        RegDst_o        = 1;
        ALUSrc_o        = 0;
        RegWrite_o      = 1;
        Branch_o        = 0;
        ALU_op_o        = ALUOP_R;
    end
    INSTR_ADDI:begin
        RegDst_o        = 0;
        ALUSrc_o        = 1;
        RegWrite_o      = 1;
        Branch_o        = 0;
        ALU_op_o        = ALUOP_ADDI;
    end
    INSTR_SLTIU:begin
        RegDst_o        = 0;
        ALUSrc_o        = 1;
        RegWrite_o      = 1;
        Branch_o        = 0;
        ALU_op_o        = ALUOP_SLTIU;
    end
    INSTR_BEQ:begin
        RegDst_o        = 0; // don't care
        ALUSrc_o        = 0;
        RegWrite_o      = 0;
        Branch_o        = 1;
        ALU_op_o        = ALUOP_BEQ;
    end
    INSTR_LUI:begin
        RegDst_o        = 0;
        ALUSrc_o        = 1; // don't care
        RegWrite_o      = 1;
        Branch_o        = 0;
        ALU_op_o        = ALUOP_LUI;
    end
    INSTR_ORI:begin
        RegDst_o        = 0;
        ALUSrc_o        = 1; // don't care
        RegWrite_o      = 1;
        Branch_o        = 0;
        ALU_op_o        = ALUOP_ORI;
    end
```

```
INSTR_BNE:begin
    RegDst_o        = 0; // don't care
    ALUSrc_o        = 0;
    RegWrite_o      = 0;
    Branch_o        = 1;
    ALU_op_o        = ALUOP_BNE;
end
default:begin
    RegWrite_o = 0;
end
```

五、**ALU-control**

```
always@(*)begin
    case( ALUOp_i )
        ALUOP_R:begin
            case( funct_i )
                // add rd,rs,rt
                32:ALUCtrl_o = CTRL_ADD;
                // sub rd,rs,rt
                34:ALUCtrl_o = CTRL_SUB;
                // and rd,rs,rt
                36:ALUCtrl_o = CTRL_AND;
                // or rd,rs,rt
                37:ALUCtrl_o = CTRL_OR;
                // slt rd,rs,rt
                42:ALUCtrl_o = CTRL_SLT;
                // sra rd,rt,shamt
                3:ALUCtrl_o = CTRL_SHR;
                // srav rd,rt,rs
                7:ALUCtrl_o = CTRL_SHR;
                // all zero
                default:ALUCtrl_o = CTRL_IDLE;
            endcase
        end
        //addi rt,rs,se100
        ALUOP_ADDI:ALUCtrl_o = CTRL_ADD;
        //sltiu rt,rs,ze10
        ALUOP_SLTIU:ALUCtrl_o = CTRL_SLT;
        //beq rs,rt,se25
        ALUOP_BEQ:ALUCtrl_o = CTRL_SUB;
        //lui rt,10
        ALUOP_LUI:ALUCtrl_o = CTRL_LUI;
        //ori rt,rs,ze100
        ALUOP_ORI:ALUCtrl_o = CTRL_OR;
        //bne rs,rt,se30
        ALUOP_BNE:ALUCtrl_o = CTRL_BNE;
    endcase
end

//use shamt as input
always@(*)begin
    if( ALUOp_i == ALUOP_R && funct_i == 3 )
        shamt_ctrl_o = 2'b01; // shamt
    else if( ALUOp_i == ALUOP_ORI || ALUOp_i == ALUOP_SLTIU )
        shamt_ctrl_o = 2'b10; // zero-extended immediate
    else
        shamt_ctrl_o = 2'b00;
end
```

六、ALU

```
//Main function
assign zero_o = (result_o==0);  //zero is true if result_o is 0

always @(ctrl_i, src1_i, src2_i) begin  //reevaluate if these changes
    case (ctrl_i)
        OP_AND: result_o <= src1_i & src2_i;
        OP_OR:  result_o <= src1_i | src2_i;
        OP_ADD: result_o <= src1_i + src2_i;
        OP_SUB: result_o <= src1_i - src2_i;
        OP_SLT: result_o <= (src1_i<src2_i)? 1: 0;
        OP_SHR: result_o <= src2_i >>> src1_i;
        OP_LUI: result_o <= src2_i << 16;
        OP_BNE: result_o <= (src1_i == src2_i);
        OP_IDLE: result_o <= src2_i;
        default: result_o <= 0;
    endcase
end
```

Problems encountered and solutions:

問題：

在測資結束後會出現 32'd0 的指令進入系統，如果沒有處理的話將會使 r0 被讀入一些不預期的值。

解法：

因為 Decoder 無法得知指令是否為 32'd0，因此將依照 R-type 的指令處理，而將交由 ALU_Ctrl 送 IDEL 指令至 ALU，使 ALU 將 r0 終得值直接送出，如此便不會使 r0 出現不可預期的狀況。

Lesson learnt (if any):

我們了解到 cpu 如何進行 fetch, decode, execute 跟 store 這幾個步驟，而在 decoder 只會分辨出 R-type 與 I-type instruction 之間的差異，主要 ALU 的控制還是交由 ALU_Ctrl 針對 func-code 去做控制。