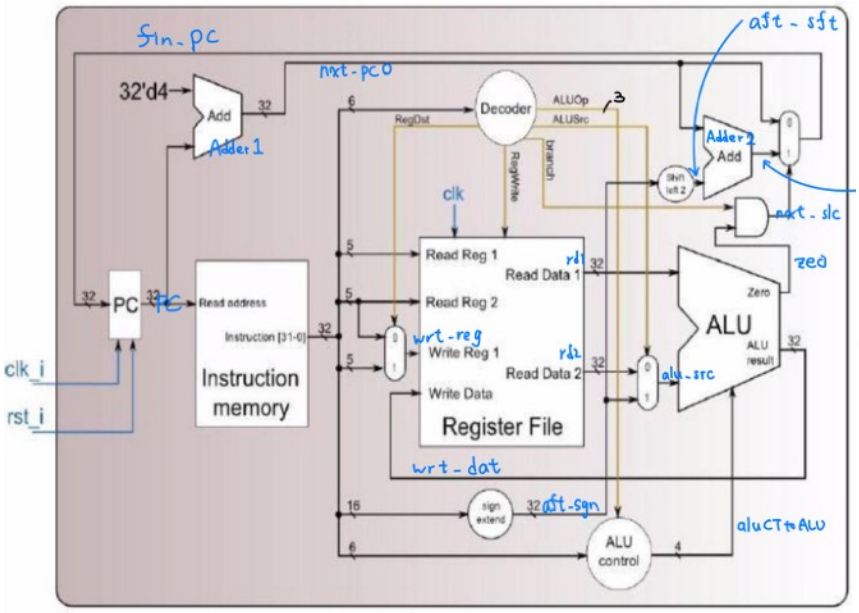
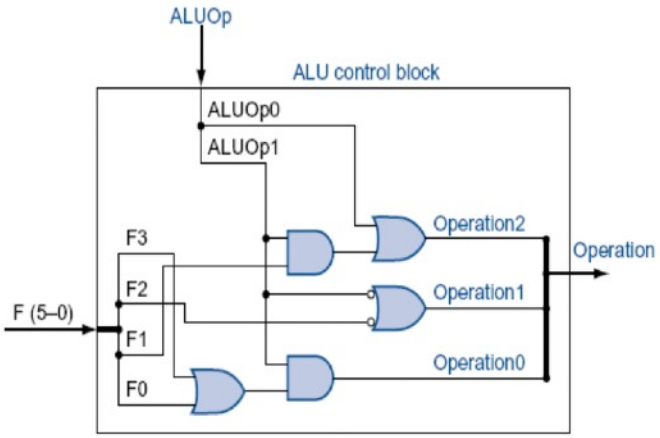
Computer Organization - Lab2 Single Cycle CPU

Architecture diagrams

Single Cycle CPU

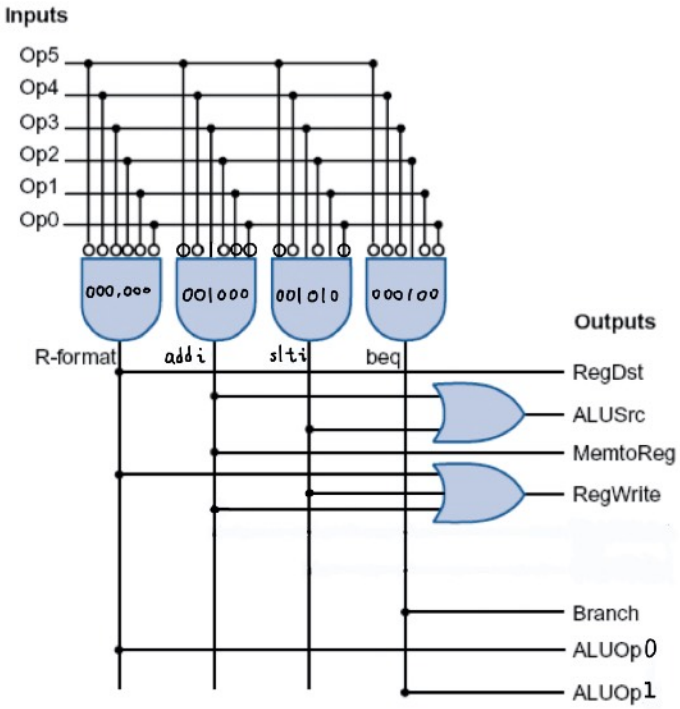


ALU Control



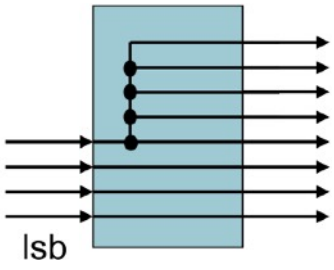
這張圖的 ALUOp 只有 2–bits，我們這次的為 3–bits ，但是 ALUOp[0] 跟 ALUOp[1] 的電路圖為上圖，最後 1–bit 做的事情會在 Part: module analysis 做說明。

Decoder

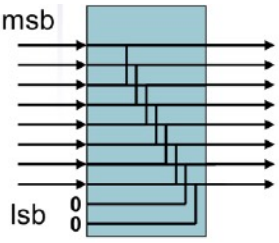


左圖為 Decoder ，用上課介紹的方式，把不同的指令 And ，然後根據要的 Ouputs 把為 1 的 Or 起來，如左圖。

Sign Extend



Shift Left 2



其它：ALU, Adder, MUX 與之前的相同（Lab0, 1 & DCD），故這裡略。

Hardware module analysis

```
always @(*) begin
    // Not addi or slti, -> R-format & Beq
    if (ALUOp_i[2] == 1'b0) begin
        ALUCtrl_o[3] <= op3;
        ALUCtrl_o[2] <= op2;
        ALUCtrl_o[1] <= op1;
        ALUCtrl_o[0] <= op0;
    end
    else begin
        case(ALUOp_i)
            3'b110: ALUCtrl_o <= 4'b0010;
            3'b101: ALUCtrl_o <= 4'b0111;
            default:
                ALUCtrl_o = 4'b0000;
        endcase
    end
end
```

ALU_Ctrl

我將 ALUOp[2] 這位為 1 的時候當作是 Addi 以及 slti 的訊號，然後將 ALUOp[1:0] 分別設為 10 跟 01 ，來做對應的 ALU crtl 給 ALU ，其他的四個操作就如同第一部分所放的圖的方法設計。

```
always @(*) begin
    RegDst_o <= rfmt;
    RegWrite_o <= rfmt | addi | slti;
    Branch_o <= bieq;
    ALUSrc_o <= addi | slti;
end
always @(*) begin
    case(instr_op_i)
        6'b001000: ALU_op_o <= 3'b110;
        6'b001010: ALU_op_o <= 3'b101;
        6'b000000: ALU_op_o <= 3'b010;
        6'b000100: ALU_op_o <= 3'b001;
        default:
            ALU_op_o <= 3'b000;
    endcase
end
```

Decoder

我將不同的指令運用 not 跟 and 把它們算出來目前適用哪一個指令，然後根據要的 Control 訊號去做 Or 起來。 送給 ALU_Ctrl 的訊號而是根據不同的指令去直接做對應的訊號。

Finished Part

最後根據兩組不同的輸入去模擬最後的結果，可以發現結果與預期吻合。

Problems & Solutions

這次在寫 *Verilog* 上有比前兩次順暢許多，也許是運氣好，也許是比較習慣 *Verilog* 的語法吧。這次有遇到的問題是在第一次完整電路測試的時候，一直有一個訊號為 x ，所以我用搜尋功能一個一個變數去找，才發現有個大小寫寫錯，所以造成沒有訊號的問題。解決後就剩一些控制訊號錯的問題，根據電路圖與對應特輸出，去改控制訊號的接線。

Summary

這次的 *Single Cycle CPU* 是我第一次寫這麼大的電路，寫完不僅了解其內部的運作方式、原理，還帶來給我很大的成就感。對於以往只知道使用電腦但不知道 *CPU* 怎麼運作的我，我覺得就由這次的作業，讓我有了初步的認識。