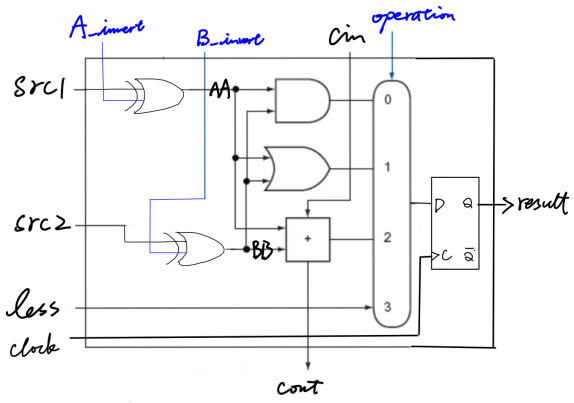
Computer Organization

Lab 1: 32-bit ALU

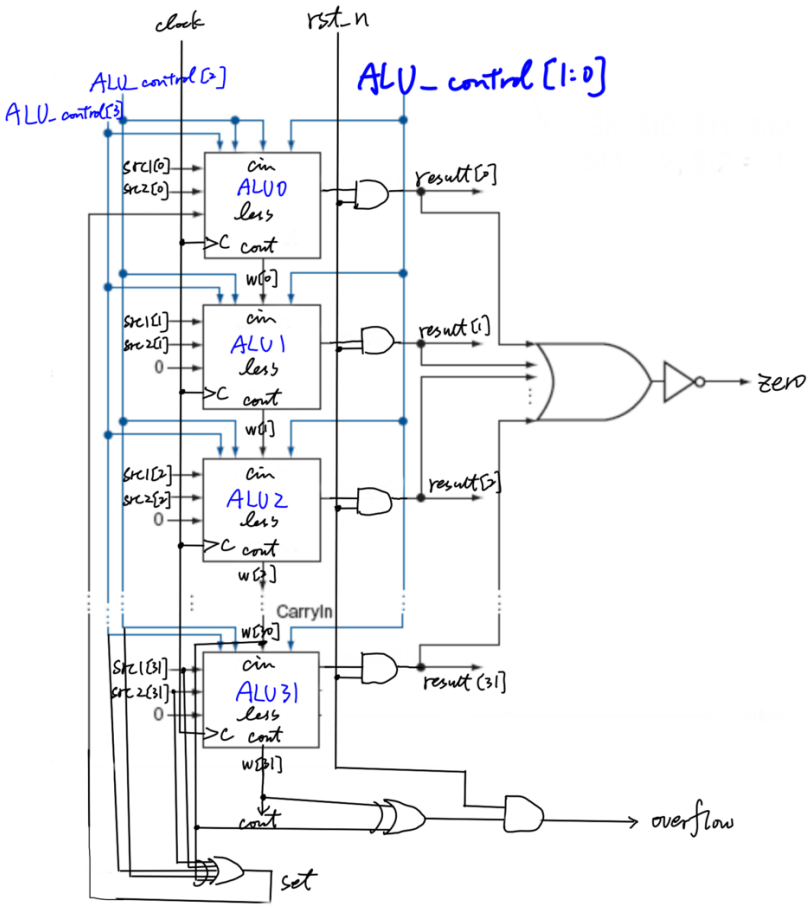
Student ID: 110550029　Name: 陳芷萱

1. Architecture diagrams

- alu_top



- alu



2. Hardware module analysis

- alu_top.v

  先以 AA 儲存 src1 根據 A_invert 轉換後的結果，以 BB 儲存 src2 根據 B_invert 轉換後的結果。AA 與 BB 表示實際參與運算的數。

  若 operation 為 00(0)，表示做 AND 運算；operation 為 01(1)，表示做 AND 運算；operation 為 10(2)，表示做加法運算；operation 為 11(3)，表示做比較，根據設計的電路圖，直接輸出 less 即可。

  由於 AND 與 OR 運算不需考慮進位，cout 直接輸出 0；加法運算或比較運算時，若 AA、BB 或 cin 有兩個以上為 1，則會產生進位。

```
reg result, cout;
wire AA, BB;

/*=============================================================*/
/*                            design                           */
/*=============================================================*/

assign AA=src1^A_invert;
assign BB=src2^B_invert;

/* HINT: You may use 'case' or 'if-else' to determine result.*/
// result
always@(*) begin
    case(operation)
        2'b00: begin result=AA&BB; cout=1'b0; end
        2'b01: begin result=AA|BB; cout=1'b0; end
        2'b10: begin result=AA^BB^cin; cout=(AA&BB)|(AA&cin)|(BB&cin); end
        2'b11: begin result=less; cout=(AA&BB)|(AA&cin)|(BB&cin); end
        default:
                begin
                        result = 0;
                end
    endcase
end
```

- alu.v

  根據題目，若 reset 為 0 則設定 result、overflow、cout 等訊號為 0；若 reset 為 1 且偵測到 clock 的正緣訊號時，將每個 1-bit ALU 單元的結果寫入 result 的 register，ALU31 的進位為整個 ALU 的 cout 輸出，ALU31 的進位輸入與輸出做 XOR 結果為整個 ALU 的 overflow。zero 則是無論 reset 值為何，輸出 result 是否等於 0。

  由於此電路並未對於第 31 個 1-bit ALU 特別設計，而是將 set 與 overflow 另做處理，因此要另外設定 set 為 ALU31 之 src1 與 src2 處理是否反轉後與 ALU30 進位相加的結果。最後將 1-bit ALU 彼此連起，以 w 作為 cin、cout 間的連線，以 r 接收每個 1-bit ALU 的輸出，並且將 set 連至 ALU0 的 less 訊號。

```
reg [32-1:0] result;
reg zero, cout, overflow;
wire w[31:0], r[31:0];
wire set;
integer i;

/*=============================================================================*/
/*                                  design                                     */
/*=============================================================================*/

always@(posedge clk or negedge rst_n)
begin
        if(!rst_n) begin
                result=32'b0;
                cout=0;
                overflow=0;
        end
        else begin
                for(i=0; i<=31; i++) result[i]=r[i];
                if(ALU_control[1:0]==2'b10) begin
                        overflow=w[30]^w[31];
                        cout=w[31];
                end
                else begin
                        overflow=0;
                        cout=0;
                end
        end
        zero=(result==0);
end

assign set=(src1[31]^ALU_control[3])^(src2[31]^ALU_control[2])^w[30];
```

```
/* HINT: You may use alu_top as submodule.*/
// 32-bit ALU
alu_top
ALU00(.src1(src1[0]), .src2(src2[0]), .less(set), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(ALU_control[2]), .operation(ALU_control[1:0]), .result(r[0]), .cout(w[0]));
alu_top ALU01(.src1(src1[1]), .src2(src2[1]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[0]), .operation(ALU_control[1:0]), .result(r[1]), .cout(w[1]));
alu_top ALU02(.src1(src1[2]), .src2(src2[2]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[1]), .operation(ALU_control[1:0]), .result(r[2]), .cout(w[2]));
alu_top ALU03(.src1(src1[3]), .src2(src2[3]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[2]), .operation(ALU_control[1:0]), .result(r[3]), .cout(w[3]));
alu_top ALU04(.src1(src1[4]), .src2(src2[4]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[3]), .operation(ALU_control[1:0]), .result(r[4]), .cout(w[4]));
alu_top ALU05(.src1(src1[5]), .src2(src2[5]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[4]), .operation(ALU_control[1:0]), .result(r[5]), .cout(w[5]));
alu_top ALU06(.src1(src1[6]), .src2(src2[6]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[5]), .operation(ALU_control[1:0]), .result(r[6]), .cout(w[6]));
alu_top ALU07(.src1(src1[7]), .src2(src2[7]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[6]), .operation(ALU_control[1:0]), .result(r[7]), .cout(w[7]));
alu_top ALU08(.src1(src1[8]), .src2(src2[8]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[7]), .operation(ALU_control[1:0]), .result(r[8]), .cout(w[8]));
alu_top ALU09(.src1(src1[9]), .src2(src2[9]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[8]), .operation(ALU_control[1:0]), .result(r[9]), .cout(w[9]));
alu_top ALU10(.src1(src1[10]), .src2(src2[10]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[9]), .operation(ALU_control[1:0]), .result(r[10]), .cout(w[10]));
alu_top ALU11(.src1(src1[11]), .src2(src2[11]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[10]), .operation(ALU_control[1:0]), .result(r[11]), .cout(w[11]));
alu_top ALU12(.src1(src1[12]), .src2(src2[12]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[11]), .operation(ALU_control[1:0]), .result(r[12]), .cout(w[12]));
alu_top ALU13(.src1(src1[13]), .src2(src2[13]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[12]), .operation(ALU_control[1:0]), .result(r[13]), .cout(w[13]));
alu_top ALU14(.src1(src1[14]), .src2(src2[14]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[13]), .operation(ALU_control[1:0]), .result(r[14]), .cout(w[14]));
alu_top ALU15(.src1(src1[15]), .src2(src2[15]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[14]), .operation(ALU_control[1:0]), .result(r[15]), .cout(w[15]));
alu_top ALU16(.src1(src1[16]), .src2(src2[16]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[15]), .operation(ALU_control[1:0]), .result(r[16]), .cout(w[16]));
alu_top ALU17(.src1(src1[17]), .src2(src2[17]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[16]), .operation(ALU_control[1:0]), .result(r[17]), .cout(w[17]));
alu_top ALU18(.src1(src1[18]), .src2(src2[18]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[17]), .operation(ALU_control[1:0]), .result(r[18]), .cout(w[18]));
alu_top ALU19(.src1(src1[19]), .src2(src2[19]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[18]), .operation(ALU_control[1:0]), .result(r[19]), .cout(w[19]));
alu_top ALU20(.src1(src1[20]), .src2(src2[20]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[19]), .operation(ALU_control[1:0]), .result(r[20]), .cout(w[20]));
alu_top ALU21(.src1(src1[21]), .src2(src2[21]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[20]), .operation(ALU_control[1:0]), .result(r[21]), .cout(w[21]));
alu_top ALU22(.src1(src1[22]), .src2(src2[22]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[21]), .operation(ALU_control[1:0]), .result(r[22]), .cout(w[22]));
alu_top ALU23(.src1(src1[23]), .src2(src2[23]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[22]), .operation(ALU_control[1:0]), .result(r[23]), .cout(w[23]));
alu_top ALU24(.src1(src1[24]), .src2(src2[24]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[23]), .operation(ALU_control[1:0]), .result(r[24]), .cout(w[24]));
alu_top ALU25(.src1(src1[25]), .src2(src2[25]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[24]), .operation(ALU_control[1:0]), .result(r[25]), .cout(w[25]));
alu_top ALU26(.src1(src1[26]), .src2(src2[26]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[25]), .operation(ALU_control[1:0]), .result(r[26]), .cout(w[26]));
alu_top ALU27(.src1(src1[27]), .src2(src2[27]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[26]), .operation(ALU_control[1:0]), .result(r[27]), .cout(w[27]));
alu_top ALU28(.src1(src1[28]), .src2(src2[28]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[27]), .operation(ALU_control[1:0]), .result(r[28]), .cout(w[28]));
alu_top ALU29(.src1(src1[29]), .src2(src2[29]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[28]), .operation(ALU_control[1:0]), .result(r[29]), .cout(w[29]));
alu_top ALU30(.src1(src1[30]), .src2(src2[30]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[29]), .operation(ALU_control[1:0]), .result(r[30]), .cout(w[30]));
alu_top ALU31(.src1(src1[31]), .src2(src2[31]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(w[30]), .operation(ALU_control[1:0]), .result(r[31]), .cout(w[31]));
```

3. Experimental result

以下為作業提供之 testbench 的執行結果。


```
Hsuan@MacBook-Air-4 Lab1 % vvp test.vvp

****************************************************
 Congratulation! All data are correct!
****************************************************
```

4. Problems you met and solutions

由於整個大二上學期完全沒有使用 Verilog，因此一開始寫本次作業時對 Verilog 的語法感到非常生疏，不過一邊寫一邊翻閱之前的課堂筆記或 google，最後比較找回了手感。

除了語法需要熟悉外，我覺得此次的 ALU 實作並不困難。在理解 ALU 電路圖的流程與邏輯後，照著電路圖連接、計算對應的輸入輸出訊號即可得出結果；不過稍微不同的是，上課講義中 ALU00~30 與 ALU31 的電路圖、稍有不同，但本次實作我是按照助教提供的模板，使用相同的 alu_top 子模組，並將原本應該在 ALU31 內處理的 overflow 和 set 訊號放到 alu 模組內計算。

5. Summary

透過這次作業詳細了解了 ALU 運作的詳細流程，並基於課程所提的 ALU 電路稍做了一些改動，如將 ALU31 的 set、overflow 移出處理或是另外考慮 reset 等等。透過這次作業也重新熟悉了 Verilog 的語法。