1. Detailed description of the implementation

(1) MUX2to1.v

```
module MUX2to1(
    input      src1,
    input      src2,
    input      select,
    output reg result
    );
/* Write your code HERE */
always @(src1,src2,select) begin
    if(select==0) begin
        result=src1;
    end
    else if(select==1) begin
        result=src2;
    end
end
endmodule
```

根據 select 判斷 result，若 select 為 0，則 result 為 src1;若 select 為 1，result 為 src2。

(2) MUX4to1.v

```
module MUX4to1(
    input           src1,
    input           src2,
    input           src3,
    input           src4,
    input   [2-1:0] select,
    output reg      result
    );
/* Write your code HERE */
always @(src1,src2,src3,src4,select)begin
    case(select)
        2'b00 : begin
            result=src1;
        end
        2'b01 : begin
            result=src2;
        end
        2'b10 : begin
            result=src3;
        end
        default : begin
            result=src4;
        end
    endcase
end
endmodule
```

根據 select 判斷 result，若 select 為 00，則 result 為 src1;若 select 為 01，result 為 src2，若 select 為 10，result 為 src3，若 select 為 11，result 為 src4。

(3) alu_1bit.v

```
`timescale 1ns/1ps

module alu_1bit(
    input               src1,       //1 bit source 1  (input)
    input               src2,       //1 bit source 2  (input)
    input               less,       //1 bit less      (input)
    input               Ainvert,    //1 bit A_invert  (input)
    input               Binvert,    //1 bit B_invert  (input)
    input               cin,        //1 bit carry in  (input)
    input       [2-1:0] operation,  //2 bit operation (input)
    output reg          result,     //1 bit result    (output)
    output reg          cout        //1 bit carry out (output)
    );

/* Write your code HERE */

wire a,b,ans;
MUX2to1 M1(src1,~src1,Ainvert,a);
MUX2to1 M2(src2,~src2,Binvert,b);
MUX4to1 M3((a&b),(a|b),(a^b^cin),less,operation,ans);
always @(*) begin
    case(operation)
        2'b00: begin
            result=ans;
            cout=0;
        end
        2'b01: begin
            result=ans;
            cout=0;
        end
        2'b10: begin
            result=ans;
            cout=(a&b)|(a&cin)|(b&cin);
        end
        2'b11: begin
            result= ans;
            cout=(a&b)|(a&cin)|(b&cin);
        end
    endcase
end
endmodule
```

首先，將 src1、~src1、Ainvert 做為 input 用 MUX2to1 判斷 result(設為 a)，接著將 src2、~src2、Binvert 也做為 input 用 MUX2to1 判斷 result(設為 b)，接著再將 a&b、a|b、a^b^cin、less、operation 作為 input 用 MUX4to1 判斷 result(設為 ans)，接著判斷 operation，若 operation 為 00，則 result 為 ans、cout 為 0;若 operation 為 01，則 result 為 ans、cout 為 0;若 operation 為 10，則 result 為 ans、cout 為 (a&b)|(a&cin)|(b&cin); 若 operation 為 11，則 result 為 ans、cout 為 (a&b)|(a&cin)|(b&cin)。

(4) alu.v

```
`timescale 1ns/1ps


module alu(
    input               rst_n,      // negative reset           (input)
    input       [32-1:0]src1,       // 32 bits source 1         (input)
    input       [32-1:0]src2,       // 32 bits source 2         (input)
    input       [ 4-1:0]ALU_control,// 4 bits ALU control input (input)
    output reg  [32-1:0]result,     // 32 bits result           (output)
    output reg          zero,       // 1 bit when the output is 0, zero must be set (output)
    output reg          cout,       // 1 bit carry out          (output)
    output reg          overflow    // 1 bit overflow           (output)
    );

```

```
/* Write your code HERE */
wire Ainvert=ALU_control[3];
wire Binvert=ALU_control[2];
wire [2-1:0] op=ALU_control[1:0];
reg constantzero=1'b0;

wire [32-1:0] out;
wire c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16,c17,c18,c19,c20,c21,c22,c23,c24,c25,c26,c27,c28,c29,c30,c31;
reg first_cin;

wire f,set;
assign {f,set}=(src1[31]+(~src2[31])+c30);


alu_1bit first(src1[0],src2[0],set,Ainvert,Binvert,first_cin,op,out[0],c0);
alu_1bit one(src1[1],src2[1],constantzero,Ainvert,Binvert,c0,op,out[1],c1);
alu_1bit two(src1[2],src2[2],constantzero,Ainvert,Binvert,c1,op,out[2],c2);
alu_1bit three(src1[3],src2[3],constantzero,Ainvert,Binvert,c2,op,out[3],c3);
alu_1bit four(src1[4],src2[4],constantzero,Ainvert,Binvert,c3,op,out[4],c4);
alu_1bit five(src1[5],src2[5],constantzero,Ainvert,Binvert,c4,op,out[5],c5);
alu_1bit six(src1[6],src2[6],constantzero,Ainvert,Binvert,c5,op,out[6],c6);
alu_1bit seven(src1[7],src2[7],constantzero,Ainvert,Binvert,c6,op,out[7],c7);
alu_1bit eight(src1[8],src2[8],constantzero,Ainvert,Binvert,c7,op,out[8],c8);
alu_1bit nine(src1[9],src2[9],constantzero,Ainvert,Binvert,c8,op,out[9],c9);
alu_1bit ten(src1[10],src2[10],constantzero,Ainvert,Binvert,c9,op,out[10],c10);
alu_1bit eleven(src1[11],src2[11],constantzero,Ainvert,Binvert,c10,op,out[11],c11);
alu_1bit twelve(src1[12],src2[12],constantzero,Ainvert,Binvert,c11,op,out[12],c12);
alu_1bit thirteen(src1[13],src2[13],constantzero,Ainvert,Binvert,c12,op,out[13],c13);
alu_1bit fourteen(src1[14],src2[14],constantzero,Ainvert,Binvert,c13,op,out[14],c14);
alu_1bit fifteen(src1[15],src2[15],constantzero,Ainvert,Binvert,c14,op,out[15],c15);
alu_1bit sixteen(src1[16],src2[16],constantzero,Ainvert,Binvert,c15,op,out[16],c16);
alu_1bit seventeen(src1[17],src2[17],constantzero,Ainvert,Binvert,c16,op,out[17],c17);
alu_1bit eighteen(src1[18],src2[18],constantzero,Ainvert,Binvert,c17,op,out[18],c18);
alu_1bit nineteen(src1[19],src2[19],constantzero,Ainvert,Binvert,c18,op,out[19],c19);
alu_1bit twenty(src1[20],src2[20],constantzero,Ainvert,Binvert,c19,op,out[20],c20);
alu_1bit twnty_one(src1[21],src2[21],constantzero,Ainvert,Binvert,c20,op,out[21],c21);
alu_1bit twenty_two(src1[22],src2[22],constantzero,Ainvert,Binvert,c21,op,out[22],c22);
alu_1bit twenty_three(src1[23],src2[23],constantzero,Ainvert,Binvert,c22,op,out[23],c23);
alu_1bit twenty_four(src1[24],src2[24],constantzero,Ainvert,Binvert,c23,op,out[24],c24);
alu_1bit twenty_five(src1[25],src2[25],constantzero,Ainvert,Binvert,c24,op,out[25],c25);
alu_1bit twenty_six(src1[26],src2[26],constantzero,Ainvert,Binvert,c25,op,out[26],c26);
alu_1bit twenty_seven(src1[27],src2[27],constantzero,Ainvert,Binvert,c26,op,out[27],c27);
alu_1bit twenty_eight(src1[28],src2[28],constantzero,Ainvert,Binvert,c27,op,out[28],c28);
alu_1bit twenty_nine(src1[29],src2[29],constantzero,Ainvert,Binvert,c28,op,out[29],c29);
alu_1bit thirty(src1[30],src2[30],constantzero,Ainvert,Binvert,c29,op,out[30],c30);
alu_1bit thirty_one(src1[31],src2[31],constantzero,Ainvert,Binvert,c30,op,out[31],c31);
```

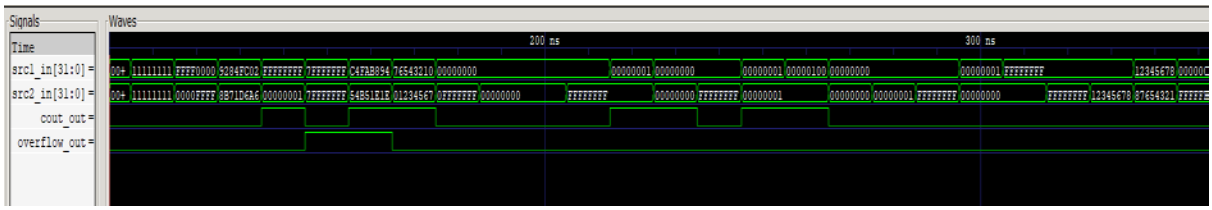
```
always @(*) begin
    if(rst_n==0)begin
        result=0;
        zero=0;
        cout=0;
        overflow=0;
    end
    else begin
        case(ALU_control)
            4'b0000: begin  //and
                first_cin=0;
                cout=0;
                result=out;
                overflow=0;
            end
            4'b0001: begin  //or
                first_cin=0;
                cout=0;
                result=out;
                overflow=0;
            end
            4'b0010: begin  //add
                first_cin=0;
                cout=c31;
                result=out;
                overflow=c30^c31;
            end
```

```
90              4'b0110: begin  //sub
91                  first_cin=1;
92                  cout=c31;
93                  result=out;
94                  overflow=c30^c31;
95              end
96              4'b0111: begin  //slt
97                  first_cin=1;
98                  cout=0;
99                  result=out;
100                 overflow=0;
101             end
102             4'b1100: begin  //nor
103                 first_cin=0;
104                 cout=0;
105                 result=out;
106                 overflow=0;
107             end
108             4'b1101: begin  //nand
109                 first_cin=0;
110                 cout=0;
111                 result=out;
112                 overflow=0;
113             end
114         endcase
115
116         zero= |result;
117         zero= ~zero;
118     end
119 end
120
121 endmodule
```

首先，將 ALU_control[3]設為 Ainvert，ALU_control[2]設為 Binvert，ALU_control[1:0]設為 op，每個 alu 的 cout 為 c0、c1、c2…c31，result 為 out[31:0]，計算 src1[31]+(~src2[31])+c30，取第零位存為 set，除了第一個 alu 代入 alu_1bit(src1[0], src2[0], set, Ainvert, Binvert, first_cin, op, out[0], c0)，其餘皆為 alu_1bit(src1[i], src2[i], 0, Ainvert, Binvert, c(i-1), op, out[i], ci)。接著，若 rst_n 為 0，則 result、zero、cout、overflow 皆為 0，若 rst_n 為 1，根據 ALU_control 判斷，若 ALU_control 為 0000(and)，first_cin 為 0，cout 為 0，result 為 0，overflow 為 0; 若 ALU_control 為 0001(or)，first_cin 為 0，cout 為 0，result 為 out，overflow 為 0; 若 ALU_control 為 0010(add)，first_cin 為 0，cout 為 c31，result 為 out，overflow 為 c30^c31; 若 ALU_control 為 0110(sub)，first_cin 為 1，cout 為 c31，result 為 out，overflow 為 c30^c31; 若 ALU_control 為 0111(slt)，first_cin 為 1，cout 為 0，result 為 out，overflow 為 0; 若 ALU_control 為 1100(nor)，first_cin 為 0，cout 為 0，result 為 out，overflow 為 0; 若 ALU_control 為 1101(nand)，first_cin 為 0，cout 為 0，result 為 out，overflow 為 0。zero 為所有 result 做 or，再做 not。

2. Implementation results

```
C:\Users\user\Desktop\Lab02\Lab02>vvp lab2.vvp
VCD info: dumpfile alu.vcd opened for output.
****************************************************
*             PATTERN RESULT TABLE                 *
****************************************************
* PATTERN *              Result              * ZCV *
****************************************************
*      Congratulation! All data are correct!       *
****************************************************
Correct Count: 30
```



3. Problems encountered and solutions

一開始並沒有正確使用 MUX2to1 和 MUX4to1，所以後續又花了點時間修正，也因為對於 32bitALU 結構的不熟悉，導致在實作 alu.v 時，根據 ALU_control 判斷的內容不夠完整，以及沒有搞懂 slt 的算法，和第一個 alu 要使用 alu_1bit.v 時，關於 less 要帶入的計算方式，花了很多時間才找出最終使用的算法，在過程中也常常因為粗心大意打錯代數名稱和預設值，因此浪費很多時間除錯，關於基礎知識的不熟悉則是需要詳讀講義和蒐集網路的說明資料。關於 alu_1bit.v，代入 MUX4to1 時，result 不能直接作為 output，需要先設置一個 wire，再將此傳為 result，這個步驟也是我一開始一直沒有注意到的地方。同時，對於 iverilog 在終端機的打法也花了很多時間研究，幸虧有熱心的同學指點。