Lab02-report

1. Implementation

- MUX:

  Implemented using case.

```
module MUX2to1(
    input       src1,
    input       src2,
    input       select,
    output reg result
    );
 /* Write your code HERE */
always @(src1, src2, select) begin
    case(select)
        1'b0:
            result <= src1;
        1'b1:
            result <= src2;
        default:
            result <= result;
    endcase
end

endmodule
```

```
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:
            result <= src1;
        2'b01:
            result <= src2;
        2'b10:
            result <= src3;
        2'b11:
            result <= src4;
        default:
            result <= result;
    endcase
end

endmodule
```

- ALU_1bit

  I added an output “set” to get the set signal.

```
3   module alu_1bit(
4       input               src1,       //1 bit source 1  (input)
5       input               src2,       //1 bit source 2  (input)
6       input               less,       //1 bit less      (input)
7       input               Ainvert,    //1 bit A_invert  (input)
8       input               Binvert,    //1 bit B_invert  (input)
9       input               cin,        //1 bit carry in  (input)
10      input       [2-1:0] operation,  //2 bit operation (input)
11      output reg          result,     //1 bit result    (output)
12      output reg          cout,       //1 bit carry out (output)
13      output reg          set         //1 bit set
14      );
```

W1 and W2 are the result of the two MUX, indicating if A or B should be inverted.

```
17  wire A_inv;
18  wire B_inv;
19
20  wire W1;
21  wire W2;
22
23  assign A_inv = ~src1;
24  assign B_inv = ~src2;
25
26  MUX2to1 MUX2_1(.src1(src1), .src2(A_inv), .select(Ainvert), .result(W1));
27  MUX2to1 MUX2_2(.src1(src2), .src2(B_inv), .select(Binvert), .result(W2));
```

W3 is AND operation, W4 is OR operation, and W5 is a full adder.

```
29  reg W3;
30  reg W4;
31  reg W5;
32
33  always @(W1, W2, cin) begin
34      W3 <= W1 & W2;
35      W4 <= W1 | W2;
36      W5 <= (W1 ^ W2) ^ cin;
37  end
```

Use a MUX to select result, calculate carry out, then get the set value from the adder output.

```
41  MUX4to1 MUX4(.src1(W3), .src2(W4), .src3(W5), .src4(less), .select(operation), .result(res));
42
43  always @(W1, W5, cin, res, operation) begin
44      result <= res;
45      set <= W5;
46      cout <= ((W1 & cin) | (W2 & cin) | (W1 & W2));
47  end
```

- ALU

Declaration of all variables.

```
16    wire [31:0] carry;
17    wire carry32;
18    reg carry0;
19
20    reg [1:0] operation;
21
22    wire [31:0] res;
23
24    wire set;
25    reg s;
26
27    reg Ainvert;
28    reg Binvert;
```

A case statement to classify the instruction and set the corresponding inputs.

```
30  always @(ALU_control) begin
31      case(ALU_control)
32          4'b0000: begin
33              Ainvert <= 0;
34              Binvert <= 0;
35              operation <= 2'b00;
36          end
37          4'b0001: begin
38              Ainvert <= 0;
39              Binvert <= 0;
40              operation <= 2'b01;
41          end
42          4'b0010: begin
43              Ainvert <= 0;
44              Binvert <= 0;
45              operation <= 2'b10;
46              carry0 <= 1'b0;
47          end
48          4'b0110: begin
49              Ainvert <= 0;
50              Binvert <= 1;
51              operation <= 2'b10;
52              carry0 <= 1'b1;
53          end
54          4'b0111: begin
55              Ainvert <= 0;
56              Binvert <= 1;
57              operation <= 2'b11;
58              carry0 <= 1'b1;
59          end
60          4'b1100: begin
61              Ainvert <= 0;
62              Binvert <= 0;
63              operation <= 2'b01;
64          end
65          4'b1101: begin
66              Ainvert <= 0;
67              Binvert <= 0;
68              operation <= 2'b00;
69          end
70          default: begin
71              Ainvert <= 0;
72              Binvert <= 0;
73              operation <= 2'b00;
74              carry0 <= 1'b0;
75          end
76      endcase
77  end
```

When the instruction is NAND or NOR, the result should be inverted, otherwise, output directly.

```
79  always @(res, ALU_control) begin
80      case(ALU_control)
81          4'b1100:
82              result <= ~res;
83          4'b1101:
84              result <= ~res;
85          default:
86              result <= res;
87      endcase
88  end
```

Calculating the ZCV flags, note that C flag only set when the operation is ADD.

```
90  always @(carry32) begin
91      case(operation)
92          2'b10:
93              cout <= carry32;
94          default:
95              cout <= 1'b0;
96      endcase
97  end
98
99  always @(result) begin
100     zero <= ~|result;
101     overflow <= (((~src1[31] & ~src2[31] & carry[31]) | (src1[31] & src2[31] & ~carry[31])) & (ALU_control == 4'b0010))
102                 | (((~src1[31] & src2[31] & carry[31]) | (src1[31] & ~src2[31] & ~carry[31])) & (ALU_control == 4'b0110));
103 end
```

Get the set value from the last ALU and send back to the first one.

```
105 always @(set) begin
106     s <= set;
107 end
```

The declaration of all ALUs. All ALU except the first and the last one is declared in the for loop.

All ALUs except the first one needs a 0 at the less input.

The last ALU’s set value is connected to the first one’s less input.

```
109   alu_1bit alu0(.result(res[0]),
110               .cout(carry[1]),
111               .src1(src1[0]),
112               .src2(src2[0]),
113               .Ainvert(Ainvert),
114               .Binvert(Binvert),
115               .operation(operation),
116               .cin(carry0),
117               .less(s));
118
119   alu_1bit alu31(.result(res[31]),
120               .cout(carry32),
121               .src1(src1[31]),
122               .src2(src2[31]),
123               .Ainvert(Ainvert),
124               .Binvert(Binvert),
125               .operation(operation),
126               .cin(carry[31]),
127               .less(1'b0),
128               .set(set));
129
130   genvar i;
131   generate
132       for(i = 1; i < 31; i = i + 1) begin
133           alu_1bit alu(.result(res[i]),
134               .cout(carry[i + 1]),
135               .src1(src1[i]),
136               .src2(src2[i]),
137               .Ainvert(Ainvert),
138               .Binvert(Binvert),
139               .operation(operation),
140               .cin(carry[i]),
141               .less(1'b0));
142       end
143   endgenerate
```

2. Implementation results

```
d@d-VirtualBox:/media/sf_Downloads/Lab02/Lab02$ ./lab2
VCD info: dumpfile alu.vcd opened for output.
************************************************
*             PATTERN RESULT TABLE             *
************************************************
* PATTERN *              Result            * ZCV *
************************************************
*      Congratulation! All data are correct!     *
************************************************
Correct Count: 30
```

3. Problem encountered

   I. The results of SLT instruction are always wrong.

   After using GTKwave, I found that carry value of some ALUs are wrong, but I didn't find any problem of the calculation process. Then I checked the whole code of 1-bit ALU, I found I set the carry only output when the operation is 2'b01. After modifying the statement, all results are correct.