Computer Organization

0810749 張君實 資工系

Lab2- 32-bit ALU

Architecture diagram:

不管是 alu 還是 ALU_1bit 的 input 和 output 都沒動過，但就是因為沒動過，少了一個 less 的 input，所以我的設計和講義上有點不一樣。在 alu 中，我會先把 32 個 ALU_1bit 的 cout 和 result 結果先分別存在 c[31:0]和 temp_result [31:0]的 wire 中，再根據 ALU control input 決定要怎麼行動。如果是 SLT 以外的指令，因為不用 less 那個 input，所以在 ALU_1bit 中我就可以算好 result 和 cout，丟到 c[31:0]和 temp_result [31:0]後再 output 出去，這時也順便運算 zero 和 overflow。如果是 SLT 的指令，因為少了 less，所以判斷是否小於 0 這件事只能在 alu 做，而 ALU_1bit 輸出的 result 和 cout 是減法後的結果，因為結果後來存到了 c[31:0]和 temp_result [31:0]上，這時程式可以判斷 temp_result[31]是否為 1 來決定 output result，然後 cout 默認為 0，zero 是用判斷後的 output result 決定，overflow 則是默認為 0。

Detailed description of the implementation:

先介紹 ALU_1bit

```
6
7   module ALU_1bit(
8       input               src1,       //1 bit source 1  (input)
9       input               src2,       //1 bit source 2  (input)
10      input               Ainvert,    //1 bit A_invert  (input)
11      input               Binvert,    //1 bit B_invert  (input)
12      input               Cin,        //1 bit carry in  (input)
13      input       [2-1:0] operation,  //2 bit operation (input)
14      output reg          result,     //1 bit result    (output)
15      output reg          cout       //1 bit carry out (output)
16      );
17
18  /* Write your code HERE */
19  always@(*)
20  case ({Ainvert,Binvert,operation})
21      4'b0000:
22          begin
23              result <= src1 & src2;
24              cout <= 1'b0;
25          end
26      4'b0001:
27          begin
28              result <= src1 | src2;
29              cout <= 1'b0;
30          end
31      4'b0010:
32          begin
33              {cout,result} <= src1 + src2 + Cin;
34          end
35      4'b0110:
36          begin
37              cout <= (src1 & ~src2) | (src1 & Cin) | (~src2 & Cin);
38              result <= src1 ^ ~src2 ^ Cin;
39          end
40      4'b0111:
41          begin
42              cout <= (src1 & ~src2) | (src1 & Cin) | (~src2 & Cin);
43              result <= src1 ^ ~src2 ^ Cin;
44          end
```

```
45      4'b1100:
46          begin
47              result <= ~ src1 & ~src2 ;
48              cout <= 1'b0;
49          end
50      4'b1101:
51          begin
52              result <= ~src1 | ~src2 ;
53              cout <= 1'b0;
54          end
55      default:
56              begin
57                  cout <= 1'b0;
58                  result <= 1'b0;
59              end
60  endcase
61  endmodule
62
```

一開始先判斷指令為何，再分別做運算。

除了 SLT 我在這裡是輸出相減後的結果，剩下的部分都是照著講義的步驟。

接著來看 alu

```
7   module alu(
8       input                   rst_n,         // negative reset            (input)
9       input        [32-1:0]   src1,          // 32 bits source 1          (input)
10      input        [32-1:0]   src2,          // 32 bits source 2          (input)
11      input        [ 4-1:0]   ALU_control,   // 4 bits ALU control input  (input)
12      output reg   [32-1:0]   result,        // 32 bits result            (output)
13      output reg              zero,          // 1 bit when the output is 0, zero must be set (output)
14      output reg              cout,          // 1 bit carry out           (output)
15      output reg              overflow       // 1 bit overflow            (output)
16      );
17
18  /* Write your code HERE */
19  wire [31:0]c;
20  wire [31:0]temp_result;
21
22  ALU_1bit    alu1_bit0(  src1[0],src2[0],ALU_control[3],ALU_control[2],ALU_control[2],ALU_control[1:0],temp_result[0],c[0]
23  ALU_1bit    alu1_bit1(  src1[1],src2[1],ALU_control[3],ALU_control[2],c[0],ALU_control[1:0],temp_result[1],c[1]);
24  ALU_1bit    alu1_bit2(  src1[2],src2[2],ALU_control[3],ALU_control[2],c[1],ALU_control[1:0],temp_result[2],c[2]);
25  ALU_1bit    alu1_bit3(  src1[3],src2[3],ALU_control[3],ALU_control[2],c[2],ALU_control[1:0],temp_result[3],c[3]);
26  ALU_1bit    alu1_bit4(  src1[4],src2[4],ALU_control[3],ALU_control[2],c[3],ALU_control[1:0],temp_result[4],c[4]);
27  ALU_1bit    alu1_bit5(  src1[5],src2[5],ALU_control[3],ALU_control[2],c[4],ALU_control[1:0],temp_result[5],c[5]);
28  ALU_1bit    alu1_bit6(  src1[6],src2[6],ALU_control[3],ALU_control[2],c[5],ALU_control[1:0],temp_result[6],c[6]);
29  ALU_1bit    alu1_bit7(  src1[7],src2[7],ALU_control[3],ALU_control[2],c[6],ALU_control[1:0],temp_result[7],c[7]);
30  ALU_1bit    alu1_bit8(  src1[8],src2[8],ALU_control[3],ALU_control[2],c[7],ALU_control[1:0],temp_result[8],c[8]);
31  ALU_1bit    alu1_bit9(  src1[9],src2[9],ALU_control[3],ALU_control[2],c[8],ALU_control[1:0],temp_result[9],c[9]);
32  ALU_1bit    alu1_bit10( src1[10],src2[10],ALU_control[3],ALU_control[2],c[9],ALU_control[1:0],temp_result[10],c[10]);
33  ALU_1bit    alu1_bit11( src1[11],src2[11],ALU_control[3],ALU_control[2],c[10],ALU_control[1:0],temp_result[11],c[11]);
34  ALU_1bit    alu1_bit12( src1[12],src2[12],ALU_control[3],ALU_control[2],c[11],ALU_control[1:0],temp_result[12],c[12]);
35  ALU_1bit    alu1_bit13( src1[13],src2[13],ALU_control[3],ALU_control[2],c[12],ALU_control[1:0],temp_result[13],c[13]);
36  ALU_1bit    alu1_bit14( src1[14],src2[14],ALU_control[3],ALU_control[2],c[13],ALU_control[1:0],temp_result[14],c[14]);
37  ALU_1bit    alu1_bit15( src1[15],src2[15],ALU_control[3],ALU_control[2],c[14],ALU_control[1:0],temp_result[15],c[15]);
38  ALU_1bit    alu1_bit16( src1[16],src2[16],ALU_control[3],ALU_control[2],c[15],ALU_control[1:0],temp_result[16],c[16]);
39  ALU_1bit    alu1_bit17( src1[17],src2[17],ALU_control[3],ALU_control[2],c[16],ALU_control[1:0],temp_result[17],c[17]);
40  ALU_1bit    alu1_bit18( src1[18],src2[18],ALU_control[3],ALU_control[2],c[17],ALU_control[1:0],temp_result[18],c[18]);
41  ALU_1bit    alu1_bit19( src1[19],src2[19],ALU_control[3],ALU_control[2],c[18],ALU_control[1:0],temp_result[19],c[19]);
42  ALU_1bit    alu1_bit20( src1[20],src2[20],ALU_control[3],ALU_control[2],c[19],ALU_control[1:0],temp_result[20],c[20]);
43  ALU_1bit    alu1_bit21( src1[21],src2[21],ALU_control[3],ALU_control[2],c[20],ALU_control[1:0],temp_result[21],c[21]);
44  ALU_1bit    alu1_bit22( src1[22],src2[22],ALU_control[3],ALU_control[2],c[21],ALU_control[1:0],temp_result[22],c[22]);
```

```
45  ALU_1bit    alu1_bit23( src1[23],src2[23],ALU_control[3],ALU_control[2],c[22],ALU_control[1:0],temp_result[23],c[23]);
46  ALU_1bit    alu1_bit24( src1[24],src2[24],ALU_control[3],ALU_control[2],c[23],ALU_control[1:0],temp_result[24],c[24]);
47  ALU_1bit    alu1_bit25( src1[25],src2[25],ALU_control[3],ALU_control[2],c[24],ALU_control[1:0],temp_result[25],c[25]);
48  ALU_1bit    alu1_bit26( src1[26],src2[26],ALU_control[3],ALU_control[2],c[25],ALU_control[1:0],temp_result[26],c[26]);
49  ALU_1bit    alu1_bit27( src1[27],src2[27],ALU_control[3],ALU_control[2],c[26],ALU_control[1:0],temp_result[27],c[27]);
50  ALU_1bit    alu1_bit28( src1[28],src2[28],ALU_control[3],ALU_control[2],c[27],ALU_control[1:0],temp_result[28],c[28]);
51  ALU_1bit    alu1_bit29( src1[29],src2[29],ALU_control[3],ALU_control[2],c[28],ALU_control[1:0],temp_result[29],c[29]);
52  ALU_1bit    alu1_bit30( src1[30],src2[30],ALU_control[3],ALU_control[2],c[29],ALU_control[1:0],temp_result[30],c[30]);
53  ALU_1bit    alu1_bit31( src1[31],src2[31],ALU_control[3],ALU_control[2],c[30],ALU_control[1:0],temp_result[31],c[31]);
54
55  always@(*)begin
56      if(rst_n)begin
57          if (ALU_control[3:0]==4'b0111) begin
58              if(temp_result[31]==1'b1)
59                  result = 32'b00000000000000000000000000000001;
60              else
61                  result = 32'b00000000000000000000000000000000;
62              if(|result == 1'b0)
63                  zero = 1'b1;
64              else
65                  zero = 1'b0;
66              overflow = 1'b0;
67              cout = 1'b0;
68          end
69          else begin
70              if(|temp_result == 1'b0)
71                  zero <= 1'b1;
72              else
73                  zero <= 1'b0;
74              cout <= c[31];
75              overflow <= c[30] ^ c[31];
76              result <= temp_result;
77          end
78      end
79      else begin
80          zero <= 1'b0;
81          cout <= 1'b0;
82          overflow <= 1'b0;
83          result = 32'b00000000000000000000000000000000;
```

```
84      end
85  end
86
87  endmodule
88
```

每次讓 ALU_1bit 運算時，會將 output 存在 c[31:0]和 temp_result [31:0]上，然後接著判斷 rst_n 有沒有開，沒開就全歸零，有開就接著判斷是不是 SLT，不是 SLT 的話就將 c[31:0]和 temp_result [31:0]直接丟到 cout 和 result 中，然後看 temp_result 的所有 bit nand 決定 zero，還有看 c[30] xor c[31]決定 overflow。如果是 SLT 的話，就判斷 temp_result[31]是不是 1’b1 決定 result 結果，至於 zero 則是看 result 後的結果判斷，cout 和 overflow 默認為 1’b0。

Implementation results:

```
D:\>cd using/iverilog/bin

D:\using\iverilog\bin>iverilog -o test.vvp alu.v ALU_1bit.v testbench.v

D:\using\iverilog\bin>vvp test.vvp
************************************************
*             PATTERN RESULT TABLE               *
************************************************
* PATTERN *             Result             * ZCV *
************************************************
*      Congratulation! All data are correct!      *
************************************************
Correct Count: 30
testbench.v:95: $finish called at 415000 (1ps)
```

安裝 iverilog，叫出 cmd，把助教給的 testbench 和 alu.v 和 ALU_1bit.v 一起編譯後執行，得到測資全過的結果。

Problems encountered and solutions:

太久沒用 verilog，語法和命名真的寫的亂七八糟，很容易接錯線，還好後來經過耐心，最後還是把 bug 一個一個 debug 出來了。喔對，這也是我第一次使用 windows 的 cmd 進行 compile，之前我是用 modelsim，雖然都很難用啦。不過至少讓我學會怎麼簡單操作 windows cmd，也是一件好事。

Lesson learnt (if any):

讓我更了解 alu 的邏輯閘和實作流程。

Comment:

這次的 lab 讓我回想起一年半前碰的 verilog，算是有收穫吧。