```
1   // Eugene Ngo
2   // 4/20/2023
3   // CSE 469
4   // Lab 1, Task 3
5
6   // A module to implement a 32 bit ARM-based ALU.
7   // This contains the central logic for calling on submodules
8   // that make up the ALU.
9
10  module alu(input logic [31:0] a, b,
11             input logic [1:0] ALUControl,
12             output logic [31:0] Result,
13             output logic [3:0] ALUFlags);
14     // 00 = add
15     // 01 = subtract
16     // 10 = AND
17     // 11 = OR
18
19     logic [31:0] carries;
20
21     // This initial call to the singleALU is to set the carry input for the
22     // genvar statements.
23     singleALU setCarries (.a(a[0]), .b(b[0]), .carryIn(ALUControl[0]),
24                           .ALUControl(ALUControl), .Result(Result[0]),
25                           .carryOut(carries[0]));
26
27     // The genvar statements break down the 32 bit inputs and sends them to
28     // 1 bit ALUs that add and subtract them based on the input control signal.
29     // The carry outs are processed and sent to the next bit that is covered,
30     // thus linking them and creating the actual 32 bit output value.
31     genvar i;
32     generate
33        for (i = 1; i < 32; i++) begin: ALUPipeline
34           singleALU results (.a(a[i]), .b(b[i]), .carryIn(carries[i - 1]),
35                              .ALUControl(ALUControl), .Result(Result[i]),
36                              .carryOut(carries[i]));
37        end // end loop
38     endgenerate // end generate
39
40     // Setting flags:
41     // Overflow is xor of carry[30], carry[31]
42     xor overFlowCheck (ALUFlags[0], carries[31], carries[30]);
43     // Carryout is the last carry.
44     assign ALUFlags[1] = carries[31];
45
46     // Inefficient and bad style. RTL would be better.
47     // Checks if any one of the bits within the 32-bit outputted value
48     // contains any 1s. If there is a single 1, the output zero flag is not
49     // raised.
50     nor zeroChecker
51        (ALUFlags[2], Result[31], Result[30], Result[29], Result[28],
52         Result[27], Result[26], Result[25], Result[24], Result[23],
53         Result[22], Result[21], Result[20], Result[19], Result[18],
54         Result[17], Result[16], Result[15], Result[14], Result[13],
55         Result[12], Result[11], Result[10], Result[9], Result[8],
56         Result[7], Result[6], Result[5], Result[4], Result[3],
57         Result[2], Result[1], Result[0]);
58
59     assign ALUFlags[3] = Result[31];
60
61  endmodule
62
63  // alu_testbench tests the behaviors of the ALU by running a .tv file
64  // through it. The results are compared to actual expected values to
65  // determine if the functionality of the ALU is correct.
66
67  module alu_testbench();
68     logic [31:0] a,b;
69     logic [1:0] ALUControl;
70     logic [31:0] Result;
71     logic [3:0] ALUFlags;
72     logic clk;
73     logic [103:0] testvectors [1000:0];
```

```
74
75      // Calls on the ALU module to test it.
76      alu dut (.a(a), .b(b), .ALUControl(ALUControl), .Result(Result),
77               .ALUFlags(ALUFlags));
78
79      parameter CLOCK_PERIOD = 100;
80
81      initial clk = 1;
82
83      // Generates a clock signal
84      always begin
85            #(CLOCK_PERIOD/2);
86            clk = ~clk;
87      end
88
89      // Reads through the .tv file and individually and test if the
90      // vector file values are the same as the values generated
91      // by the alu files.
92      initial begin
93         $readmemh("alu.tv", testvectors);
94
95         for (int i = 0; i < 20; i = i + 1) begin
96            {ALUControl, a, b, Result, ALUFlags} = testvectors[i];
97            @(posedge clk);
98         end // end loop
99      end // end initial
100  endmodule
101
```