```
1   // Nattapon Oonlamom
2   // 04/07/2023
3   // EE 469
4   // Lab 1: ALU and Register (Task 3)
5
6   /* ALU - Artihmatic Logic Unit: (32-bit)
7    * The ALU is able to Add, Subtract, Compare (AND, OR)
8    * The ALU also verify Negative, Zero, Carry, oVerflow
9    *
10   * Overall Inputs/Outputs listed below:
11   *    Inputs:  32-bit a, b
12   *             2-bit  ALUControl
13   *    Outputs: 32-bit Result
14   *             4-bit  ALUFlags
15   */
16  module alu (input  logic [31:0] a, b,
17              input  logic [1:0]  ALUControl,
18              output logic [31:0] Result,
19              output logic [3:0]  ALUFlags);
20
21     // Logic for ALU
22     logic [31:0] sum;   // overall sum
23     logic [31:0] temp;  // updated b
24     logic [32:0] temp1; // sum with carry
25     logic c_out, temp2;  // carry out and most-sig-bit
26
27     // Assign addition with updated b and a carry
28     assign temp  = (ALUControl == 2'b01) ? ~b:b;
29     assign temp2 = (ALUControl[0] && (temp != 32'b0) && (temp != ~32'b0)) ? 1'b1:1'b0;  //
   padded new b
30     assign temp1 = ({1'b0, a} + {temp2, temp} + ALUControl[0]); // adder
31     assign sum   = temp1[31:0];
32     assign c_out = (~ALUControl[1]) ? temp1[32] : 1'b0;
33
34     // Combinational logics for ALU
35     always_comb begin
36        case (ALUControl)
37           2'b00: Result = sum;   // Add
38           2'b01: Result = sum;   // Sub
39           2'b10: Result = a & b; // AND
40           2'b11: Result = a | b; // OR
41        endcase
42     end
43
44     assign ALUFlags[3] = Result[31];                                  // Negative Flag  (N)
45     assign ALUFlags[2] = (Result == 32'b0);                           // Zero Flag      (Z)
46     assign ALUFlags[1] = c_out;                                       // Carry Flag     (C)
47     assign ALUFlags[0] = ((~ALUControl[1]) &&                         // oVerflow flag  (V)
48                           (~(ALUControl[0] ^ a[31] ^ b[31])) &&
49                           (a[31] ^ Result[31]));
50
51  endmodule
```

```
1   0_00000000_00000000_00000000_4
2   0_00000000_FFFFFFFF_FFFFFFFF_8
3   0_00000001_FFFFFFFF_00000000_6
4   0_000000FF_00000001_00000100_0
5   1_00000000_00000000_00000000_6
6   1_00000000_FFFFFFFF_00000001_0
7   1_00000001_00000001_00000000_4
8   1_00000100_00000001_000000FF_0
9   2_FFFFFFFF_FFFFFFFF_FFFFFFFF_8
10  2_FFFFFFFF_12345678_12345678_0
11  2_12345678_87654321_02244220_0
12  2_00000000_FFFFFFFF_00000000_4
13  3_FFFFFFFF_FFFFFFFF_FFFFFFFF_8
14  3_12345678_87654321_97755779_8
15  3_00000000_FFFFFFFF_FFFFFFFF_8
16  3_00000000_00000000_00000000_4
17
```

```
1   // Nattapon Oonlamom
2   // 04/07/2023
3   // EE 469
4   // Lab 1: ALU and Register (Task 3)
5
6   /*
7    * Testbench for the alu.sv to test for correctly functioning ALUControl
8    * and displaying the correct result as expected
9    */
10  module alu_testbench();
11     // logic to simulate
12     logic clk;
13     logic [31:0]  a, b;
14     logic [1:0]   ALUControl;
15     logic [31:0]  Result;
16     logic [3:0]   ALUFlags;
17     logic [103:0] testvectors [1000:0];
18
19     // device under test
20     alu dut(.*);
21
22     // clock setup
23     parameter clock_period = 100;
24
25     initial clk = 1;
26     always begin
27        #(clock_period /2);
28        clk <= ~clk;
29     end
30
31     // initial simulation
32     initial begin
33        $readmemh("alu.tv", testvectors);
34
35        for(int i = 0; i < 20; i = i + 1) begin
36           {ALUControl, a, b, Result, ALUFlags} = testvectors[i];   @(posedge clk);
37        end
38     end
39  endmodule
```

```
1   // Nattapon Oonlamom
2   // 04/07/2023
3   // EE 469
4   // Lab 1: ALU and Register (Task 2)
5
6   /* 16x32 Register file module for specified data and address bus widths.
7    * 2 Asynchronous read port (read_addr1 -> read_data1, read_addr2 -> read_data2)
8    * and synchronous write port (write_data -> write_addr if wr_en)
9    *
10   * Overall Inputs/Outputs listed below:
11   *    Inputs:  32-bit write_data
12   *             4-bit  write_addr,read_addr1, read_addr2
13   *             1-bit  clk
14   *    Outputs: 32-bit read_data1, read_data2
15   */
16  module reg_file(input  logic clk, wr_en,
17                      input  logic [31:0] write_data,
18                      input  logic [3:0]  write_addr,
19                      input  logic [3:0]  read_addr1, read_addr2,
20                      output logic [31:0] read_data1, read_data2);
21
22     // array declaration (registers)
23     logic [15:0] RAM [31:0];
24
25     // write operation (synchronous)
26     always_ff @(posedge clk) begin
27        if (wr_en) begin
28           RAM[write_addr] <= write_data;
29        end
30     end
31
32     // read operation (asynchronous)
33     assign read_data1 = RAM[read_addr1];
34     assign read_data2 = RAM[read_addr2];
35
36  endmodule // reg_file
37
38
```

```
1   // Nattapon Oonlamom
2   // 04/07/2023
3   // EE 469
4   // Lab 1: ALU and Register (Task 2)
5
6   /*
7    * Testbench for the reg_file.sv to test for correctly functions read and write
8    */
9   module reg_file_testbench();
10     // logic to simulate
11     logic clk, wr_en;
12     logic [31:0] write_data;
13     logic [3:0]  write_addr;
14     logic [3:0]  read_addr1, read_addr2;
15     logic [31:0] read_data1, read_data2;
16
17     // device under test
18     reg_file dut(.*);
19
20     // clock setup
21     parameter clock_period = 100;
22
23     initial begin
24        clk <= 0;
25        forever #(clock_period /2) clk <= ~clk;
26     end
27
28     // initial simulation
29     initial begin
30        write_data <= 4'b0001; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
31        write_data <= 4'b0001; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
32        write_data <= 4'b0001; wr_en <= 1; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
33        write_data <= 4'b0010; wr_en <= 1; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
34        write_data <= 4'b0010; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
35        write_data <= 4'b0010; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b00;
   write_addr <= 2'b00;     @(posedge clk);
36        write_data <= 4'b0011; wr_en <= 1; read_addr1 <= 2'b00; read_addr2 <= 2'b01;
   write_addr <= 2'b01;     @(posedge clk);
37        write_data <= 4'b0011; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b01;
   write_addr <= 2'b01;     @(posedge clk);
38        write_data <= 4'b0011; wr_en <= 0; read_addr1 <= 2'b00; read_addr2 <= 2'b01;
   write_addr <= 2'b01;     @(posedge clk);
39        $stop;
40     end
41
42  endmodule
```