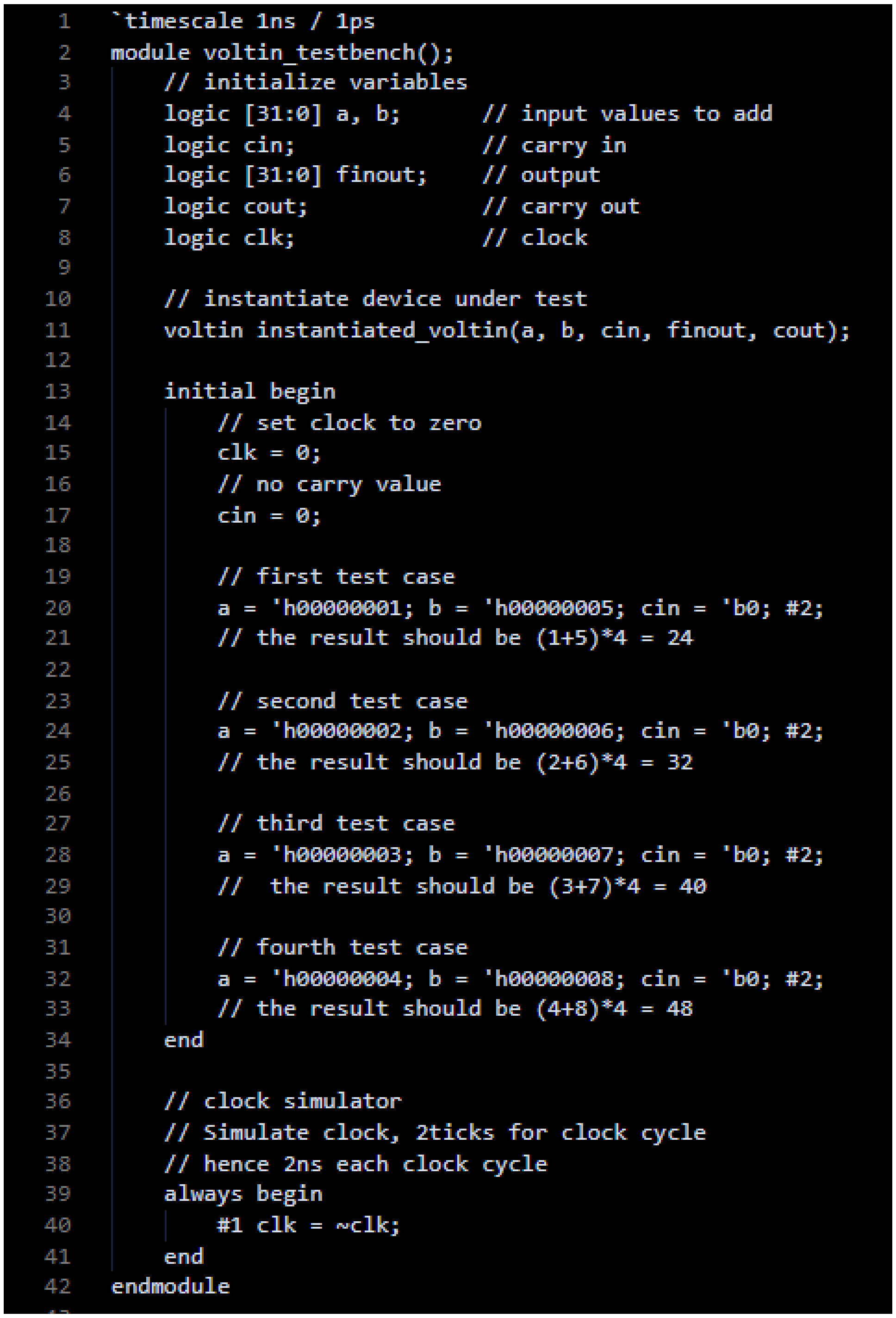
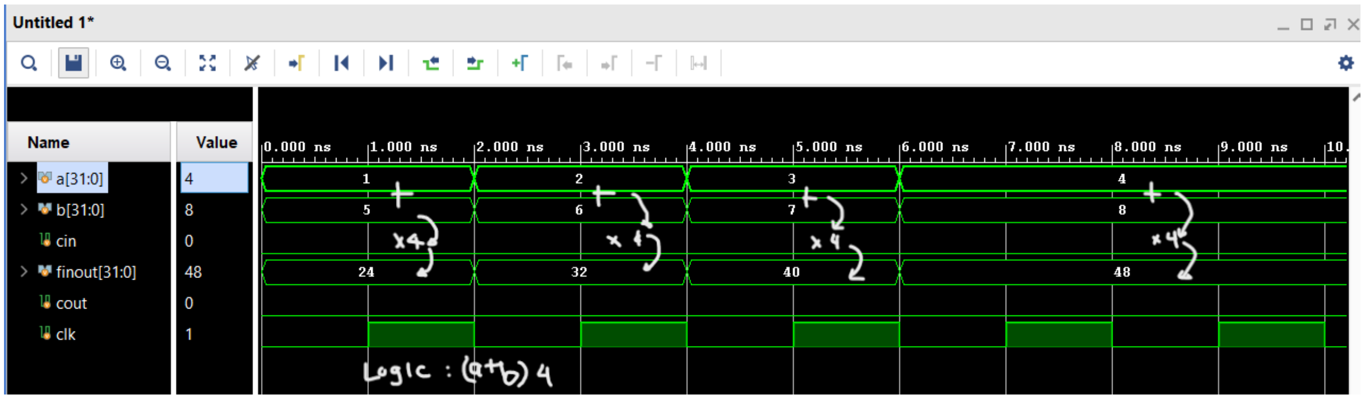
1a.



```
`timescale 1ns / 1ps
module voltin_testbench();
    // initialize variables
    logic [31:0] a, b;      // input values to add
    logic cin;              // carry in
    logic [31:0] finout;    // output
    logic cout;             // carry out
    logic clk;              // clock

    // instantiate device under test
    voltin instantiated_voltin(a, b, cin, finout, cout);

    initial begin
        // set clock to zero
        clk = 0;
        // no carry value
        cin = 0;

        // first test case
        a = 'h00000001; b = 'h00000005; cin = 'b0; #2;
        // the result should be (1+5)*4 = 24

        // second test case
        a = 'h00000002; b = 'h00000006; cin = 'b0; #2;
        // the result should be (2+6)*4 = 32

        // third test case
        a = 'h00000003; b = 'h00000007; cin = 'b0; #2;
        //  the result should be (3+7)*4 = 40

        // fourth test case
        a = 'h00000004; b = 'h00000008; cin = 'b0; #2;
        // the result should be (4+8)*4 = 48
    end

    // clock simulator
    // Simulate clock, 2ticks for clock cycle
    // hence 2ns each clock cycle
    always begin
        #1 clk = ~clk;
    end
endmodule
```

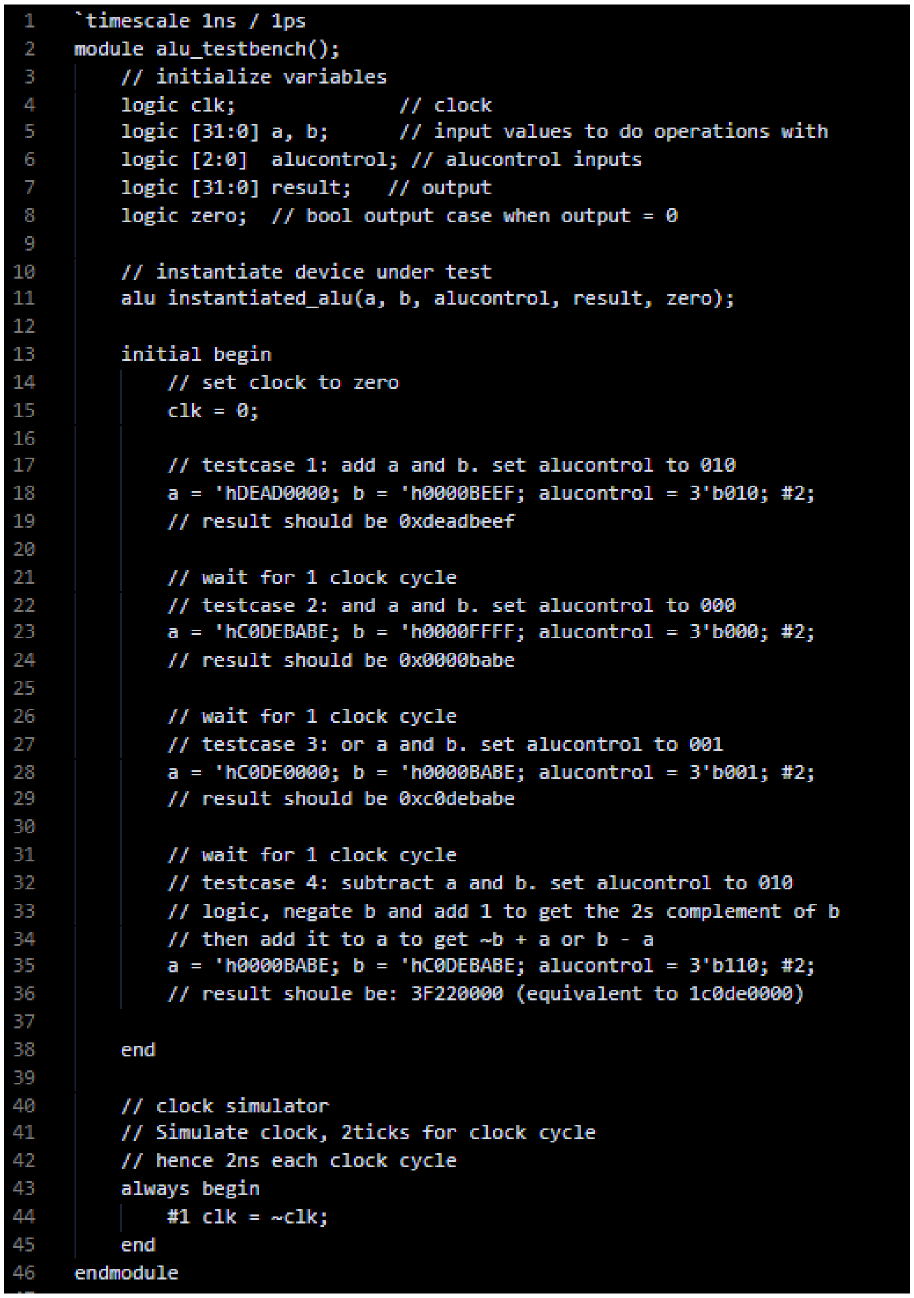
1b.



2a.

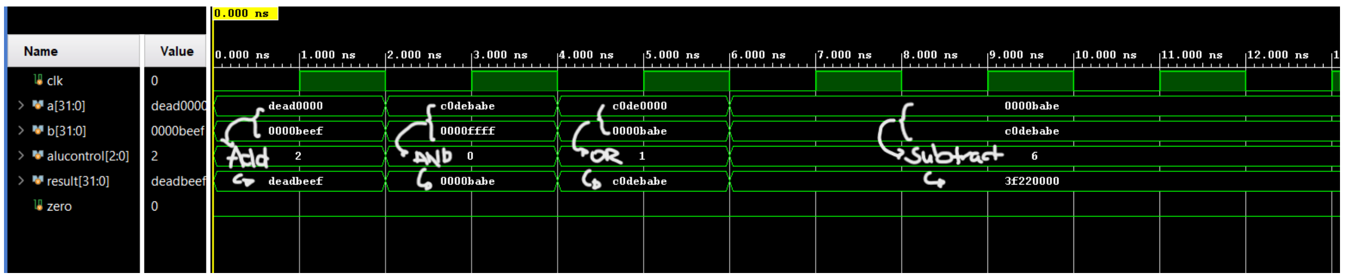
Operation	**a**	**b**	**alucontrol**	expected result
addition	**0xDEAD0000**	**0x0000BEEF**	**0b010**	0xDEADBEEF
and	**0xC0DEBABE**	**0x0000FFFF**	**0b000**	0x0000BABE
or	**0xC0DE0000**	**0x0000BABE**	**0b001**	0xC0DEBABE
subtraction	**0x0000BABE**	**0xCODEBABE**	**0b110**	0x3F220000

2b.

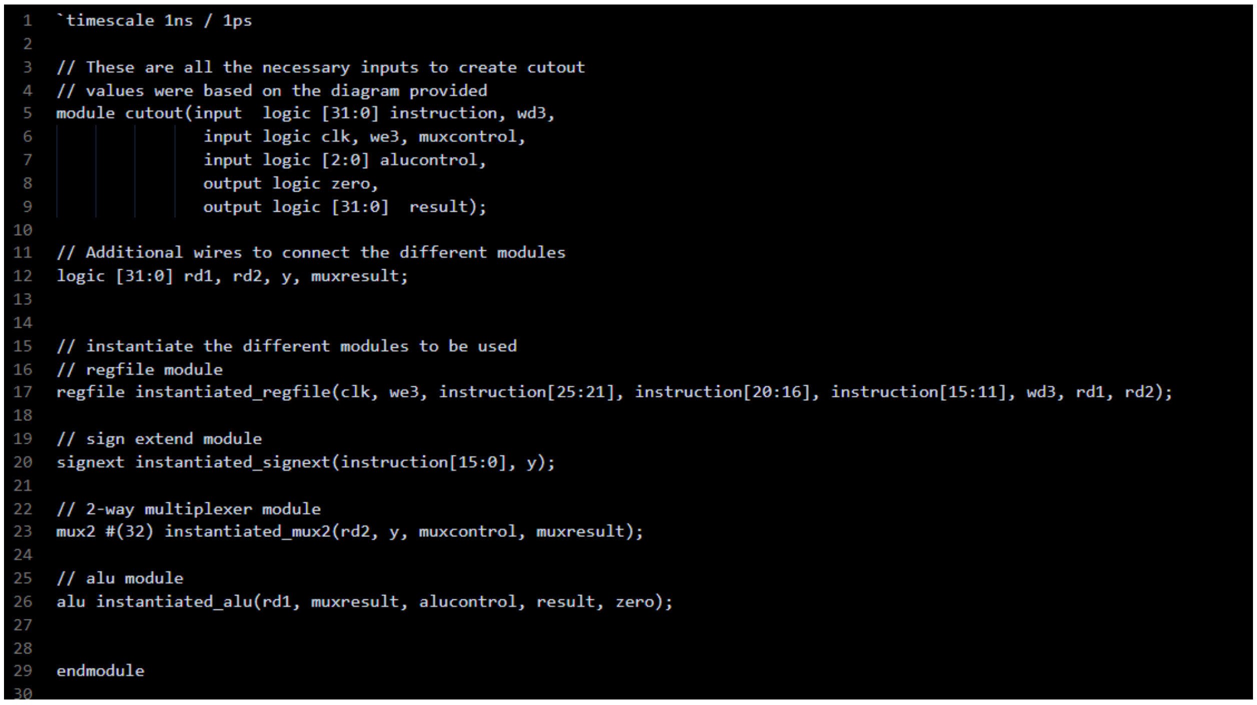


```
`timescale 1ns / 1ps
module alu_testbench();
    // initialize variables
    logic clk;              // clock
    logic [31:0] a, b;      // input values to do operations with
    logic [2:0]  alucontrol; // alucontrol inputs
    logic [31:0] result;    // output
    logic zero;  // bool output case when output = 0

    // instantiate device under test
    alu instantiated_alu(a, b, alucontrol, result, zero);

    initial begin
        // set clock to zero
        clk = 0;

        // testcase 1: add a and b. set alucontrol to 010
        a = 'hDEAD0000; b = 'h0000BEEF; alucontrol = 3'b010; #2;
        // result should be 0xdeadbeef

        // wait for 1 clock cycle
        // testcase 2: and a and b. set alucontrol to 000
        a = 'hC0DEBABE; b = 'h0000FFFF; alucontrol = 3'b000; #2;
        // result should be 0x0000babe

        // wait for 1 clock cycle
        // testcase 3: or a and b. set alucontrol to 001
        a = 'hC0DE0000; b = 'h0000BABE; alucontrol = 3'b001; #2;
        // result should be 0xc0debabe

        // wait for 1 clock cycle
        // testcase 4: subtract a and b. set alucontrol to 010
        // logic, negate b and add 1 to get the 2s complement of b
        // then add it to a to get ~b + a or b - a
        a = 'h0000BABE; b = 'hC0DEBABE; alucontrol = 3'b110; #2;
        // result shoule be: 3F220000 (equivalent to 1c0de0000)

    end

    // clock simulator
    // Simulate clock, 2ticks for clock cycle
    // hence 2ns each clock cycle
    always begin
        #1 clk = ~clk;
    end
endmodule
```

2c.



3a.

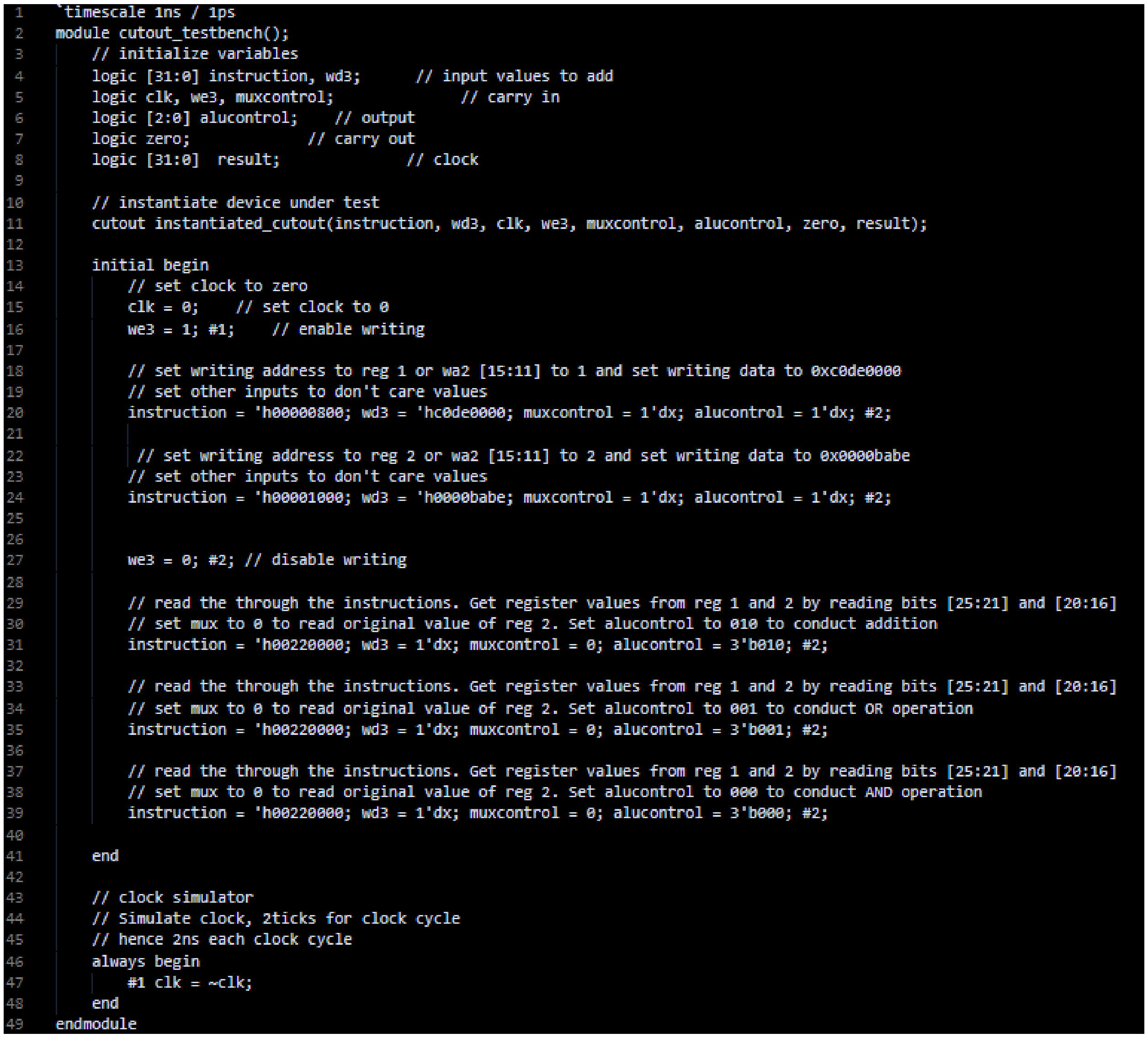


```
`timescale 1ns / 1ps

// These are all the necessary inputs to create cutout
// values were based on the diagram provided
module cutout(input  logic [31:0] instruction, wd3,
              input logic clk, we3, muxcontrol,
              input logic [2:0] alucontrol,
              output logic zero,
              output logic [31:0]  result);

// Additional wires to connect the different modules
logic [31:0] rd1, rd2, y, muxresult;


// instantiate the different modules to be used
// regfile module
regfile instantiated_regfile(clk, we3, instruction[25:21], instruction[20:16], instruction[15:11], wd3, rd1, rd2);

// sign extend module
signext instantiated_signext(instruction[15:0], y);

// 2-way multiplexer module
mux2 #(32) instantiated_mux2(rd2, y, muxcontrol, muxresult);

// alu module
alu instantiated_alu(rd1, muxresult, alucontrol, result, zero);


endmodule
```

3b.

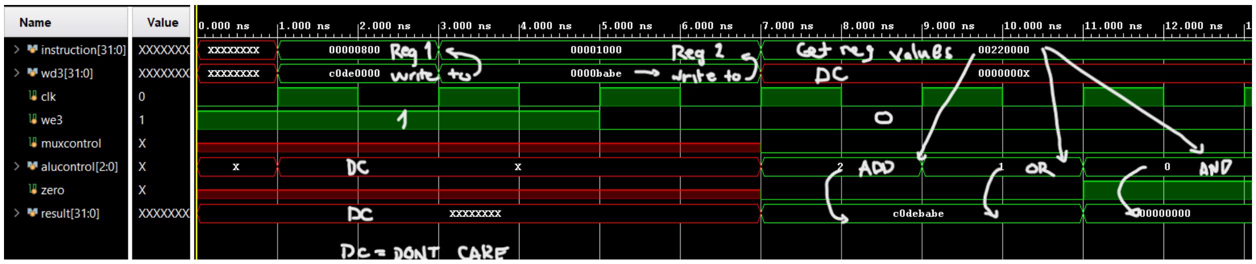
instruction	wd3	we3	muxcontrol	alucontrol
h00200800	c0de0000	1	x	x
h00011000	0000babe	1	x	x
x	x	0	0	b010
x	x	0	0	b001
x	x	0	0	b000

3c.



```
`timescale 1ns / 1ps
module cutout_testbench();
    // initialize variables
    logic [31:0] instruction, wd3;      // input values to add
    logic clk, we3, muxcontrol;              // carry in
    logic [2:0] alucontrol;    // output
    logic zero;            // carry out
    logic [31:0]  result;              // clock

    // instantiate device under test
    cutout instantiated_cutout(instruction, wd3, clk, we3, muxcontrol, alucontrol, zero, result);

    initial begin
        // set clock to zero
        clk = 0;    // set clock to 0
        we3 = 1; #1;    // enable writing

        // set writing address to reg 1 or wa2 [15:11] to 1 and set writing data to 0xc0de0000
        // set other inputs to don't care values
        instruction = 'h00000800; wd3 = 'hc0de0000; muxcontrol = 1'dx; alucontrol = 1'dx; #2;

         // set writing address to reg 2 or wa2 [15:11] to 2 and set writing data to 0x0000babe
        // set other inputs to don't care values
        instruction = 'h00001000; wd3 = 'h0000babe; muxcontrol = 1'dx; alucontrol = 1'dx; #2;


        we3 = 0; #2; // disable writing

        // read the through the instructions. Get register values from reg 1 and 2 by reading bits [25:21] and [20:16]
        // set mux to 0 to read original value of reg 2. Set alucontrol to 010 to conduct addition
        instruction = 'h00220000; wd3 = 1'dx; muxcontrol = 0; alucontrol = 3'b010; #2;

        // read the through the instructions. Get register values from reg 1 and 2 by reading bits [25:21] and [20:16]
        // set mux to 0 to read original value of reg 2. Set alucontrol to 001 to conduct OR operation
        instruction = 'h00220000; wd3 = 1'dx; muxcontrol = 0; alucontrol = 3'b001; #2;

        // read the through the instructions. Get register values from reg 1 and 2 by reading bits [25:21] and [20:16]
        // set mux to 0 to read original value of reg 2. Set alucontrol to 000 to conduct AND operation
        instruction = 'h00220000; wd3 = 1'dx; muxcontrol = 0; alucontrol = 3'b000; #2;

    end

    // clock simulator
    // Simulate clock, 2ticks for clock cycle
    // hence 2ns each clock cycle
    always begin
        #1 clk = ~clk;
    end
endmodule
```

3d.


Name
Value
instruction[31:0]
wd3[31:0]
clk
we3
muxcontrol
alucontrol[2:0]
zero
result[31:0]
XXXXXXX
X
0.000 ns
1.000 ns
2.000 ns
3.000 ns
4.000 ns
5.000 ns
6.000 ns
7.000 ns
8.000 ns
9.000 ns
10.000 ns
11.000 ns
12.000 ns
XXXXXXXX
c0de0000
0000babe
0000000X
x
c0debabe
Reg 1 write to
Reg 2 write to
Get reg values
DC
ADD
OR
AND
DC = DONT CARE