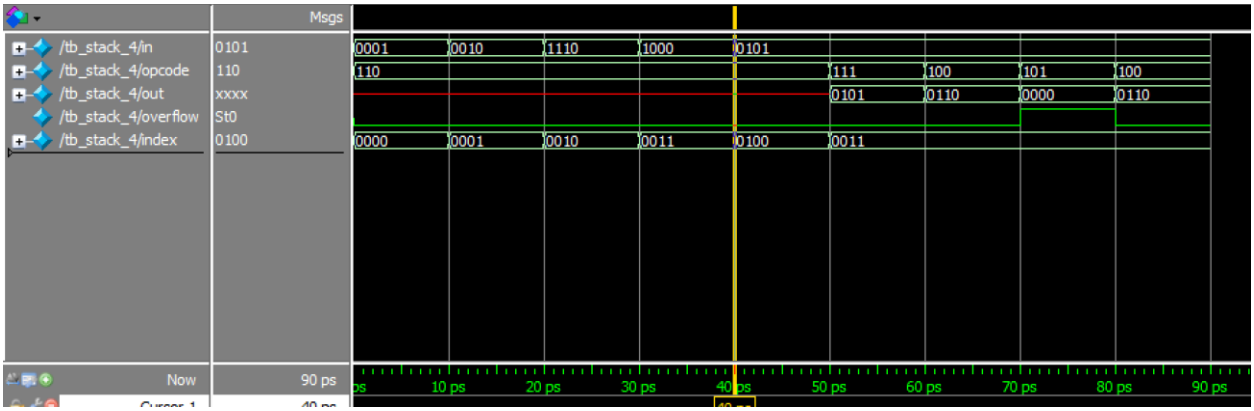
Module stack:

```
module stack #(parameter WIDTH = 8) (
    input wire [WIDTH-1:0] in,
    output reg [WIDTH-1:0] out,
    input wire [2:0] opcode,
    output reg overflow,
    output reg [WIDTH-1:0] index
);

    reg [WIDTH-1:0] stack[0:50];
    reg [WIDTH-1:0]maxvalue ={WIDTH{1'b1}};
    initial index = -1;
    initial overflow=0;

    always @(*) begin
        overflow = 0;
        if (opcode == 4 && index != 0) begin
            out = stack[index] + stack[index-1];
            if ((stack[index] !=0 || stack[index-1] !=0 ) && out == 0) begin
            overflow = 1;
             end
        end
        else if (opcode == 5 && index != 0) begin
            out = stack[index] * stack[index-1];
            if (stack[index] !=0 && stack[index-1] !=0 && out == 0) begin
            overflow = 1;
            end
        end
        else if (opcode == 6 && index != 50) begin
            index = index + 1;
            stack[index] = in;
        end
        else if (opcode == 7 && index != 0) begin
            out = stack[index];
            index = index - 1;
        end
    end

endmodule
```

Test bench 4-bit:

```
module tb_stack_4;
  reg [3:0] in;
  reg [2:0] opcode;
  wire [3:0] out;
  wire overflow;
  wire [3:0]index;
  stack #(4) mystack  (
    .in(in),
    .out(out),
    .opcode(opcode),
    .overflow(overflow),
    .index(index)
  );
  initial begin
    in = 1;
    opcode = 6;
    #10;
    in = 2;
    opcode = 6;
    #10;
    in = 30;
    opcode = 6;
    #10;
    in = 40;
    opcode = 6;
    #10;
    in = 5;
    opcode = 6;
    #10;
    opcode = 7;
    #10;
    opcode = 4;
    #10;
    opcode = 5;
    #10;
    opcode =4;
    #10;
    $display("Final output: %d", out);
    $finish;
```

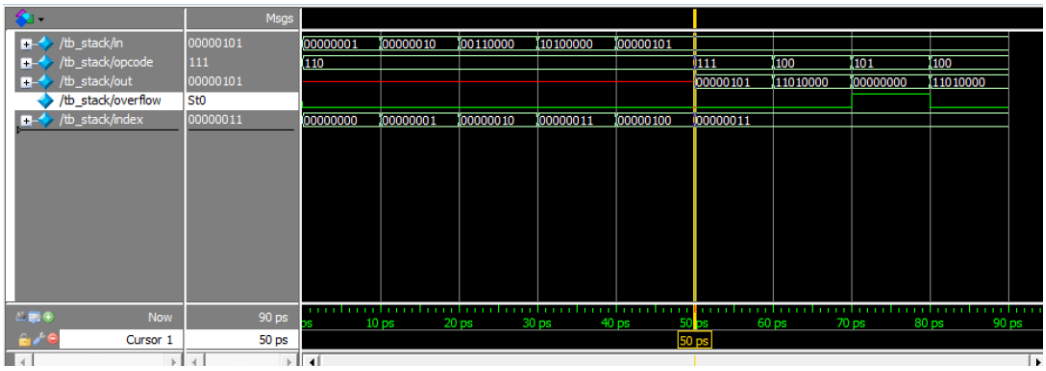
Wave form:



Testbench 8-bit:

```
module tb_stack_8;
  reg [7:0] in;
  reg [2:0] opcode;
  wire [7:0] out;
  wire overflow;
  wire [7:0]index;
  stack #(8) mystack  (
    .in(in),
    .out(out),
    .opcode(opcode),
    .overflow(overflow),
    .index(index)
  );
  initial begin
    in = 1;
    opcode = 6;
    #10;
    in = 2;
    opcode = 6;
    #10;
    in = 30000;
    opcode = 6;
    #10;
    in = 4000;
    opcode = 6;
    #10;
    in = 5;
    opcode = 6;
    #10;
    opcode = 7;
    #10;
    opcode = 4;
    #10;
    opcode = 5;
    #10;
    opcode =4;
    #10;
    $display("Final output: %d", out);
    $finish;
```

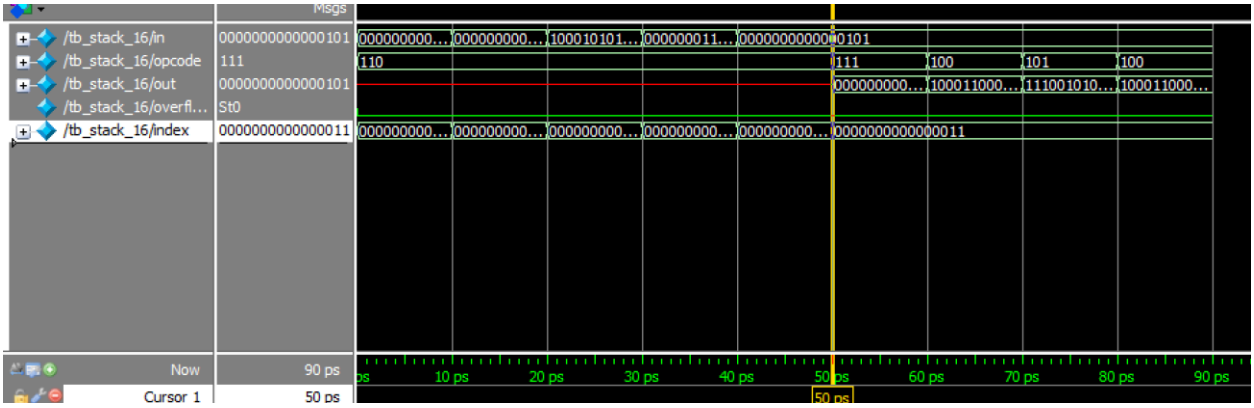
Wave form:



Testbench 16-bit:

```
module tb_stack_16;
  reg [15:0] in;
  reg [2:0] opcode;
  wire [15:0] out;
  wire overflow;
  wire [15:0]index;
  stack #(16) mystack  (
    .in(in),
    .out(out),
    .opcode(opcode),
    .overflow(overflow),
    .index(index)
  );
  initial begin
    in = 1;
    opcode = 6;
    #10;
    in = 2;
    opcode = 6;
    #10;
    in = -30000;
    opcode = 6;
    #10;
    in = 400;
    opcode = 6;
    #10;
    in = 5;
    opcode = 6;
    #10;
    opcode = 7;
    #10;
    opcode = 4;
    #10;
    opcode = 5;
    #10;
    opcode =4;
    #10;
    $display("Final output: %d", out);
    $finish;
```

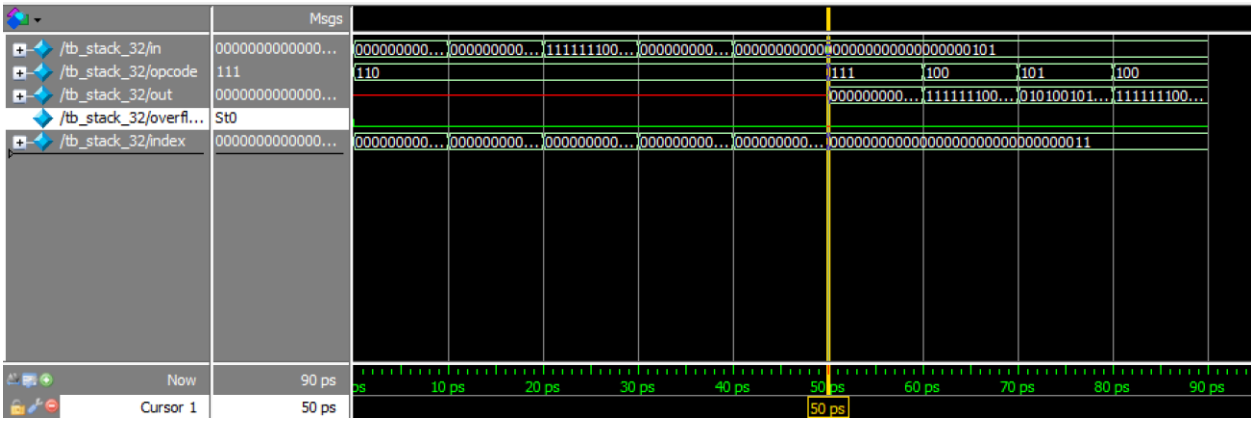
Wave form:



Testbech 32bit:

```
module tb_stack_32;
  reg [31:0] in;
  reg [2:0] opcode;
  wire [31:0] out;
  wire overflow;
  wire [31:0]index;
  stack #(32) mystack  (
    .in(in),
    .out(out),
    .opcode(opcode),
    .overflow(overflow),
    .index(index)
  );
  initial begin
    in = 1;
    opcode = 6;
    #10;
    in = 2;
    opcode = 6;
    #10;
    in = -30000000;
    opcode = 6;
    #10;
    in = 4000000;
    opcode = 6;
    #10;
    in = 5;
    opcode = 6;
    #10;
    opcode = 7;
    #10;
    opcode = 4;
    #10;
    opcode = 5;
    #10;
    opcode =4;
    #10;
    $display("Final output: %d", out);
    $finish;
```

Wave form:



In the `stack` module, we utilize a vector array with a variable size. Within an `always` block, we perform operations such as push, pop, addition, and multiplication based on the specified opcode. To test this module, we set parameters according to the desired bit value. We know that the module's input ports are of type `reg`, and the outputs must be of type `wire`. Then, in an `initial` block, which executes sequentially and only once, we verify test cases that we suspect might be incorrect.

For the second part, the code is not complete, but the process is as follows: First, the infix expression needs to be converted to postfix. I've used two stacks for this purpose. All numbers are entered into one stack, while the + and * operators are placed in another stack. When a + operator needs to be added to the stack, we consider the * operator with higher precedence. In that case, we pop all contents from the stack2 (from top to bottom) and select two numbers from the stack 1. We then perform the operation on those two numbers. For further understanding, please refer to the two additional links provided on the GitHub homepage

Infix to postfix module:

```
`timescale 1ns / 1ps

module postfix(input [7:0]infix,input clk,
                    output reg [7:0]postfix);
```

```
reg [7:0] in,in2 ;
wire [7:0] out,out2;
reg [2:0] opcode,opcode2;
wire overflow,overflow2;
wire [7:0]index,index2;
stack #(8) s1 (in, out,opcode,overflow,index);
stack #(8) s2 (in2, out2,opcode2,overflow2,index2);
integer l=0;
integer k=0;
reg [7:0] temp;

always @ (infix)
begin
    if (infix != "+"&& infix!="*"&&infix!="=")
    begin
      in= infix;
      opcode=6;
      postfix=in;

    end
    else if (infix == "*")
    begin
        in2=infix;
        opcode2=6;
        temp="*";
    end
    else if(infix=="+")
    begin
        if(temp=="*")k=1;


            in2 = infix;
            opcode=6;
        temp="+";
    end
    else if(infix =="=")begin
            k=1;
    end
end

always @(posedge clk)
begin
    if( k==1)
    begin
```

```
        begin
            if(index2>=0)
            begin
                opcode2=7;
                opcode=6;
                in=out2;
                postfix=out2;
            end
            else
            begin
                k=0;
            end
        end

    end
end
endmodule
```

test bench:

```
`timescale 1ns / 1ps


module postfix_tb;

    reg [7:0] infix;

    reg clk;

    wire [7:0] postfix;

    postfix uut (
        .infix(infix),
        .clk(clk),
        .postfix(postfix)

    );

    initial begin

        infix = 0;
        clk = 0;
        #10;
      clk = 1;
        #30 infix = 3;
        #30 infix = "+";
        #30 infix = 4;
        #30 infix="*";
        #30 infix=2;
        #30 infix="+";
        #30 infix=1;
        #30 infix="=";
      #100 $stop;
    end
   always #5 clk = ~clk;
endmodule
```