جبرانی پایانترم

طراحی سیستم‌های دیجیتال

محمد داودآبادی فراهانی
401110331

سوال ۱ میانترم

الف)

در این بخش از سوال ماژول STACK_BASED_ALU را پیاده‌سازی می‌کنیم که در ادامه این پیاده‌سازی را خواهیم دید.

```
module STACK_BASED_ALU #(parameter N = 8) (
    input signed [N-1:0] input_data,
    input [2:0] opcode,
    output reg signed [N-1:0] output_data,
    output reg overflow
);
    // Internal stack for ALU operations
    reg [N-1:0] stack [0:N-1];
    integer top = -1;

    always @(*) begin
        overflow = 0;
        output_data = 0;
        case (opcode)
            3'b100: // Addition
                if (top > 0) begin
                    output_data = stack[top] + stack[top-1];
                    overflow = ((stack[top] > 0 && stack[top-1] > 0 &&
output_data <= 0) || (stack[top] < 0 && stack[top-1] < 0 && output_data >= 0)) ?
1 : 0;
                end
            3'b101: // Multiplication
                if (top > 0) begin
                    output_data = stack[top] * stack[top-1];
                    if (stack[top] == 0 || stack[top-1] == 0) begin
                        overflow = 0;
                    end
                    else begin
                        overflow = (stack[top] == output_data / stack[top-1]) ? 0
: 1;
                    end
                end
            3'b110: // PUSH
                if (top < N-1) begin
                    top = top + 1;
                    stack[top] = input_data;
                end
            3'b111: // POP
                if (top >= 0) begin
```

```
                    output_data = stack[top];
                    top = top - 1;
                end
            default: // No Operation
                output_data = 0;
        endcase
    end
endmodule
```

در این کد همانطور که مشخص است ۵ عمل ضرب و جمع و پوش و پاپ و Nop انجام می‌شود و در حالت جمع و ضرب درصورتی که اورفلو رخ داده باشد گفته می‌شود.

حال تست برای این ماژول می‌نویسیم.

```
`timescale 1ns / 1ps

module TB;
    reg [3:0] input_data_4;
    reg [7:0] input_data_8;
    reg [15:0] input_data_16;
    reg [31:0] input_data_32;
    reg [2:0] opcode;
    wire signed [3:0] output_data_4;
    wire signed [7:0] output_data_8;
    wire signed [15:0] output_data_16;
    wire signed [31:0] output_data_32;
    wire overflow_4, overflow_8, overflow_16, overflow_32;

    // Instantiate the module with N = 4
    STACK_BASED_ALU #(.N(4)) alu_4 (
        .input_data(input_data_4),
        .opcode(opcode),
        .output_data(output_data_4),
        .overflow(overflow_4)
    );

    // Instantiate the module with N = 8
    STACK_BASED_ALU #(.N(8)) alu_8 (
        .input_data(input_data_8),
        .opcode(opcode),
        .output_data(output_data_8),
        .overflow(overflow_8)
```

```
    );

    // Instantiate the module with N = 16
    STACK_BASED_ALU #(.N(16)) alu_16 (
        .input_data(input_data_16),
        .opcode(opcode),
        .output_data(output_data_16),
        .overflow(overflow_16)
    );

    // Instantiate the module with N = 32
    STACK_BASED_ALU #(.N(32)) alu_32 (
        .input_data(input_data_32),
        .opcode(opcode),
        .output_data(output_data_32),
        .overflow(overflow_32)
    );

    initial begin
        // Test the ALU with some operations
        input_data_32 = -32'd1;
        input_data_16 = -16'd1;
        input_data_8 = -8'd1;
        input_data_4 = 4'd7;

        opcode = 3'b110; // PUSH
        #10
        input_data_32 = 32'd2;
        input_data_16 = 16'd2;
        input_data_8 = 8'd2;
        input_data_4 = 4'd1;
        #10
        opcode = 3'b100; // Addition
        #10
        $display("after addition 4 -> output_4: %d_%b, overflow_4: %d_%b\n",
output_data_4, output_data_4, overflow_4, overflow_4);
        $display("after addition 8 -> output_8: %d_%b, overflow_8: %d_%b\n",
output_data_8, output_data_8, overflow_8, overflow_8);
        $display("after addition 16 -> output_16: %d_%b, overflow_16: %d_%b\n",
output_data_16, output_data_16, overflow_16, overflow_16);
        $display("after addition 32 -> output_32: %d_%b, overflow_32: %d_%b\n",
output_data_32, output_data_32, overflow_32, overflow_32);
        opcode = 3'b101; // Multiplication
        #10
```

```
        $display("after multiplication 4 -> output_4: %d_%b, overflow_4:
%d_%b\n", output_data_4, output_data_4, overflow_4, overflow_4);
        $display("after multiplication 8 -> output_8: %d_%b, overflow_8:
%d_%b\n", output_data_8, output_data_8, overflow_8, overflow_8);
        $display("after multiplication 16 -> output_16: %d_%b, overflow_16:
%d_%b\n", output_data_16, output_data_16, overflow_16, overflow_16);
        $display("after multiplication 32 -> output_32: %d_%b, overflow_32:
%d_%b\n", output_data_32, output_data_32, overflow_32, overflow_32);
        opcode = 3'b111; // POP
        #10
        $display("after pop 4 -> output_4: %d_%b, overflow_4: %d_%b\n",
output_data_4, output_data_4, overflow_4, overflow_4);
        $display("after pop 8 -> output_8: %d_%b, overflow_8: %d_%b\n",
output_data_8, output_data_8, overflow_8, overflow_8);
        $display("after pop 16 -> output_16: %d_%b, overflow_16: %d_%b\n",
output_data_16, output_data_16, overflow_16, overflow_16);
        $display("after pop 32 -> output_32: %d_%b, overflow_32: %d_%b\n",
output_data_32, output_data_32, overflow_32, overflow_32);
        opcode = 3'b000; // No Operation
        #10;
        $finish;
    end
endmodule
```

در این تست ۴ instance از ماژول می‌گیریم با سایزهای مورد نیاز و در ۴ بیتی ضرب و جمع را روی ۱ و ۷ انجام می‌دهیم و برای بقیه روی ۲ و منفی ۱ انجام می‌دهیم.

در ابتدا همه‌ی این اعداد گفته شده را روی استک پوش می‌کنیم سپس عملیات ضرب و جمع را انجام می‌دهیم و در آخر یکی از اعداد را پاپ می‌کنیم و یک Nop انجام می‌دهیم تا از صحت عملکرد این دستورها مطمئن شویم.

انتظار داریم که جمع ۱ و ۷ و ضرب ۲ و منفی ۱ اورفلو کند ولی برای باقی حالت‌ها اورفلو رخ ندهد.

خروجی برنامه در عکس زیر مشخص است.

```
VSIM 2> run -all
# after addition 4 -> output_4:  -8_1000, overflow_4: 1_1
#
# after addition 8 -> output_8:    1_00000001, overflow_8: 0_0
#
# after addition 16 -> output_16:      1_0000000000000001, overflow_16: 0_0
#
# after addition 32 -> output_32:           1_00000000000000000000000000000001, overflow_32: 0_0
#
# after multiplication 4 -> output_4:   7_0111, overflow_4: 0_0
#
# after multiplication 8 -> output_8:   -2_11111110, overflow_8: 1_1
#
# after multiplication 16 -> output_16:     -2_1111111111111110, overflow_16: 1_1
#
# after multiplication 32 -> output_32:          -2_11111111111111111111111111111110, overflow_32: 1_1
#
# after pop 4 -> output_4:   1_0001, overflow_4: 0_0
#
# after pop 8 -> output_8:    2_00000010, overflow_8: 0_0
#
# after pop 16 -> output_16:      2_0000000000000010, overflow_16: 0_0
#
# after pop 32 -> output_32:           2_00000000000000000000000000000010, overflow_32: 0_0
#
# ** Note: $finish    : C:/Users/DavoudAbadi/Desktop/MO/Sharif/Term 4/DSD/Extra/TB.v(81)
#    Time: 60 ns  Iteration: 0  Instance: /TB
# 1
```