سوال ۱ میان ترم) STACK_BASED_ALU

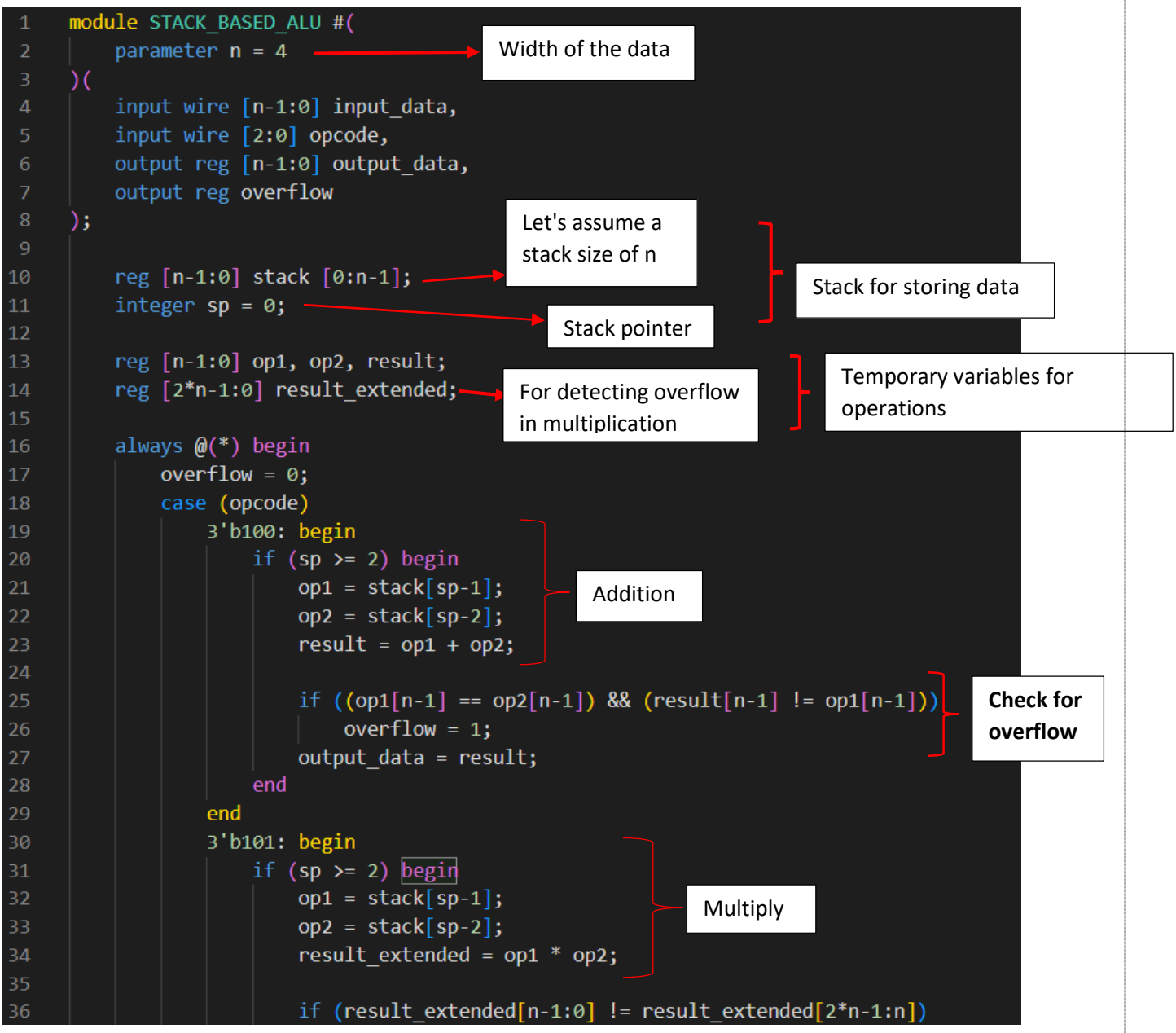
ماژول‌ها:

۱) STACK_BASED_ALU
۲) tb4_STACK_BASED_ALU
۳) tb8_STACK_BASED_ALU
۴) tb16_STACK_BASED_ALU
۵) tb32_STACK_BASED_ALU
۶) InfixToPostfix
۷) PostfixEvaluator
۸) ExpressionEvaluator (testbench برای ماژول ۶ و ۷)

ماژول ۶ تا ۸ برای قسمت ب سوال می‌باشد.

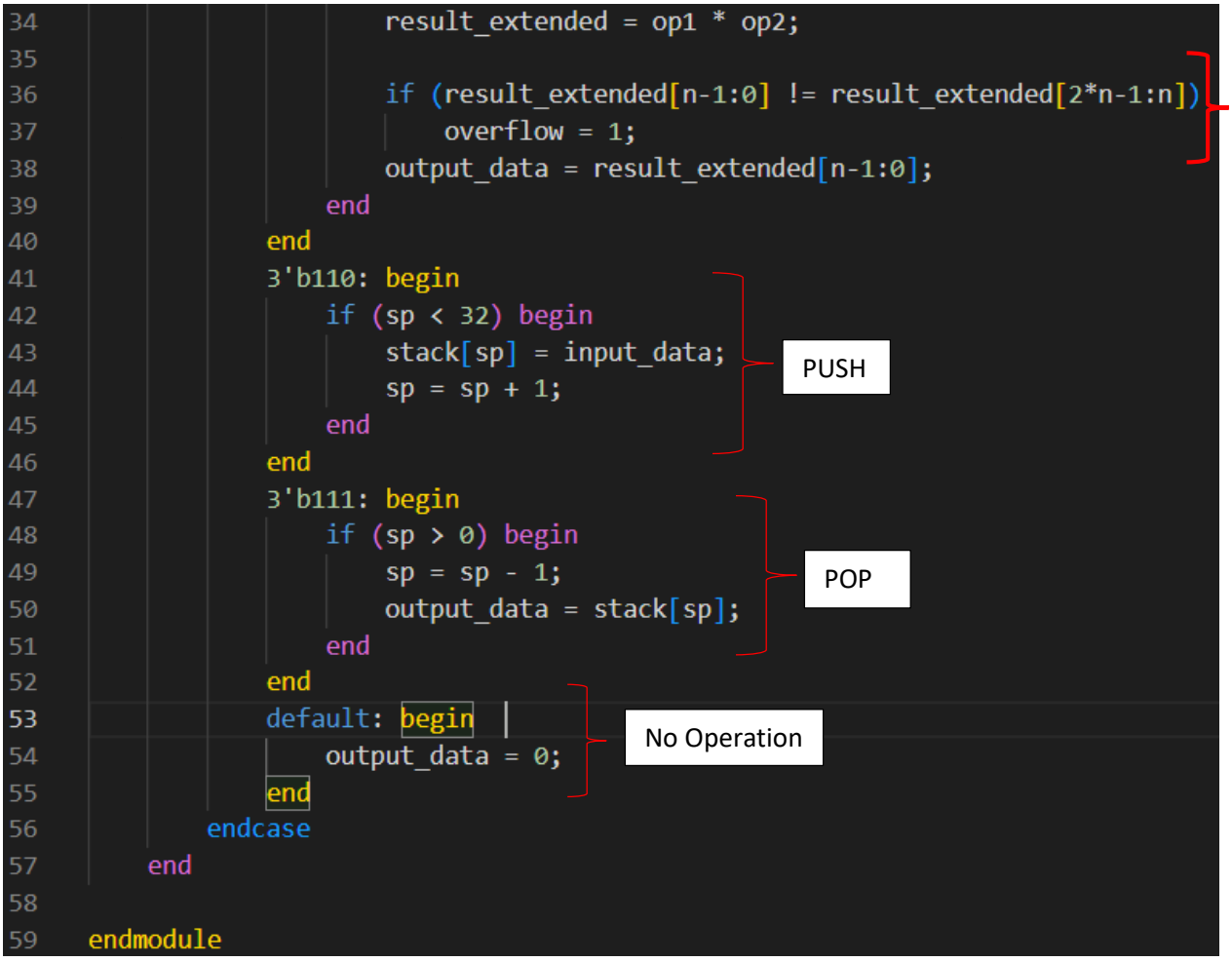
در ادامه داک، کد وریلاگ ماژول ها به ترتیب شماره نوشته شده، شرح داده می‌شوند.

ماژول STACK_BASED_ALU با توضیحات:

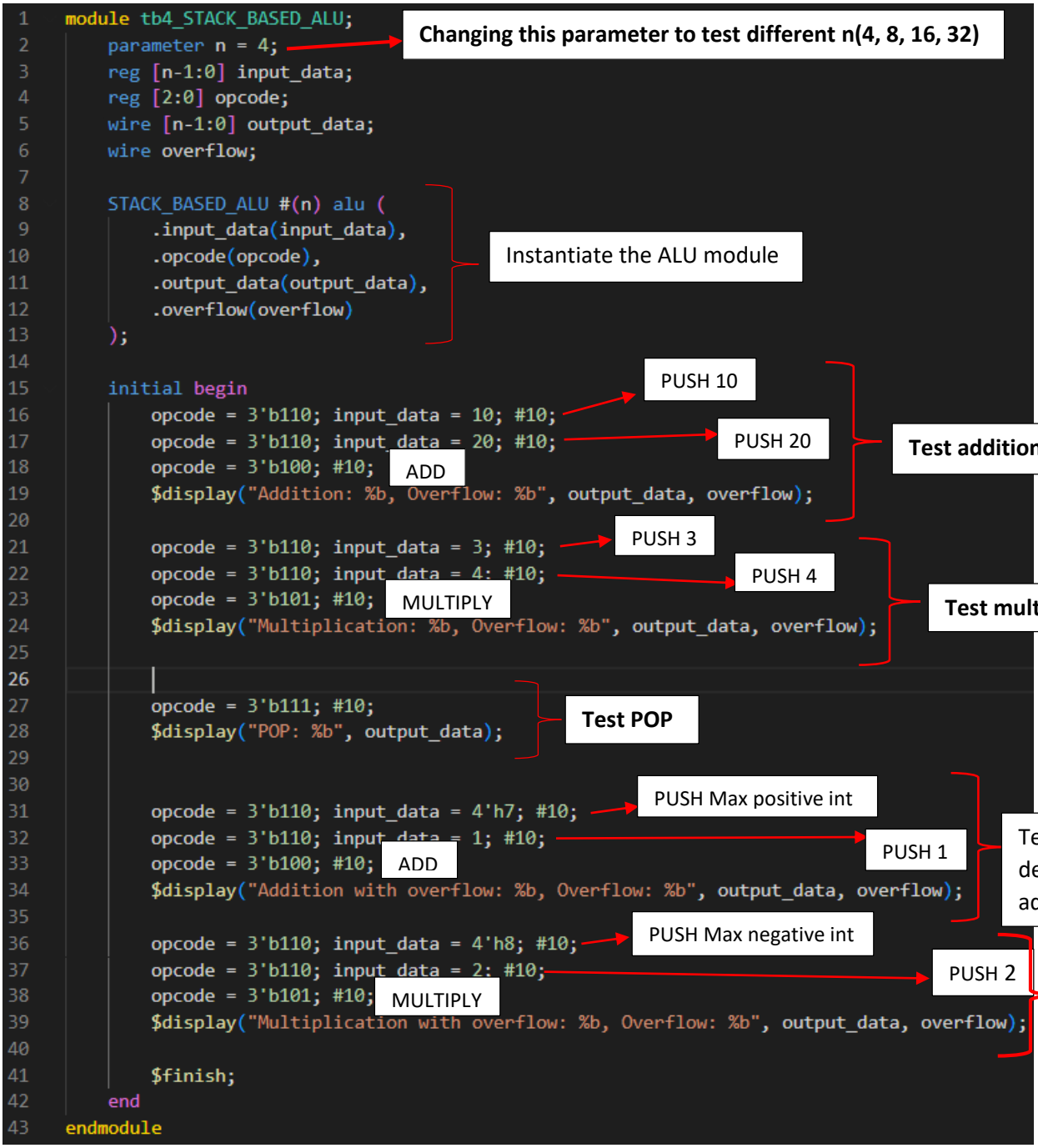



```
1   module STACK_BASED_ALU #(
2       parameter n = 4
3   )(
4       input wire [n-1:0] input_data,
5       input wire [2:0] opcode,
6       output reg [n-1:0] output_data,
7       output reg overflow
8   );
9
10      reg [n-1:0] stack [0:n-1];
11      integer sp = 0;
12
13      reg [n-1:0] op1, op2, result;
14      reg [2*n-1:0] result_extended;
15
16      always @(*) begin
17          overflow = 0;
18          case (opcode)
19              3'b100: begin
20                  if (sp >= 2) begin
21                      op1 = stack[sp-1];
22                      op2 = stack[sp-2];
23                      result = op1 + op2;
24
25                      if ((op1[n-1] == op2[n-1]) && (result[n-1] != op1[n-1]))
26                          overflow = 1;
27                      output_data = result;
28                  end
29              end
30              3'b101: begin
31                  if (sp >= 2) begin
32                      op1 = stack[sp-1];
33                      op2 = stack[sp-2];
34                      result_extended = op1 * op2;
35
36                      if (result_extended[n-1:0] != result_extended[2*n-1:n])
```

ادامه ماژول **STACK_BASED_ALU** :



توضیحات ماژول **testbench** برای ماژول STACK_BASED_ALU :




```
module tb4_STACK_BASED_ALU;
    parameter n = 4;
    reg [n-1:0] input_data;
    reg [2:0] opcode;
    wire [n-1:0] output_data;
    wire overflow;

    STACK_BASED_ALU #(n) alu (
        .input_data(input_data),
        .opcode(opcode),
        .output_data(output_data),
        .overflow(overflow)
    );

    initial begin
        opcode = 3'b110; input_data = 10; #10;
        opcode = 3'b110; input_data = 20; #10;
        opcode = 3'b100; #10;
        $display("Addition: %b, Overflow: %b", output_data, overflow);

        opcode = 3'b110; input_data = 3; #10;
        opcode = 3'b110; input_data = 4; #10;
        opcode = 3'b101; #10;
        $display("Multiplication: %b, Overflow: %b", output_data, overflow);

        opcode = 3'b111; #10;
        $display("POP: %b", output_data);


        opcode = 3'b110; input_data = 4'h7; #10;
        opcode = 3'b110; input_data = 1; #10;
        opcode = 3'b100; #10;
        $display("Addition with overflow: %b, Overflow: %b", output_data, overflow);

        opcode = 3'b110; input_data = 4'h8; #10;
        opcode = 3'b110; input_data = 2; #10;
        opcode = 3'b101; #10;
        $display("Multiplication with overflow: %b, Overflow: %b", output_data, overflow);

        $finish;
    end
endmodule
```

نمایش خروجی در transcript و waveform:

✓ **(parameter n = 4) tb4_STACK_BASED_ALU**

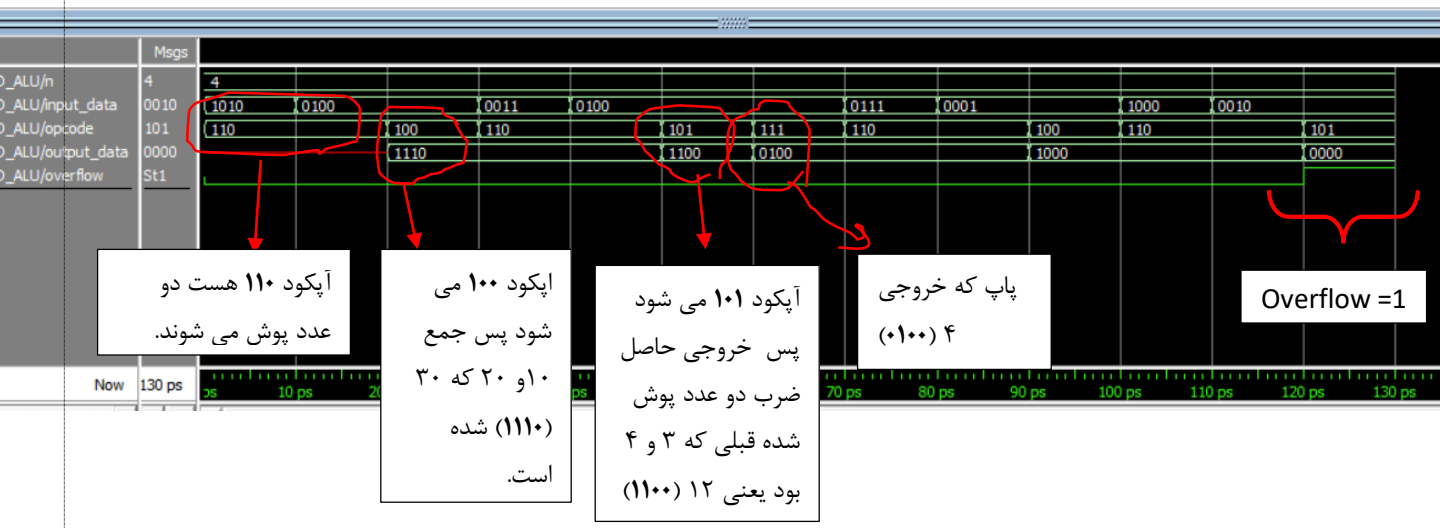
```
# Addition: 1110, Overflow: 0
# Multiplication: 1100, Overflow: 0
# POP: 0100
# Addition with overflow: 1000, Overflow: 0
# Multiplication with overflow: 0000, Overflow: 1
# ** Note: $finish    : E:/ATEFE-UNIVERSITY/Q1_STACK_BASED_ALU/tb4_STACK_BASED_ALU.v(41)
#    Time: 130 ps  Iteration: 0  Instance: /tb4_STACK_BASED_ALU
```

۱) طبق ماژول تست بنچ که در صفحه پیش جز به جز توضیح داده شد، در ابتدا مقدار opcode، 110 است که دو مقدار 10و20 در استک پوش میشوند، سپس مقدار opcode، 100 میشود که یعنی جمع کردن دو مقدار بالا پشته، پس همانطور که میبینیم در تست کیس اول حاصل30 شده است. (این نکته که در سوال آمده رعایت شده "عملوندهای Opcode ضرب و جمع دو عدد بالایی پشته است")

۲) سپس مجدد opcode، 110 میشود و 3 و 4 در پشته پوش میشوند، حال opcode، 101 میشود که باید دو مقدار بالا پشته را در هم ضرب کند که حاصل در خط دوم transcript، 1100 = 12 شده است.

۳) حال opcode، 111 میشود که یعنی پاپ کردن از پشته، با توجه به اینکه در مرحله قبل ابتدا 3 پوش شد و سپس 4 این به این معنا است که بالاترین داده در استک 4 هست، همانطور که مشاهده می‌شود مقداری نیز که پاپ شده 100 = 4 است. (پس این نکته سوال که " نتیجه عملیات ضرب/جمع در خروجی ماژول در دسترس خواهند بود و تغییری در پشته ایجاد نخواهند کرد." رعایت شده است. چون در مرحله قبل ۳ را در ۴ ضرب کردیم و حالا که اپکود ۱۱۱ شد همان مقدار ۴ که در پشته بود پاپ شد ← تغییری در پشته ایجاد نشد.)

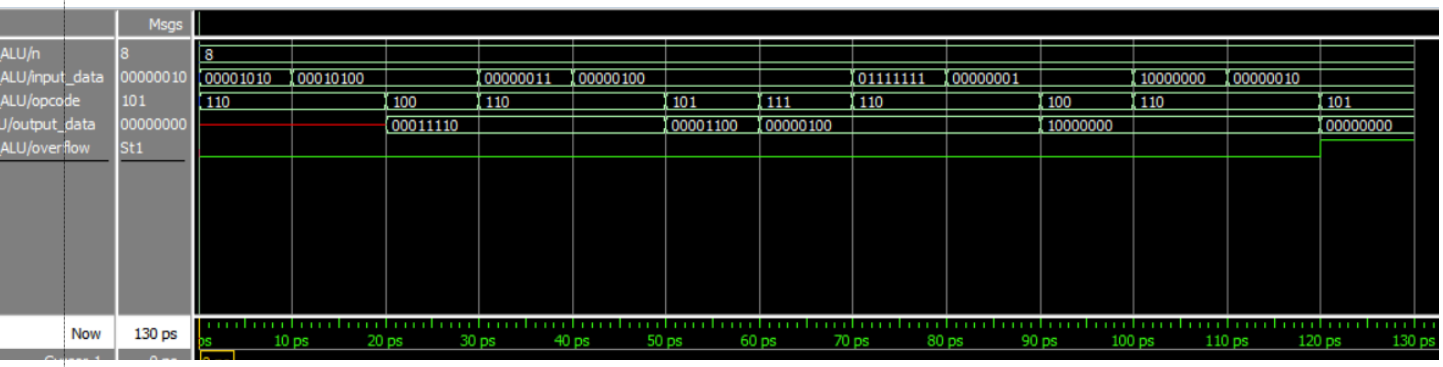
۴) خط ۴و۵ در transcript نیز چک کردن اورفلو هست که همانطور که پیداست هنگام ضرب چون بیشترین اینتیجر منفی در ۲ ضرب میشود اورفلو داریم که مقدار overflow ، 1 شده است.



(parameter n = 8) tb8_STACK_BASED_ALU ✓

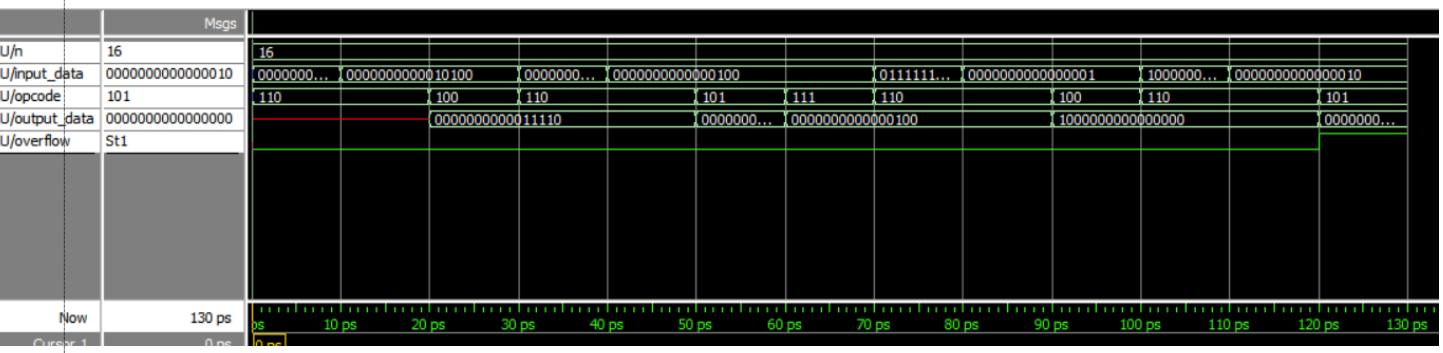
```
# Addition: 00011110, Overflow: 0
# Multiplication: 00001100, Overflow: 0
# POP: 00000100
# Addition with overflow: 10000000, Overflow: 0
# Multiplication with overflow: 00000000, Overflow: 1
# ** Note: $finish    : E:/ATEFE-UNIVERSITY/Q1_STACK_BASED_ALU/tb8_STACK_BASED_ALU.v(41)
#    Time: 130 ps  Iteration: 0  Instance: /tb8_STACK_BASED_ALU
```

توضیحات و عملکرد عینا همانند خروجی های قبلی است (هنگامی که پارامتر ۴ است) تنها خروجی ها ۸ بیتی هستند.



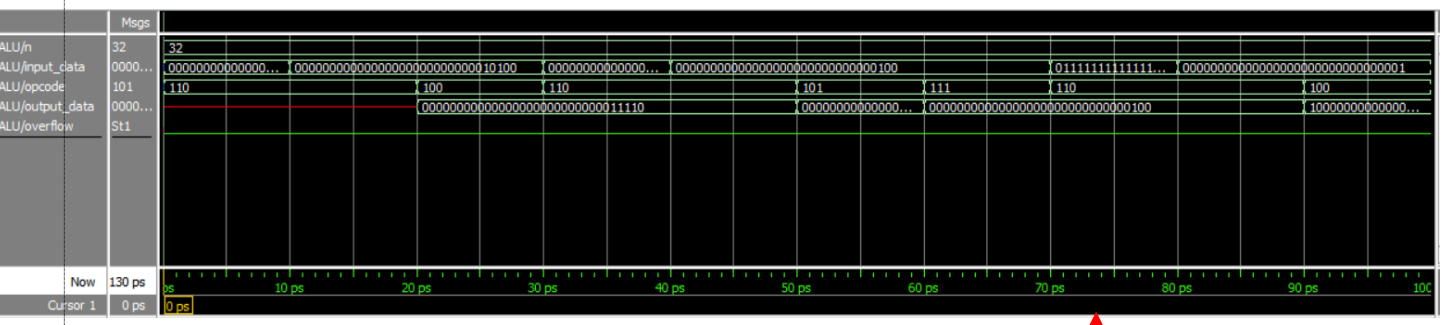
(parameter n = 16) tb16_STACK_BASED_ALU ✓

```
# Addition: 0000000000011110, Overflow: 0
# Multiplication: 0000000000001100, Overflow: 0
# POP: 0000000000000100
# Addition with overflow: 1000000000000000, Overflow: 0
# Multiplication with overflow: 0000000000000000, Overflow: 1
# ** Note: $finish    : E:/ATEFE-UNIVERSITY/Q1_STACK_BASED_ALU/tb16_STACK_BASED_ALU.v(41)
#    Time: 130 ps  Iteration: 0  Instance: /tb16_STACK_BASED_ALU
```



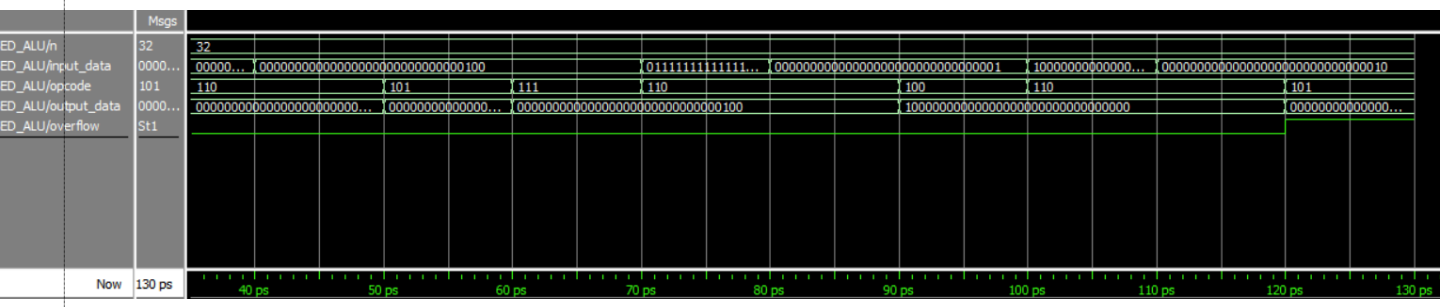
✓ **(parameter n = 32) tb32_STACK_BASED_ALU**

```
# Addition: 00000000000000000000000000011110, Overflow: 0
# Multiplication: 00000000000000000000000000001100, Overflow: 0
# POP: 00000000000000000000000000000100
# Addition with overflow: 10000000000000000000000000000000, Overflow: 0
# Multiplication with overflow: 00000000000000000000000000000000, Overflow: 1
# ** Note: $finish    : E:/ATEFE-UNIVERSITY/Q1_STACK_BASED_ALU/tb32_STACK_BASED_ALU.v(41)
#    Time: 130 ps  Iteration: 0  Instance: /tb32_STACK_BASED_ALU
```



Waveform تا ps۱۰۰

Waveform از ps۱۰۰ به بعد :



قسمت ب)

برای نوشتن یک ماژول که بتواند عبارات را به پسوندی تبدیل کرده و سپس محاسبه کند، ابتدا باید دو مرحله را انجام دهیم:

۱. تبدیل عبارت میانوندی (infix) به عبارت پسوندی.(postfix)

۲. استفاده از ماژول STACK_BASED_ALUبرای محاسبه نتیجه عبارت پسوندی.

ابتدا یک ماژول برای تبدیل عبارت میانوندی به پسوندی نوشته می‌شود. سپس با استفاده از یک ماژول دیگر، محاسبه نتیجه عبارت پسوندی انجام می‌گردد.

ماژول تبدیل عبارت میانوندی به پسوندی InfixToPostfix

در این قسمت از یک الگوریتم مانند الگوریتم شانتینگ-یارد (Shunting Yard) برای تبدیل عبارت میانوندی به پسوندی استفاده می‌کنیم.

جزئیات بیشتر ماژول به صورت کامنت گذاری در کد:

```
module InfixToPostfix (
    input [2047:0] infix, // Input infix expression, maximum length 256 characters (8 bits each)
    output reg [2047:0] postfix // Output postfix expression
);
    reg [7:0] stack [0:255];
    integer sp;
    integer i, j;
    reg [7:0] current_char;
    reg done; // Variable to manage loop exit

    initial begin
        sp = 0;  // Initialize stack pointer
        j = 0;  // Initialize output index
        done = 0;
        for (i = 0; i < 256 && !done; i = i + 1) begin
            current_char = infix[i*8 +: 8];
            if (current_char == 8'd0) done = 1;
            else begin
                case (current_char)
                    8'd40: begin  // '('
                        stack[sp] = current_char;
                        sp = sp + 1;
                    end
                    8'd41: begin  // ')'
                        while (sp > 0 && stack[sp-1] != 8'd40) begin
                            postfix[j*8 +: 8] = stack[sp-1];
                            sp = sp - 1;
                            j = j + 1;
                        end
                        sp = sp - 1; // Pop '('
                    end
                    8'd42, 8'd43, 8'd45: begin  // '*', '+', '-'
                        while (sp > 0 && stack[sp-1] != 8'd40) begin
                            postfix[j*8 +: 8] = stack[sp-1];
                            sp = sp - 1;
                            j = j + 1;
                        end
```

ادامه ماژول InfixToPostfix

```
35                                  sp = sp - 1;
36                                  j = j + 1;
37                              end
38                              stack[sp] = current_char;
39                              sp = sp + 1;
40                          end
41                          default: begin  // Operand
42                              postfix[j*8 +: 8] = current_char;
43                              j = j + 1;
44                          end
45                      endcase
46                  end
47              end
48
49              while (sp > 0) begin
50                  postfix[j*8 +: 8] = stack[sp-1];
51                  sp = sp - 1;
52                  j = j + 1;
53              end
54              postfix[j*8 +: 8] = 8'd0; // Null terminate the postfix expression
55      end
56  endmodule
```

ماژول محاسبه عبارت پسوندی PostfixEvaluator

در این قسمت از ماژول STACK_BASED_ALUبرای محاسبه نتیجه عبارت پسوندی استفاده می‌شود.

جزئیات بیشتر ماژول به صورت کامنت گذاری در کد:

```
module PostfixEvaluator (
    input [2047:0] postfix, // Input postfix expression, maximum length 256 characters (8 bits each)
    output reg [31:0] result, // Output result
    output reg overflow // Overflow flag
);
    reg [31:0] input_data;
    reg [2:0] opcode;
    wire [31:0] output_data;
    wire alu_overflow;

    // Instantiate the ALU module
    STACK_BASED_ALU #(32) alu (
        .input_data(input_data),
        .opcode(opcode),
        .output_data(output_data),
        .overflow(alu_overflow)
    );

    reg [31:0] stack [0:255];
    integer sp;
    integer i;
    reg [7:0] current_char;
    reg done; // Variable to manage loop exit

    initial begin
        sp = 0;
        overflow = 0;
        done = 0;
        for (i = 0; i < 256 && !done; i = i + 1) begin
            current_char = postfix[i*8 +: 8];
            if (current_char == 8'd0) done = 1;
            else begin
                case (current_char)
                    8'd42: begin  // '*'
                        opcode = 3'b101;
                        #10; // Wait for ALU operation
                        stack[sp-2] = output_data;
                        sp = sp - 1;
```

ادامه ماژول **PostfixEvaluator**:

```
                    stack[sp-2] = output_data;
                    sp = sp - 1;
                end
                8'd43: begin  // '+'
                    opcode = 3'b100;
                    #10; // Wait for ALU operation
                    stack[sp-2] = output_data;
                    sp = sp - 1;
                end
                8'd45: begin  // '-'
                    // Subtraction is not defined in the provided ALU, assuming a workaround using addition of negative numbers
                    opcode = 3'b110;
                    input_data = -stack[sp-1];
                    #10; // Push negative of the top value
                    opcode = 3'b100;
                    #10; // Perform addition
                    stack[sp-2] = output_data;
                    sp = sp - 1;
                end
                default: begin  // Operand
                    opcode = 3'b110;
                    input_data = current_char - 8'd48; // Convert ASCII to integer
                    #10; // Wait for ALU operation
                    stack[sp] = input_data;
                    sp = sp + 1;
                end
            endcase
        end
    end
    result = stack[sp-1];
    overflow = alu_overflow;
  end
endmodule
```

ماژول ترکیبی ExpressionEvaluator

ماژول نهایی که از دو ماژول قبلی استفاده می‌کند.

جزئیات بیشتر ماژول به صورت کامنت گذاری در کد:

```
module ExpressionEvaluator;
    reg [2047:0] infix;
    wire [2047:0] postfix;
    wire [31:0] result;
    wire overflow;

    // Instantiate InfixToPostfix module
    InfixToPostfix itp (
        .infix(infix),
        .postfix(postfix)
    );

    // Instantiate PostfixEvaluator module
    PostfixEvaluator pe (
        .postfix(postfix),
        .result(result),
        .overflow(overflow)
    );

    initial begin
        // Initialize infix expression: (5 + 6) + 20 - (3 + 4 + 10) * 3 + 2
        infix = "5 6 + 20 + 3 4 + 10 + * 3 2 * -";
        #100; // Wait for the evaluation to complete

        // Display the result
        $display("Result: %d, Overflow: %b", result, overflow);
    end
endmodule
```